THESE

présentée à l'
INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

pour obtenir le grade de
DOCTEUR D'ETAT ES SCIENCES
Spécialité INFORMATIQUE

par

Radhia COUSOT

FONDEMENTS DES METHODES DE PREUVE D'INVARIANCE ET DE FATALITE DE PROGRAMMES PARALLELES

Thèse soutenue le 15 novembre 1985 devant le jury :

Président :	C. Pair	Rapporteur
Examinateurs :	J.P. Jouannaud	
	G. Roucairol	Rapporteur extérieur
	M. Sintzoff	Rapporteur extérieur
	J.P. Verjus	

THESE

présentée à l'
INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

pour obtenir le grade de
DOCTEUR D'ETAT ES SCIENCES
Spécialité INFORMATIQUE

par

Radhia COUSOT

FONDEMENTS DES METHODES DE PREUVE D'INVARIANCE ET DE FATALITE DE PROGRAMMES PARALLELES

Thèse soutenue le 15 novembre 1985 devant le jury :

Président	: C. Pair	Rapporteur
Examinateurs	: J.P. Jouannaud	
	G. Roucairol	Rapporteur extérieur
	M. Sintzoff	Rapporteur extérieur
	J.P. Verjus	

à

Patrick, mes fils Laurent et Thibault

mes Parents, mes Frères et Sœurs

mes Beaux-Parents

Mon affectueuse gratitude à Monsieur Patrick COUSOT, Professeur d'Informatique à l'Ecole Polytechnique, avec qui j'ai fait ce travail, avec mon admiration pour sa compétence, son exigeance et sa rigueur scientifique.

Je remercie vivement,

Monsieur Claude PAIR, Professeur d'Informatique à l'Institut National Polytechnique de Lorraine, qui m'a fait l'honneur d'accepter, malgré ses nombreuses tâches, de diriger mon travail en tant que Directeur de thèse et Directeur de Recherche auprès du CNRS. Ses encouragements, ses idées synthétiques, sa lecture et sa critique détaillée, précise, pertante des divers articles sur lesquels est basée cette thèse et du manuscrit ont eu une grande influence sur le fond et la forme de ce travail.

Monsieur Michel SINTZOFF, Professeur d'Informatique à l'Université Catholique de Louvain, qui a toujours encouragé ce travail et qui après avoir lu en détail et annoté le manuscrit avec beaucoup de talent, de compétence et autant d'humour, me fait l'honneur de juger cette thèse.

Messieurs Jean-Pierre JOUANNAUD, Professeur d'Informatique à l'Université de Nancy II, Gérard ROUCAIROL, Professeur d'Informatique à l'Université de Paris XI et Jean-Pierre VERJUS, Professeur d'Informatique à l'Université de Rennes qui me font l'honneur de juger ce travail.

Madame SEGAL, de l'Ecole des Mines de Nancy, Mesdames du Service de Scolarité de l'INPL, pour leur obligeance et leur efficacité.

Radhia COUSOT-REZIG

FONDEMENTS DES METHODES DE PREUVE D'INVARIANCE ET DE FATALITE DE PROGRAMMES PARALLELES

Radhia COUSOT

1. INTRODUCTION

1. INTRODUCTION

1.1 BREF HISTORIQUE DE NOTRE DEMARCHE

1.2 MOTIVATION ESSENTIELLE ET COMPARAISON AVEC D'AUTRES APPROCHES

1.3 RESUME SUCCINT ET PLAN DE LA THESE

1.3.1 NOTATIONS

1.3.2 SEMANTIQUE OPERATIONNELLE

1.3.3 PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

1.3.4 PREUVES D'INVARIANCE

1.3.5 PREUVES DE FATALITE

1.3.6 CONCLUSION ET REFERENCES

1. INTRODUCTION

1.1 BREF HISTORIQUE DE NOTRE DEMARCHE

La motivation de ce travail remonte à nos premières réflexions en 1975 concernant les techniques de mise au point des programmes. Partant du problème de la détermination automatique de jeux d'essais, et après l'avoir généralisé, nous avons étudié les problèmes d'analyse sémantique des programmes (c'est-à-dire la détermination statique (à la compilation) de propriétés dynamiques (à l'exécution) des programmes). Plutôt que de chercher à établir un catalogue de méthodes d'analyse, nous avons préféré étudier comment construire une méthode d'analyse sémantique quelconque à partir d'une méthode de preuve en appliquant des techniques d'approximation (cf. 4.3.2.5). Ceci nous a conduit depuis 1979-80 à nous intéresser presque exclusivement aux méthodes de preuve. Comme nous avions du mal à comprendre très profondément les méthodes existantes, nous avons cherché à les formuler de manière aussi concise et rigoureuse que possible de façon à pouvoir les comparer et les généraliser. Il se pose alors très vite le problème de la justification puis de la construction d'une méthode de preuve à partir d'une sémantique et donc celui du choix de la méthode de définition de la sémantique des programmes. Ceci nous amène à l'étude des sémantiques de programmes qui est le point de départ de cette thèse, nous étudions ensuite les méthodes de preuve de propriétés d'invariance et de fatalité de programmes binaires, séquentiels, nondéterministes ou parallèles.

1.2 MOTIVATION ESSENTIELLE ET COMPARAISON AVEC D'AUTRES APPROCHES

Il y a de très nombreuses façons d'aborder le problème des preuves de programmes et nous n'avons pas pu, ni voulu explorer toutes les voies de recherche possibles. Il nous semble que tous les problèmes relatifs aux preuves de programmes ne pourront faire de progrès significatifs que si les méthodes de preuve sont mieux comprises qu'elles ne le sont actuellement. Nous avons donc cherché à établir les fondements des méthodes de preuve de programmes ce qui nous a contraint à faire des choix voire des impasses :

- Avant de démontrer une propriété d'un programme il faut spécifier cette propriété. Nous n'étudierons pas le problème de la spécification de propriétés de programmes et nous ne considérerons que deux classes de propriétés à savoir l'invariance conditionnelle et la fatalité sous invariance.

- Les preuves ne sont pas encore entrées dans la pratique des programmeurs. Elles ne sont généralement appliquées formellement que sur de très petits programmes (quelques lignes) ou informellement sur de petits programmes (quelques pages). Il serait donc intéressant d'essayer de les appliquer sur des programmes moyens ou grands (quelques centaines voire milliers de pages) pour tirer de ces expériences des enseignements pratiques. Nos exemples sont toujours très courts et n'ont évidemment pas cet objectif, ils ne servent qu'à illustrer des notions développées. Parmi les difficultés pratiques qui rebutent les programmeurs, il y a le problème de la taille des preuves, de manque de temps, d'expérience et d'entraînement mais également de manque de connaissances (souvent réduites aux méthodes de Floyd et Dijkstra). Notre

contribution dans ce domaine vise plutôt à essayer d'élargir la panoplie des outils disponibles.

- Même pour des programmes simples, les preuves peuvent être difficiles et parfois complexes. Ceci conduit à l'idée que les preuves ne pourront être faites de manière efficace et rigoureuse que si ce sont les ordinateurs qui les font. Cette voie a beaucoup été explorée aux Etats-Unis sans rencontrer les succès espérés, à cause principalement des déficiences des démonstrateurs de théorèmes. Comme la part d'invention nécessaire pour faire des inductions est relativement faible au regard des très nombreuses conditions souvent simples qu'il faut vérifier, l'idée d'outils de preuve semi-automatiques est séduisante : l'ordinateur n'est plus utilisé pour faire mais pour aider à faire la preuve. On pourra penser à un simple aide-mémoire pour guider une preuve faite à la main. De tels systèmes interactifs sont généralement beaucoup plus ambitieux et cherchent à réduire les interventions nécessaires de l'utilisateur. Celui-ci intervient en programmant diverses stratégies possibles qui peuvent être utilisées pour tenter de faire la preuve ou bien de manière conversationnelle pour apporter la part d'invention nécessaire pour faire les inductions. Malheureusement, en cas d'échec de la preuve, les causes de l'échec sont difficilement présentables de manière synthétique à l'utilisateur. De ce fait, l'expérience a montré que les résultats deviennent vite incompréhensibles pour des programmes dépassant quelques lignes. Nous n'avons pas cherché à étudier un quelconque système d'aide à la preuve. Il nous semble que les systèmes existants ont bien souvent le défaut d'être basés sur une vaste méthode qui ne marche évidemment pas à coup sûr de même qu'en mathématiques un raisonnement peut être beaucoup plus simple ou d'une autre nature s'il sont équivalents c'est-à-dire

qu'il est possible de passer formellement de l'un à l'autre). Notre apport dans ce domaine est donc essentiellement de présenter de manière uniforme des méthodes apparemment dissemblables. Dans le futur, ceci pourrait encourager à l'emploi d'une combinaison de méthodes plutôt que d'une seule.

- Notre formalisme pour étudier les méthodes de preuve est de nature sémantique plutôt que syntaxique. Il fait appel aux modèles ensemblistes et non aux systèmes logiques formels. Ceci va un peu à l'opposé des nombreux travaux actuels qui cherchent à formaliser les méthodes de preuve à l'aide de logiques de toutes sortes (algorithmiques, dynamiques, modales, temporelles, ...). Ces travaux reposent sur l'idée que l'utilisation de systèmes formels permettra une automatisation plus facile des preuves, (ce qui reste à démontrer). L'utilisation de systèmes formels introduit des problèmes supplémentaires (comme l'incomplétude syntaxique) qu'il nous semble utile d'éliminer dans un premier temps (par exemple pour ne pas cacher, quand il se pose, le problème plus fondamental de l'incomplétude sémantique). De plus ces logiques ne sont pas indépendantes du langage auquel elles s'appliquent et sont donc peut être à la fois beaucoup moins concises, moins abstraites et moins compréhensibles que la formalisation que nous proposons. Enfin, ces logiques d'emploi assez lourd nous semblent mal adaptées pour des preuves informelles qui semblent inévitables en pratique. Il reste que notre travail devrait pouvoir trouver son utilité comme base pour étudier ces logiques (du moins celles ayant trait à l'invariance et la fatalité).

- Comme les programmes sont difficiles à comprendre (et donc à prouver) une fois qu'ils sont terminés il semble que l'étude de

soit plus prometteuse que celle des preuves de correction à posteriori. En suivant cette démarche il reste qu'à chaque étape de la construction, il faut démontrer des propriétés relatives à cette étape. Pour ce faire, les méthodes de preuve de propriétés de programmes sont indispensables (et ce d'autant plus que nous proposons une notion abstraite du comportement d'un programme). D'autre part lors du passage d'une étape à la suivante (par exemple par transformation) un certain nombre de propriétés doivent être conservées sans que les preuves soient à refaire. Notre contribution dans ce domaine concerne l'étude de relations entre sémantiques qui conservent les propriétés d'invariance et de fatalité. Il s'agit bien évidemment de quelques résultats techniques et nous n'avons pas l'ambition de proposer une méthodologie de la programmation par transformation de programmes.

- Notre étude adopte un point de vue endogène c'est-à-dire que le comportement du programme est supposé complètement donné sans rien connaître de ce qui lui est extérieur : ce point de vue s'oppose au point de vue exogène où on s'intéresse seulement à une composante du programme (sous-programme, module, processus, etc.) dont le comportement dépend de causes externes et agit sur l'extérieur (dont la connaissance est en général imparfaite). Cette distinction ne nous paraît pas essentielle dans la mesure où l'état du système peut englober l'état du programme et la partie de l'état de l'extérieur ayant un intérêt pour la preuve.
La sémantique du système n'étant pas raisonnablement close, on peut étudier des propriétés de systèmes dont l'évolution ne dépend pas uniquement de leur état courant mais aussi d'un aléa extérieur (et donc de causes extérieures inconnues et imprévues). Dans le même ordre d'idées, il faut une nouvelle considération des preuves

relatives à un programme donné, nous conduit à étudier la façon de décomposer cette preuve en fonction de la structure (des états ou des actions) du programme mais nous n'avons pas étudié le problème dual qui consiste à étudier comment la composition d'un programme à partir de parties induit une construction de la preuve du programme par composition des preuves des parties, (exemple de la méthode de Hoare).

Ayant brièvement présenté les voies de recherche qu'il nous aurait été possible d'explorer mais que nous n'avons pas choisies, nous résumons succintement maintenant le contenu de cette thèse.

1.3 RESUME SUCCINT ET PLAN DE LA THESE

1.3.1 NOTATIONS

Nous utiliserons évidemment des notations mathématiques classiques (concernant la logique, les ensembles, les ordres, les ordinaux, les séquences et les cardinaux) qui sont résumées dans l'annexe I. Il est préférable de commencer par lire cette annexe, mais pour l'éviter nous donnons en annexe II un index des notations mathématiques qui pourra être consulté au fur et à mesure des besoins. Les notations informatiques sont introduites au fil du texte et résumées dans un index donné en annexe III.

1.3.2 SEMANTIQUE OPERATIONNELLE

Le chapitre 2 est consacré à l'étude de la sémantique opérationnelle des programmes.

Toutes les méthodes de preuve de programmes utilisent la notion de "pas de programme". Ceci conduit donc naturellement à formaliser la sémantique d'un programme par un système de transition (cf. 2.2) formé par un ensemble d'états (en général couple état mémoire - état contrôle), un ensemble d'actions, une caractérisation des états initiaux et une relation de transition qui pour toute action "a" caractérise des pairs d'états "s" et "s'" tels que par exécution de l'action "a" on peut passer de l'état "s" dans l'état "s'".

La sémantique engendrée par ce système de transition (cf. 2.4) est [illegible]

ce système de transition. Une trace est une suite d'états séparés par des actions (conformément à la relation de transition) qui commence par un état initial et est infinie ou bien se termine par un état de blocage (sans successeur possible). Les traces représentent donc un calcul fini et achevé ou bien un calcul infini mais jamais un calcul en cours.

Malheureusement, toutes les sémantiques de programmes ne sont pas directement engendrées par un système de transition. Par exemple, les sémantiques de programmes parallèles équitables ne le sont pas car l'évolution du calcul ne dépend pas uniquement de l'état courant (qui ne contient pas toutes les informations nécessaires pour déterminer l'évolution future des calculs). Autrement dit, les systèmes de transition permettent de rendre compte de l'évolution des programmes quand elle ne dépend que de l'état courant mais pas quand elle dépend de l'histoire du calcul pour arriver dans cet état. Par souci de généralité, nous sommes donc conduits à définir une sémantique (cf. 2.1) comme un ensemble d'états, un ensemble d'actions et un ensemble de traces quelconques.

Pour formaliser la notion de "pas d'exécution" d'un programme dont la sémantique est arbitraire nous définissons (cf. 2.3) la notion de système de transition engendré par une sémantique. Les transitions sont simplement celles qu'on peut observer le long d'une trace quelconque.

Pour faire des preuves de programmes il est souvent plus pratique de ne pas raisonner sur la sémantique du programme mais plutôt sur une sémantique qui lui est proche. De telles techniques sont souvent utilisées sans justification. Pour démontrer leur validité (aux chapitres 4 pour l'invariance et 5 pour la fatalité) il est nécessaire de les formaliser. Nous le faisons au moyen de relations entre sémantiques qui sont étudiées aux paragraphes 2.5 et 2.6.

Par exemple pour faire une preuve de correction partielle d'un programme parallèle, on peut ignorer l'hypothèse que son exécution est faiblement équitable (tout processus toujours activable est fatalement activé). Nous formaliserons la relation entre la sémantique non équitable et la sémantique équitable de ce programme au moyen d'une fermeture d'une sémantique par réduction aux traces équitables (cf. 2.6.4). Dans une preuve d'invariance, on peut ne pas tenir compte des branches du programme qui sont mortes. Pour le démontrer nous introduirons la notion de fermeture d'une sémantique par réduction aux états accessibles (cf. 2.6.3). Enfin, il est fréquent d'utiliser des variables auxiliaires pour démontrer la correction partielle de programmes parallèles. La relation entre la sémantique du programme et celle du programme transformé (contenant les variables auxiliaires) peut être définie par réduction des actions inobservables (pour éliminer les affectations aux variables auxiliaires) (cf. 2.5.4.2) et par concordance à une relation entre états près (pour éliminer les variables auxiliaires). Lorsque nous définissons une relation entre sémantiques, ceci induit une relation entre systèmes de transition (en considérant la relation entre les sémantiques qu'ils engendrent) dont nous étudions les propriétés.

Nous constatons au paragraphe 2.5 qu'en général, une sémantique et le système de transition qu'elle engendre ne donnent pas les mêmes informations. En effet, une sémantique est en général différente de sa rétraction par transitions c'est-à-dire de la sémantique engendrée par le système de transition qu'elle engendre. Nous dirons que la sémantique est close dans le cas contraire. Ceci nous amènera (cf. 2.5.5) à la caractérisation des sémantiques closes à l'aide de divers opérateurs sur les sémantiques définis au paragraphe 2.4). Pour résumer, très intuitivement, une sémantique

est close si elle est fermée par fusion (cf 2.6.5, le comportement futur de l'exécution dépend seulement de l'état courant qui a été atteint et non de la façon dont il a été atteint), réduite par élimination des traces préfixes strictes (cf. 2.6.6, l'arrêt de l'exécution en un état ne dépend que cet état) et fermée par limites (cf. 2.6.7, les limites des comportements finis sont des comportements infinis acceptables).

Nous posons au paragraphe 2.7 le problème de la spécification de la sémantique d'un programme. Quand la sémantique est close (cf. 2.7.1), on peut le faire aisément à l'aide d'un système de transition (qui peut être lui-même défini par induction sur la syntaxe du programme). Pour une sémantique non close, ce n'est pas possible directement. On peut toujours le faire indirectement en incluant un résumé de l'histoire des calculs dans les états et un contrôleur pour surveiller les transitions (cf. 2.7.2.2) ou bien spécifier la sémantique non close comme un sous-ensemble des préfixes des traces engendrées par un système de transition (cf. 2.7.2.1). Quand la sémantique est non close mais réduite par élimination des traces préfixes strictes, il suffit de considérer un sous-ensemble des traces engendrées par un système de transition (cf. 2.7.3).

Nous terminons ce chapitre par des exemples de définitions de la sémantique d'un langage de programmation (cf 2.8). Il s'agit d'illustrer ce qui vient d'être dit sur les méthodes de spécification de sémantiques de programmes mais surtout de disposer d'exemples qui nous servirons dans la suite pour illustrer la construction systématique de méthodes de preuves. Nous définissons la syntaxe et la sémantique opérationnelle des programmes séquentiels (cf 2.8.1, qui sont composés d'affectations (déterministes ou aléatoires), de conditionnelles

et d'itérations), de programmes parallèles asynchrones (cf. 2.8.2, qui sont composés de processus séquentiels partageant des données communes) auxquels nous ajoutons en 2.8.3 la possibilité de communiquer sur rendez-vous par envoi et réception de messages sur des canaux (avec possibilité de sélection entre plusieurs alternatives comme dans CSP ou ADA) et en 2.8.4 l'hypothèse d'exécution faiblement équitable. Finalement, nous ajoutons en 2.8.5 la possibilité de synchroniser les processus au moyen de sémaphores. Pour nous convaincre que la définition des sémaphores que nous avons utilisée est bien celle proposée à l'origine par Dijkstra, nous démontrons certaines propriétés des sémaphores qui ont été énoncés par Habermann et sont généralement tenues pour vraies sans justification. Enfin nous donnons une sémantique libérale des sémaphores qui peut être utilisée pour démontrer des propriétés d'invariance.

1.3.3 PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

Au chapitre 3 nous définissons et illustrons très brièvement les propriétés de programmes que nous considérons dans cette thèse à savoir l'invariance (cf. 3.2) et la fatalité (cf. 3.3).

La propriété d'invariance conditionnelle (cf. 3.2.1) est la propriété d'invariance la plus générale que nous considérons. Nous disons que ψ est invariante sous condition Φ pour une sémantique si pour toute trace de cette sémantique et tout état courant dans cette trace la relation w est avoir entre l'état initial et l'état

courant quand la relation ϕ a été vraie entre l'état initial et tous les états précédant l'état courant. Nous parlons d'invariance relationnelle (cf. 3.2.2) quand ϕ est toujours vraie et d'invariance assertionnelle (cf. 3.2.3) quand de plus ψ ne dépend pas de l'état initial.

Ces définitions recouvrent un grand nombre de propriétés classiques des programmes comme la correction partielle, l'absence d'erreurs à l'exécution, la non-terminaison, l'exclusion mutuelle, l'absence d'interblocages globaux permanents et des propriétés moins classiques comme les propriétés de précédence du genre l'état courant ne peut pas satisfaire ψ sans qu'un état précédent ait satisfait ϕ.

La propriété de fatalité sous invariance (cf. 3.3.1) est la propriété de fatalité la plus générale que nous considérons. Nous dirons que ψ est fatale sous invariance de ϕ pour une sémantique si pour toute trace de cette sémantique il existe un état (dit but) telle que la relation ψ soit vraie entre l'état initial et le but et la relation ϕ soit vraie entre l'état initial et tous les états qui précèdent le but. A nouveau, nous parlons de fatalité relationnelle quand ϕ est toujours vraie et de fatalité assertionnelle si de plus ψ ne dépend pas de l'état initial.

Ces définitions recouvrent également un grand nombre de propriétés classiques des programmes comme la terminaison, la correction totale, la garantie d'entrée en section critique ou de réponse à un signal, l'absence de famine d'un processus, etc.

D'autres propriétés des programmes peuvent également se ramener à l'invariance et à la fatalité par exemple en considérant les suffixes de la sémantique du programme ou d'autres relations entre sémantiques comme étudiées au chapitre 2.

1.3.4 PREUVES D'INVARIANCE

Le chapitre 4 est consacré aux fondements des méthodes de preuve de propriétés d'invariance de programmes séquentiels, non-déterministes ou parallèles.

Nous commençons par étudier au paragraphe 4.1 des relations entre sémantiques qui conservent l'invariance avec l'idée que pour démontrer une propriété d'invariance d'un programme relativement à une sémantique nous pouvons essayer de nous ramener à la preuve d'une propriété similaire relativement à une autre sémantique (généralement plus simple).

Par exemple, les propriétés d'invariance sont conservées pour des sémantiques concordantes à des relations ou fonctions entre états et/ou actions près (cf. 4.1.1). Elles sont également conservées après réduction des états inobservables (cf. 4.1.2). La propriété la plus importante est que pour faire une preuve d'invariance pour une sémantique, il est toujours correct de raisonner sur le système de transition qu'elle engendre. Autrement dit les propriétés d'invariance sont conservées par extraction de la sémantique par transitions (cf. 4.1.3). Cette méthode de preuve est correcte mais elle n'est en général pas sémantiquement complète c'est-à-dire qu'il se peut qu'une propriété

soit invariante pour une sémantique mais pas pour sa rétraction par transitions. Toutefois la méthode est sémantiquement complète quand la sémantique est fermée par fusion (cf. 4.1.3.v4) et donc en particulier pour les langages considérés au paragraphe 2.8.

Au paragraphe 4.2 nous étudions les principes d'induction qu'on peut utiliser pour démontrer les propriétés d'invariance des programmes. Un principe d'induction décrit l'essence d'une méthode de preuve de manière très concise et abstraite.

Comme nous avons vu dans le paragraphe 4.1, nous rencontrons très fréquemment (mais pas toujours) des sémantiques closes. Nous commençons donc par étudier ce cas particulier (cf. 4.2.1). La méthode la plus connue pour démontrer des propriétés d'invariance de programmes est la méthode de Floyd, Naur et Hoare. Partant d'un exemple, nous inférons le principe d'induction de base pour cette méthode (cf. 4.2.1.1). Essentiellement celui-ci exprime la propriété assez évidente suivante : pour démontrer que la fermeture transitive reflexive t^* d'une relation de transition $\exists a \in A. t_a$ entraine une relation invariante ψ ($\forall s,s' \in S.\ t^*(s,s') \Rightarrow \psi(s,s')$) il faut et il suffit qu'il existe un invariant I plus fort que ψ ($\forall s,s' \in S.\ I(s,s') \Rightarrow \psi(s,s')$) qui soit vrai quand l'état courant est un état initial ($\forall \underline{s} \in S.\ I(\underline{s},\underline{s})$) et reste vrai pour tous les descendants possibles des états initiaux ($\forall s,s',s'' \in S, a \in A.\ (I(s,s') \wedge t_a(s,s'')) \Rightarrow I(s,s'')$). Dans ce chapitre nous traitons de l'invariance relationnelle en remarquant que la généralisation à l'invariance conditionnelle est triviale (cf. 4.2.1.1-2).

Nous étudions ensuite les variantes possibles de ce principe d'induction de base de façon à en dériver toutes les méthodes existantes plus quelques autres. Puisque l'étude soit systématique nous considérons des transformations de principes d'induction (cf. 4.2.1.2). Pour une propriété de la forme ($[E(\underline{s}) \wedge S(\bar{s})] \Rightarrow \psi(\underline{s},\bar{s})$) relative à une relation entre états initiaux et finaux, il est possible de restreindre l'invariant aux états initiaux (cf. 4.2.1.2.1). Une autre transformation également triviale (cf. 4.2.1.2.2) consiste à remarquer que la condition de vérification ($\forall s,s' \in S, a \in A.\ [\exists s \in S.\ I(s,s') \wedge t_a(s,s')] \Rightarrow I(\underline{s},s')$) est équivalente à ($\forall \underline{s},s \in S.\ I(\underline{s},s) \Rightarrow \neg[\exists s' \in S, a \in A.\ t_a(s,s') \wedge \neg I(\underline{s},s')]$). Autrement dit, nous pouvons utiliser une plus forte post-condition (comme Floyd pour l'affectation) ou bien une plus faible pré-condition (comme Hoare pour l'affectation). Une autre transformation (cf. 4.2.1.2.3) consiste à remarquer que nous pouvons raisonner sur les relations inverses ($t^* \Rightarrow \psi$ si et seulement si $(t^{-1})^* \Rightarrow \psi^{-1}$). Ceci nous permet de formaliser la méthode de Morris-Wegbreit dite "subgoal induction" qui consiste donc à appliquer la méthode de Floyd sur l'inverse du programme. Il est également possible (cf. 4.2.1.2.4) de remplacer l'invariant I par sa négation ce qui conduit à des preuves contrapositives par l'absurde (qui sont ignorées dans la littérature). Enfin (cf. 4.2.1.2.5), quand la propriété à démontrer est une assertion (au lieu d'une relation), nous pouvons utiliser une assertion (comme dans la méthode de Floyd-Naur) au lieu d'une relation invariante (comme dans la méthode de Manna).

Dans le paragraphe 4.2.1.3 nous déterminons tous les principes d'induction que nous pouvons dériver par les transformations ci-dessus, ce qui permet de retrouver toutes les méthodes classiques et de découvrir quelques autres.

Nous montrons ensuite que tous ces principes d'induction sont fortement équivalents (cf. 4.2.1.4) en ce sens que si nous avons

découvert l'invariant I qui convient pour un principe d'induction nous pouvons déterminer l'invariant I' qui convient pour faire la preuve avec tout autre principe d'induction. (Ceci m'empêche d'ailleurs pas que la preuve avec un principe d'induction soit plus facile qu'avec un autre). Ils sont également corrects (si l'invariant I satisfait les conditions de vérification alors ψ est invariante) et sémantiquement complets (si ψ est invariante, nous pouvons toujours trouver un invariant I satisfaisant les conditions de vérification (mais nous n'avons pas forcément de complétude syntaxique en ce sens que si le langage d'assertions est mal choisi, nous ne pouvons peut-être pas formuler I dans ce langage)).

Les résultats précédents se généralisent aux sémantiques non closes fermées par fusions (cf. 4.2.2) puisque dans ce cas une propriété est invariante pour cette sémantique si et seulement si elle l'est pour la réduction de cette sémantique par transitions.

Ceci n'est évidemment pas vrai pour une sémantique non fermée par fusions (cf. 4.2.3).

Si cette sémantique a été définie comme un sous-ensemble des préfixes des traces engendrés par un système de transition, il est correct mais pas sémantiquement complet d'utiliser les principes d'induction précédents pour ce système de transition. Pour être complets, nous proposons un principe d'induction utilisant des variables auxiliaires (dans la preuve mais pas dans le programme) permettant de cumuler des histoires. Ceci permet de tenir compte dans le principe d'induction proposé du fait que les successeurs possibles d'un état ne dépendent pas uniquement de cet état mais également de la façon dont il a été atteint (ce que nous savons quand nous cumulons l'histoire).

Si cette sémantique non fermée par fusions a été définie par concordance avec une sémantique close (cf. 4.2.3.2), nous pouvons aisément nous ramener aux principes d'induction qui nous étudiés pour les sémantiques closes.

De plus nous montrons que ces nouveaux principes d'induction se ramènent aux précédents quand la sémantique est fermée par fusions.

Enfin nous montrons (cf. 4.2.3.3) que les deux approches (cumul de l'histoire dans des variables auxiliaires ou utilisation d'une sémantique close concordante) sont fortement équivalentes.

Les principes d'induction tels que nous les avons proposés ne sont pas très pratiques à mettre en œuvre directement dans une preuve. Par exemple, dans la condition de vérification $(I(s,s') \wedge t_a(s,s')) \Rightarrow I(s,s')$, t_a serait une énorme formule définissant l'exécution d'un pas quelconque du programme. Il est donc préférable de décomposer cette condition de vérification complexe en une conjonction de conditions de vérification plus simples correspondant par exemple chacune à un pas élémentaire du programme. C'est l'objet du paragraphe 4.3 qui porte sur la construction systématique d'une méthode de preuve d'invariance à partir d'une sémantique opérationnelle et d'un principe d'induction par décomposition de l'invariant global en invariants locaux.

Plutôt que d'imaginer empiriquement une méthode de preuve pour un langage de programmation puis de démontrer sa correction et sa complétude sémantique à posteriori, nous proposons de construire la méthode de preuve de manière systématique. La démarche (cf. 4.3.1) consiste tout d'abord à définir la sémantique opérationnelle

à l'aide d'un système de transition (cf. 4.3.1.1), puis à définir la propriété invariante à démontrer (cf. 4.3.1.2) ce qui permet de choisir le principe d'induction adéquat (cf. 4.3.1.3) parmi ceux précédemment proposés. Ce principe d'induction fait intervenir un invariant global (portant sur les états du programme) alors qu'on préfère généralement des invariants locaux (qui portent par exemple sur les états du programmes correspondant à chaque point du programme, ce qui permet par exemple dans la méthode de Floyd de les associer en commentaire au point du programme auquel ils correspondent). Les invariants locaux étant choisis (cf. 4.3.1.4), il faut définir leur sémantique (cf 4.3.1.5) c'est-à-dire définir l'invariant global qui correspond à des invariants locaux et inversement. La détermination de la méthode de preuve consiste alors à dériver les conditions de vérification correspondantes. Ceci (cf. 4.3.1.7) en remplaçant le système de transition par sa définition et l'invariant global par les invariants locaux dans le principe d'induction qui a été choisi. Il ne reste ensuite qu'à simplifier pour obtenir les conditions de vérification élémentaires. La méthode obtenue est correcte par construction. Il faut ensuite vérifier qu'elle est sémantiquement complète (cf. 4.3.1.8). Cette vérification peut être inutile ou simplifiée selon la nature de la relation entre invariant global et invariants locaux. C'est pourquoi nous remarquons au paragraphe 4.3.1.6.1 qu'en général, l'ensemble des invariants locaux forme un treillis qui correspond au treillis des invariants globaux (qui sont des sous-ensembles de l'ensemble des paires d'états). Nous étudions ensuite (cf. 4.3.1.6.2) diverses propriétés possibles des correspondances entre invariants locaux et globaux (correspondances monotones, (demi- ou quasi-) correspondances de Galois (injectives, surjectives), isomorphismes complets).

Dans le paragraphe suivant 4.3.2, nous donnons des exemples

construction d'une méthode de preuve de non-terminaison, d'absence d'erreurs à l'exécution et d'invariance globale, par l'absurde pour les programmes séquentiels (cf. 4.3.2.1). Comme ces méthodes ne sont pas classiques, nous les illustrons par des exemples simples.

Ensuite (cf. 4.3.2.2), nous étendons la méthode de Morris-Wegbreit (dite "subgoal induction" pour démontrer la correction partielle de programmes séquentiels) aux programmes parallèles (comme Owicki-Gries ont étendu la méthode de Floyd-Naur-Hoare aux programmes parallèles asynchrones) et nous la généralisons à d'autres propriétés d'invariance. Nous commençons par considérer la correction partielle des programmes séquentiels (cf. 4.3.2.2.1-4) puis l'invariance globale (alors que Morris-Wegbreit croyaient que ce n'était pas possible). Nous abordons ensuite la correction partielle de programmes parallèles asynchrones (cf. 4.3.2.2.5) où nous retrouvons la décomposition de la preuve en une preuve séquentielle par processus et une preuve d'absence d'interférences. Cette méthode étant nouvelle nous des exemples d'application (comme le calcul parallèle asynchrone de n!).

Ayant remarqué que dans les méthodes "en avant" (à la Floyd) l'invariant décrit ce qui a été fait (relation entre l'état initial et l'état courant) alors que pour les méthodes "en arrière" (à la Morris-Wegbreit) l'invariant décrit ce qui reste à faire (relation entre l'état courant et l'état final) et que ces deux types d'informations peuvent être très utiles pour aider à la compréhension du programme, nous proposons un principe d'induction (cf. 4.3.2.2.5-3) combinant les avantages de ces deux méthodes (mais où la preuve présente évidemment des redondances).

Nous généralisons ensuite cette méthode aux programmes parallèles synchrones pour la preuve d'absence d'interblocages

globaux permanents (cf 4.3.2.2.3), la preuve d'exclusion mutuelle (cf. 4.3.2.2.4) et la preuve de non-terminaison (cf. 4.3.2.2.5).

Nous tentons ensuite (cf. 4.3.2.2.6) d'expliquer pourquoi la méthode de Morris-Wegbreit n'a pas eu le même succès que la méthode de Floyd. La raison principale nous semble bien mise en évidence dans le cas des programmes parallèles où les mêmes invariants peuvent être utilisés, pour les méthodes "à la Floyd", pour démontrer la correction partielle, l'absence d'erreurs à l'exécution, l'absence d'interblocages, l'exclusion mutuelle etc., alors que ce n'est pas le cas pour les méthodes "à la Morris-Wegbreit". Pour remédier à cet inconvénient, nous proposons d'utiliser un principe d'induction combinant l'induction en avant et en arrière.

Nous terminons cette série d'exemples en construisant une méthode de preuve de propriétés d'invariance pour les programmes parallèles communicants (cf. 4.3.2.3). Pour toutes ces méthodes que nous avons construites, nous avons donné une preuve de correction et de complétude sémantique.

Les méthodes de preuve de propriétés d'invariance pour les programmes parallèles connues dans la littérature (Aschroft, Hoare, Howard, Keller, Lamport, Mazurkiewitz, Newton, Owicki-Gries, ...) sont souvent difficiles à comparer à cause des formalismes souvent très différents qui sont utilisés pour les présenter. L'objectif du paragraphe 4.3.2.4 est de montrer qu'elles dérivent toutes du même principe d'induction (qui est à la base de la méthode de Floyd) et ne diffèrent que par la façon de décomposer l'invariant global utilisé dans ce principe d'induction en invariants locaux associés à des points du programme.

Par exemple, la méthode de Ashcroft et de Keller consiste à utiliser un seul invariant global (cf. 4.3.2.4.1). Ce fut la première généralisation de la méthode de Floyd aux programmes parallèles mais elle a l'inconvénient qu'il y a une seule condition de vérification qui n'est pas décomposée en conditions élémentaires.

A l'inverse, la méthode de Aschroft-Manna consiste à utiliser un invariant local associé à chaque état de contrôle (cf. 4.3.2.4.3). Dans ce cas, la décomposition est par contre trop fine, et par conséquent le nombre de conditions de vérification est trop grand.

Un premier compromis dans la méthode de Owicki-Gries consiste à utiliser un invariant local sur les variables, associé à chaque point (et non plus à chaque état) de contrôle du programme (cf. 4.3.2.4.3). Mais cette méthode est sémantiquement incomplète.

Pour y remedier, nous pouvons suivre Newton et Lamport et utiliser des invariants locaux portant sur l'état de contrôle et les variables, associés à chaque point de contrôle du programme (cf. 4.3.2.4.4).

Nous pouvons également comme l'ont proposé Owicki-Gries utiliser des invariants locaux portant sur les variables du programme et sur des variables auxiliaires, associés à chaque point de contrôle du programme (cf. 4.3.2.4.5). Nous montrons que la méthode est correcte et sémantiquement complète (en définissant la sémantique auxiliaire, qui peut toujours être utilisée pour faire la preuve, à une réduction des états inobservables et à une fonction des états près. Dans cette sémantique auxiliaire, les états de contrôle sont simplement simulés par des variables, ce qui montre que les variables auxiliaires ne servent dans la méthode de Owicki-Gries qu'à simuler l'état de contrôle dont les invariants locaux sont indépendants).

Avec ces décompositions, le nombre de conditions de vérification est proportionnel au produit des longueurs des processus du programme parallèle alors qu'en pratique on souhaite que le nombre de conditions de vérification croisse linéairement avec la taille du programme. C'est le cas avec une méthode proposée par Lamport qui consiste à utiliser des invariants locaux portant sur l'état de contrôle et les variables, associés à chaque processus du programme parallèle

Il est possible enfin, d'utiliser une information redondante (comme dans des méthodes proposées par Hoare, Howard, ...) sous la forme d'un invariant global et d'invariants locaux associés à divers points du programme (cf. 4.3.2.4.7).

Cette profusion de méthodes nous amène à les classer selon le principe d'induction sous-jacent et selon la finesse de la décomposition de l'invariant global en invariants locaux (cf. 4.3.2.4.8).

Pour conclure ce paragraphe, il nous semble que les exemples que nous avons donnés montrent que le choix de la finesse de la décomposition de l'invariant global en invariants locaux ne devrait pas être fixée une fois pour toutes dans une méthode de preuve. Il est bien préférable de choisir cette décomposition en fonction du problème à traiter. La formalisation des méthodes de preuve d'invariance que nous avons proposée permet de le faire sans difficultés.

Nous terminons ce chapitre sur les preuves d'invariance par le paragraphe 4.3.5 consacré à l'analyse sémantique de programmes. Nous le faisons parce que ce problème (qui rappelons le, consiste

à déterminer statiquement et automatiquement des invariants pour un programme) a motivé le travail que nous présentons ici. Nous le faisons surtout pour montrer que les résultats obtenus se généralisent sans peine aux programmes parallèles. La présentation est très brève. Nous rappelons simplement comment faire une analyse sémantique "en avant" (cf. 4.3.2.5.1, déterminer un sur-ensemble des descendants des états initiaux), comment faire une analyse sémantique "en arrière" (cf. 4.3.2.5.2, déterminer un sur-ensemble des ascendants des états finaux) et comment faire une analyse combinée "avant-arrière" (cf. 4.3.2.5.3, déterminer un sur-ensemble des états qui sont à la fois descendants des états initiaux et ascendants des états finaux). Comme le formalisme utilisé est très général et qu'il englobe les programmes parallèles nous nous contentons de donner quelques exemples pour montrer l'application des méthodes d'analyse sémantique à ce type de programmes.

1.3.5 PREUVES DE FATALITE

Le chapitre 5 est consacré à l'étude des méthodes de preuve de propriétés de fatalité des programmes séquentiels et parallèles.

Un certain nombre de résultats obtenus au chapitre précédent (comme les transformations de principes d'induction, la décomposition de l'invariant global en invariant locaux, ...) s'appliquent également ici (avec les légères adaptations qui pourraient être nécessaires). Pour éviter les répétitions, nous ne reprendrons pas ces mêmes idées dans ce chapitre même si elles s'appliquent

Ce chapitre comprend deux paragraphes importants, le paragraphe 5.2 consacré aux principes d'induction "à la Floyd" et le paragraphe 5.3 consacré aux principes d'induction "à la Burstall". En fait nous aurions pu rédiger différemment en présentant 5.2 en quelques phrases comme un cas particulier de 5.3. Nous avons choisi d'aller du particulier (5.2) au général (5.3) de façon à refléter l'évolution historique mais surtout pour graduer les difficultés. Pour éviter les redites nous présentons en 5.2 un certain nombre de résultats (comme les principes d'induction pour les sémantiques non closes, ...) qui ne seront pas repris en 5.3 car leur généralisation ne nous a pas semblé présenter des difficultés une fois que l'idée a été donnée en 5.2.

Nous commençons l'étude des méthodes de preuve de fatalité par celle des relations entre sémantiques qui conservent la fatalité (cf. 5.1). Malheureusement, les résultats positifs sont beaucoup moins nombreux que pour l'invariance. Sans chercher l'exhaustivité, nous montrons (cf. 5.1.1) que les propriétés de fatalité sont conservées par inclusion de sémantiques (mais dans un sens seulement car par exemple il n'est pas toujours possible de démontrer une propriété de fatalité d'un programme parallèle synchrone en raisonnant sur la sémantique libérale des sémaphores) et que (cf. 5.1.2) les propriétés de fatalité sont conservées pour des sémantiques concordantes à des relations entre états et actions près (dans les deux sens, sous certaines conditions).

Dans le paragraphe 5.2, nous étudions les preuves de fatalité par des principes d'induction généralisant la méthode de Floyd.

Nous commençons par rappeler en 5.2.1 la méthode de Floyd (dite des assertions invariantes et de l'ordre bien fondé) pour démontrer la correction totale des programmes séquentiels.

Ceci nous permet d'en extraire le principe d'induction de base pour démontrer les propriétés de fatalité des sémantiques closes (cf. 5.2.2).

Nous étudions ensuite une série de principes d'induction équivalents au principe d'induction de base (cf. 5.2.3) qui reflètent quelques unes des variantes possibles de la méthode de Floyd. Par exemple, il n'est pas nécessaire d'associer une fonction de terminaison à tous les points de contrôle du programme mais seulement aux points de coupure des boucles. L'utilisation de bons-ordres n'est pas obligatoire puisque les relations bien-fondées suffisent et sont quelquefois plus commodes. La fonction de terminaison peut être remplacée par une variable auxiliaire (dans la preuve mais qui n'apparaît pas nécessairement dans le programme) qui décroît strictement à chaque pas. Cette variable auxiliaire peut toujours être choisie comme un ordinal, etc.

Nous abordons ensuite le problème de la correction et de la complétude sémantique des principes d'induction à la Floyd qui précèdent (cf. 5.2.4).

Nous remarquons en 5.2.5 que si la propriété fatale est une relation entre des états initiaux et finaux il faut en général, que la fonction de terminaison porte sur l'état courant mais également

sur l'état initial (ce que beaucoup d'ouvrages introductifs ignorent. Ils présentent donc une méthode sémantiquement incomplète).

Il est également intéressant de caractériser les relations bien-fondées (ou de manière équivalente les ordinaux) qui sont nécessaires pour faire des preuves de fatalité basées sur les principes d'induction généralisant la méthode de Floyd (cf. 5.2.6). Pour ce faire, nous dirons que le non-déterminisme d'une sémantique est m-borné si le cardinal de l'ensemble des successeurs d'un état quelconque pour la relation de transition qu'elle engendre est strictement inférieur à m. En particulier le non-déterminisme est fini (Dijkstra dit "borné") si tout état a un nombre fini de successeurs possible. Nous montrons que pour une sémantique close dont le non-déterminisme est m-borné, il est toujours possible de faire des preuves de fatalité avec des relations bien-fondées dont l'ordre est inférieur à m^+ (où $m^+ = \omega$ si $m < \omega$, $m^+ = m$ quand m est un cardinal régulier suivi m^+ qui est le plus petit cardinal strictement supérieur à m). Cette limite est stricte quand m est régulier. Comme cas particulier, nous obtenons que la méthode de Knuth-Luckham-Suzuki (qui consiste à utiliser un compteur strictement incrémenté à chaque tour de boucle et dont la valeur est bornée) n'est pas sémantiquement complète quand le non-déterminisme n'est pas fini (et ne peut donc pas être généralisée au cas des programmes parallèles équitables).

Le paragraphe 5.2.7 est consacré à la décomposition des conditions de vérification. Le cas général ayant été étudié en 4.3, nous nous contentons d'illustrer quelques décompositions (de façon à montrer comment les méthodes de Lamport et Owicki-Gries initialement conçues pour les preuves de correction partielle peuvent être étendues aux preuves de correction totale).

N'ayant abordé jusqu'ici que le cas des sémantiques closes, le cas des preuves de fatalité pour les programmes parallèles équitables étaient exclues. C'est pourquoi nous étudions au paragraphe 5.2.8 les principes d'induction "à la Floyd" pour démontrer les propriétés de fatalité de sémantiques non closes.

Comme pour l'invariance, nous pouvons définir la sémantique non close par concordance avec une sémantique close à une fonction de état près (cf. 5.2.8.1). Dans ces conditions, n'importe lequel des principes d'induction introduits pour les sémantiques closes est utilisable. Ceci revient, par exemple pour un programme parallèle équitable, à raisonner sur un programme transformé qui incorpore un contrôleur d'exécution assurant l'équité.

Une autre approche (cf. 5.2.8.2) peut être utilisée quand nous spécifions la sémantique non close par un sous-ensemble des préfixes des traces engendrées par un système de transition. Elle consiste à cumuler l'histoire des calculs dans une variable auxiliaire. Ceci permet, quand la sémantique n'est pas fermée par limites de ne pas imposer que la fonction de terminaison décroisse à chaque pas mais seulement aux points de coupure qui ne sont pas déterminés statiquement comme dans la méthode de Floyd mais dynamiquement c'est-à-dire en fonction de l'histoire des calculs. Comme cas particulier nous retrouverons la méthode de Pnueli-Lehmann-Stavi pour démontrer la correction totale de programmes parallèles faiblement équitables et qui consiste essentiellement à appliquer la méthode de Floyd mais avec la possibilité que la fonction de terminaison ne décroisse pas tant qu'un processus reste activable sans être activé. De plus, quand la sémantique n'est pas réduite par élimination des traces préfixes strictes, l'histoire est utilisée pour s'assurer que la

but est atteint pour les traces finies avant la fin de la trace (qui peut ne pas être un état sans successeur). Enfin, quand la sémantique n'est pas fermée par fusion, l'invariant doit être vrai pour tous les états qui peuvent être atteints en suivant le préfixe d'une trace où le but n'est jamais atteint mais pas forcément pour tous les préfixes obtenus par transitions successives. Là encore, le cumul de l'histoire dans une variable auxiliaire est utile.

Enfin, nous montrons au paragraphe 5.2.8.3 que les deux approches (utilisation d'une sémantique auxiliaire incluant un contrôleur d'exécution ou bien utilisation de variables auxiliaires pour cumuler l'histoire) sont équivalentes.

Au paragraphe 5.3.1, nous présentons la méthode des assertions intermittentes de Burstall à l'aide d'exemples puis nous en déduisons le principe d'induction de base formalisant de manière très concise cette méthode. Nous démontrons que ce principe de preuve est correct. La question de la complétude sémantique est plus complexe. En utilisant l'induction transfinie (plutôt que finie puisque le principe d'induction généralise la méthode de Burstall au non-déterminisme infini) nous montrons que ce principe d'induction est sémantiquement complet sous une condition suffisante (mais pas nécessaire) sur la sémantique et la propriété de fatalité. Cette condition exprime qu'un état ne peut pas être un but sur une trace et appartenir à un préfixe d'une autre trace le long duquel le but n'a pas été atteint. Cette condition est évidemment vérifiée dans le cas de Burstall où l'on traite des programmes déterministes mais également pour des généralisations aux programmes non-déterministes (Francez, Apt-Delporte, ...) où sont considérées des

Lorsqu'on considère des propriétés de fatalité unaires, les relations entre les valeurs initiales et finales des variables des programmes ne peuvent être exprimées qu'en considérant un programme transformé dans lequel les valeurs initiales sont affectées à des variables auxiliaires. Outre la transformation du programme sans raison fondamentale, l'utilisation de variables auxiliaires est en un sens trop souple parce que nous pouvons relier des états intermédiaires quelconques lors d'un calcul et même mémoriser toute l'histoire du calcul. Une telle liberté d'utilisation de variables auxiliaires n'est pas dans l'esprit de la méthode proposée par Burstall et des exemples donnés par Manna-Waldinger où les lemmes démontrés par induction sur les données sont toujours de la forme :

"if sometime $\phi(x_1,\dots,x_m) \wedge X_1=x_1 \wedge \dots \wedge X_m=x_m$ at l then
sometime $\psi(x_1,\dots,x_m,X_1,\dots,X_m)$ at l'"

(où $X_1,\dots,X_m$ sont les variables du programme et $x_1,\dots,x_m$ leurs valeurs symboliques respectives au point l du programme). Ceci s'exprime dans notre principe d'induction de base par l'utilisation de propriétés de fatalité binaires (mieux qu'en imposant des restrictions adéquates sur l'utilisation des variables auxiliaires qui dépendraient de la syntaxe des programmes). Cependant, nous faisons la conjecture que même pour les programmes déterministes, il existe des propriétés de fatalité pour lesquelles l'utilisation d'assertions binaires n'est pas sémantiquement complète.

Cette conjecture nous conduit en 5.3.2 à généraliser la méthode des assertions intermittentes de Burstall d'une part en utilisant l'induction transfinie (pour traiter le non-déterminisme non borné) et des assertions intermittentes ternaires (permettant d'exprimer des lemmes d'une forme plus générale "if sometime $\phi(z_1,\dots,z_m,X_1,\dots,X_m) \wedge X_1=z_1 \wedge \dots \wedge X_m=z_m$ at l then sometime $\psi(z_1,\dots,z_m,x_1,\dots,x_m,X_1,\dots,X_m)$ at l'" où $z_1,\dots,z_m$ (respectivement $x_1,\dots,x_m$) désignent les valeurs des variables au point d'entrée (respectivement au point l)

du programme). Nous démontrons que ce principe d'induction généralisé est correct et sémantiquement complet.

De ce principe, nous dérivons au paragraphe 5.3.3 toute une série de principes d'induction de plus en plus abstraits et concis. Par exemple, il est intéressant de considérer un nombre transfini et non plus fini d'assertions intermittentes (que nous pouvons représenter de manière finie au moyen de variables auxiliaires). Ceci étend la méthode de Burstall de façon à incorporer la méthode de Floyd et permet d'utiliser des assertions intermittentes binaires et non plus tertiaires (en prenant formellement un lemme différent pour chaque état initial). Parmi ces principes d'induction il en est un qui formalise l'idée de Schwarz que la méthode de Burstall consiste à démontrer des théorèmes par induction mathématique à partir d'axiomes spécifiant l'effet des commandes élémentaires du programme. Les principes d'induction les plus abstraits permettent une meilleure compréhension de la méthode de Burstall (par exemple nous montrons que "l'évaluation symbolique" et l'"induction sur les données" peuvent être comprises de manière unifiée et réduite à une induction sur les calculs). Ces généralisations successives introduisent plus de souplesse dans l'écriture des preuves mais pas de puissance supplémentaire puisque nous démontrons que tous les principes de preuve considérés sont corrects et sémantiquement complets donc équivalents. Comme les principes d'induction les plus abstraits peuvent paraître très éloignés de la méthode de Burstall, nous donnons quelques exemples pour montrer le contraire.

Le principe d'induction "à la Floyd" comporte une induction le long des traces d'exécution tandis que le principe d'induction "à la Burstall" comporte la combinaison d'une induction le long de parties de traces d'exécution (en relation avec "l'évaluation symbolique" de Burstall) et d'une récursivité (en relation avec "l'induction sur les données" de Burstall). Le principe d'induction "à la Floyd" correspond au cas particulier du principe d'induction "à la Burstall" où la récursivité n'est pas utilisée. Comme conséquence immédiate, le principe d'induction "à la Burstall" est sémantiquement complet puisque nous avons démontré précédemment que celui "à la Floyd" l'est. Cette remarque explique également pourquoi la méthode de Burstall est mieux adaptée que la méthode de Floyd pour démontrer la correction totale de programmes itératifs obtenus par élimination de la récursivité (la raison étant qu'il est possible de conserver la récursivité dans la preuve). Beaucoup plus important est le fait que le principe d'induction "à la Burstall" offre des possibilités de décomposer une preuve d'un théorème de fatalité en preuves indépendantes de lemmes plus simples, qui n'existent pas avec le principe d'induction "à la Floyd" qui nécessite une preuve globale (cf. 5.3.5).

L'argument de complétude sémantique pour le principe d'induction "à la Burstall" n'est pas pleinement satisfaisant parce que le style des preuves permises est fixe. Les utilisateurs de la méthode de Burstall ont besoin d'un résultat de complétude plus fort puisqu'ils aimeraient savoir si les lemmes qu'ils ont l'intention d'utiliser dans leurs preuves peuvent toujours être choisis librement. Une réponse affirmative est donnée au paragraphe 5.3.4 (avec la condition nécessaire et suffisante que chaque lemme doit concerner une propriété qui est fatale pour le programme mais aussi relativement aux autres lemmes qui sont utilisés dans sa preuve).

Une certaine polémique a entouré la comparaison des méthodes de Floyd et Burstall (Manna-Waldinger, Gries, ...), les uns affirmant qu'il y a des programmes qui ont une preuve "naturelle" par la méthode

de Burstall et pas avec la méthode de Floyd, les autres travaillant suffisamment la preuve avec la méthode de Floyd jusqu'à donner une impression de simplicité. (La plupart des exemples fournis (comme la version itérative de la fonction d'Ackermann) étaient obtenus par élimination de la récursivité et nous en donnons un (cf. 5.4.1) qui est simple, semble convaincant et n'est pas de cette nature). Comme contribution à ce débat nous démontrons au paragraphe 5.4 que toute preuve obtenue par une méthode peut se récrire systématiquement en une preuve par l'autre méthode. La preuve est assez technique et longue mais intuitivement la transformation entre les deux preuves est très similaire dans un sens à l'élimination de la récursivité dans les programmes et dans l'autre sens à la présentation récursive de programmes itératifs, (nous ne prétendons donc pas que ces transformations préservent le "naturel" des preuves).

Ayant montré que la méthode de Floyd est un cas particulier de la méthode de Burstall (après les généralisations adéquates que nous avons faites), il reste néanmoins que les présentations classiques des preuves par ces deux méthodes à l'aide d'assertions invariantes d'une part et d'assertions intermittentes d'autre part sont suffisamment dissemblables pour qu'il soit difficile d'uniformiser ces deux méthodes. C'est pourquoi nous introduisons, au paragraphe 5.5, la notion de charte de preuve.

L'idée de présenter graphiquement les preuves de programmes par des diagrammes acycliques fut introduite par Lamport et développée ultérieurement par Owicki-Lamport et Manna-Pnueli. Cependant ces méthodes n'étaient pas sémantiquement complètes à cause d'un certain

nombre de restrictions, principalement l'impossibilité de faire des inductions infinies. Notre formalisation est plus générale du fait qu'elle consiste à introduire des chartes de preuve qui sont bien structurées, peuvent éventuellement présenter des cycles et peuvent être utilisés récursivement pour les preuves par induction sur les données. Après avoir défini la notion de charte de preuve (cf. 5.5.1) nous démontrons la correction et la complétude sémantique de la méthode en montrant qu'elle correspond à l'utilisation d'un principe d'induction "à la Burstall" (cf. 5.5.2). Nous donnons ensuite quelques exemples de présentation de preuves par chartes (cf. 5.5.3), pour montrer que les preuves "à la Floyd" peuvent se présenter très naturellement par des chartes et également pour montrer que les chartes de preuve sont très utiles pour démontrer des propriétés de fatalité de programmes parallèles asynchrones. Enfin, les idées développées au paragraphe 5.2 concernant les preuves de fatalité pour les sémantiques non closes, s'appliquent directement. Nous le montrons simplement par des exemples en étendant les preuves de fatalité par chartes de preuves au cas des programmes parallèles faiblement équitables puis à celui des programmes parallèles synchrones.

1.3.6 CONCLUSION ET REFERENCES

Le chapitre 6 est une brève conclusion. Les références sont données à la fin de chaque chapitre.

2. SEMANTIQUE OPERATIONNELLE

2. SEMANTIQUE OPERATIONNELLE

2.1 DEFINITION DE LA SEMANTIQUE PAR UN ENSEMBLE DE TRACES COMPLETES

2.1.1 TRACES

2.1.2 SEMANTIQUE

2.2 DEFINITION DES SYSTEMES DE TRANSITION

2.3 SYSTEME DE TRANSITION ENGENDRE PAR UNE SEMANTIQUE

2.4 SEMANTIQUE ENGENDREE PAR UN SYSTEME DE TRANSITION

2.5 RELATIONS ENTRE SEMANTIQUES ET ENTRE SYSTEMES DE TRANSITION

2.5.1 INCLUSION DE SEMANTIQUES ET DE SYSTEMES DE TRANSITION

2.5.2 EQUIVALENCE DE SYSTEMES DE TRANSITION

2.5.3 CONCORDANCE ENTRE SEMANTIQUES ET ENTRE SYSTEMES DE TRANSITION A DES RELATIONS ENTRE ETATS ET/OU ACTIONS PRES

2.5.3.1 Concordance à une fonction des états près

2. SEMANTIQUE OPERATIONNELLE

Une preuve de correction d'un programme consiste à démontrer une relation entre une sémantique et une spécification de ce programme. Pour faire l'étude des méthodes de preuve de programmes, il est donc nécessaire de disposer d'une méthode de définition de la sémantique des programmes. Dans ce chapitre nous expliquerons la méthode que nous avons choisie pour définir la sémantique des programmes.

Pour définir la sémantique d'un langage de programmation il faut simplement définir la sémantique de tous les programmes de ce langage. Pour ce faire on procède généralement par induction sur la syntaxe abstraite des programmes.

Pour définir la sémantique d'un programme, il faut définir un modèle de l'ensemble des exécutions possibles de ce programme sur ordinateur. Il s'agit d'un modèle parcequ'une simplification de la réalité est nécessaire pour éviter d'avoir à tenir compte d'une multitude de détails, en général liés à une implémentation (nombre et capacité des mémoires, conventions de codage des informations, temps de calcul des opérations, ...). Il se pose alors le problème de savoir à quel niveau d'abstraction doit être définie la sémantique. Par exemple de manière classique la sémantique dénotationnelle (Milne-Strachey [76]) a pour but d'associer à un texte de programme syntaxiquement correct une fonction mathématique qui, à une donnée, associe le résultat du programme pour cette donnée. Ce type de définition rend difficilement compte de certaines notions liées à l'exécution (par exemple que deux processus d'un programme peuvent être en exclusion mutuelle en cours d'exécution).

Plutôt que de retenir les états d'entrée et de sortie, nous considérons un modèle opérationnel dont le niveau d'abstraction est celui qui permet d'observer la suite d'états intermédiaires par lesquels passe l'ordinateur pendant l'exécution d'un programme (ce qui permet par exemple d'étudier l'ensemble des valeurs prises par une variable au cours du calcul, de s'assurer qu'un processus d'un programme parallèle ne meurt pas de faim ou d'étudier des propriétés temps-réel en incluant une mesure de distance entre deux états successifs).

Un premier modèle de la sémantique opérationnelle des programmes consiste à utiliser les systèmes de transition (Keller [79]). Un système de transition définit l'ensemble des états du programme et une relation de transition entre un état et ses successeurs possibles. Les traces (ou exécutions) du programme sont alors les séquences d'états qui commencent par un état initial et telles que deux états successifs sont liés par la relation de transition. C'est le modèle défini dans Cousot-P [79, 81] que nous avons utilisé dans nos premiers travaux. Cette définition de la sémantique des programmes est bien adaptée à l'étude des méthodes de preuve car la relation de transition formalise bien la notion de pas du programme et les preuves procèdent généralement par induction sur le nombre de pas d'exécution du programme. Malheureusement cette approche ne permet pas de définir de relation entre plus de deux états successifs du programme. Ceci est gênant par exemple pour définir la sémantique de programmes parallèles équitables dont l'exécution est contrôlée par un agent extérieur dont l'état ne fait pas partie de celui du programme.

Un modèle plus général est celui des traces d'exécution (Pratt [79] Lamport [80]). Avec ce modèle la sémantique d'un programme est définie comme un ensemble de traces, chaque

trace étant la séquence des états intermédiaires observés au cours d'un calcul terminé ou infini. Contrairement à ce qui se fait généralement dans la littérature (cf. par exemple Pratt [79]), nous ne considérons pas de traces incomplètes qui correspondraient à un calcul en cours. Ceci nous permet de n'avoir à faire aucune hypothèse sur l'ensemble des traces que nous considérons (alors que Pratt [79] doit supposer que tout préfixe d'une trace en cours est une trace en cours, etc.). De plus, il n'y a aucune perte de généralité en omettant les traces inachevées. Ce modèle est plus général que les systèmes de transition car il permet de définir des relations entre un nombre quelconque d'états successifs du programme. Cependant il est mal adapté pour rendre compte des méthodes de preuve de programmes qui reprennent toutes l'idée due à Floyd [67] - Naur [66] qu'il est plus simple de raisonner sur des ensembles d'états par induction sur le nombre de pas du programme que sur l'ensemble des traces d'exécution.

Dans la suite nous associerons donc à un programme une sémantique (définie comme un ensemble de traces, chaque trace étant une suite finie ou infinie d'états) et un système de transition (défini comme une relation de transition sur un ensemble d'états). Dans ce chapitre nous nous poserons la question suivante : étant donnée une propriété P relative à une sémantique S caractérisée les propriétés P' et les sémantiques S' en fonction de P et S telles que P est vrai de S si, seulement si ou si et seulement si P' est vrai de S'. Ceci nous amènera en particulier à la question suivante : étant donnée une sémantique est-elle (ou jusqu'à quel point est-elle) engendrée par un système de transition ? Les résultats obtenus nous permettront plus tard de répondre à la question plus générale : étant donnée une preuve d'une propriété relative à une sémantique est-il possible (ou jusqu'à quel point est-il possible) de la remplacer par une preuve d'une autre propriété relative à une autre sémantique ou à

2.1 DEFINITION DE LA SEMANTIQUE PAR UN ENSEMBLE DE TRACES COMPLETES

Dans ce paragraphe nous définissons formellement la notion de trace. Une trace représente la séquence finie (ou infinie) des états intermédiaires au cours d'une exécution achevée (respectivement, ayant bouclé) du programme. Chaque état se décompose généralement en un état contrôle (spécifiant le ou les points du programme où se trouve l'exécution) et un état mémoire (indiquant la valeur actuelle des identificateurs). Deux états successifs dans une trace sont séparés par une action qui est généralement le nom de l'opération atomique qui a permis de passer d'un état à son successeur.

Nous définissons ensuite une sémantique comme étant un ensemble de traces d'exécution.

2.1.1 TRACES

Une trace d'exécution p est une séquence non vide finie ou infinie d'états séparés par des actions.

$$p = A_0 \xrightarrow{a_0} A_1 \xrightarrow{a_1} A_2 \xrightarrow{a_2} A_3 \dots A_i \xrightarrow{a_i} A_{i+1} \dots$$

Intuitivement, cette trace est un modèle d'une exécution du programme qui commence dans l'état A_0, exécute une action nommée a_0 pour atteindre l'état A_1, puis l'action a_1 pour atteindre l'état A_2, etc. Si l'exécution se termine alors la trace est finie sinon elle est infinie. Les traces sont associées en effet au déroulement d'un calcul ordinaire qui ne se termine pas (et jamais un calcul "en cours"). Nous n'introduirons pas la notion de trace vide qui ne correspond à aucune exécution physique (puisque une exécution doit toujours commencer dans un état [illegible])

Exemple

L'exécution d'un programme qui exécute deux fois l'action a puis l'action b et se termine, peut se décrire par la trace finie :

$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 1$$

L'exécution d'un programme qui boucle en exécutant l'action a sans arrêt peut se décrire par la trace infinie :

$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \dots 0 \xrightarrow{a} 0 \dots$$

□

Une trace p sur un ensemble S d'états et un ensemble A d'actions est un triplet $\langle m, A, a\rangle$. La longueur m de la trace est un entier non nul ($m = \{0, \dots, m-1\} \in (\omega \setminus 0)$) si la trace est finie. C'est $\omega = \{0, 1, \dots\}$ si la trace est infinie. Par conséquent $m \in ((\omega+1) \setminus 0)$. $A \in (m \to S)$ est la séquence des états (le i ème état $A(i)$ en comptant à partir de zéro étant noté A_i). La séquence des actions est $a \in ((m-1) \to A)$, (la i ème action $a(i)$ en comptant à partir de zéro étant notée a_i).

Définition 2.1.1:1

L'ensemble des traces sur un ensemble S d'états et A d'actions est défini par :

- $\Sigma^m\langle S, A\rangle = \{\langle m, A, a\rangle : A \in (m \to S) \wedge a \in ((m-1) \to A)\}$
  (traces de longueur $m \in ((\omega+1) \setminus 0)$)
- $\Sigma^{<l}\langle S, A\rangle = \bigcup_{m \in (l \setminus 0)} \Sigma^m\langle S, A\rangle$
  (traces de longueur strictement inférieure à $l \in (\omega+2)$)
- $\Sigma^{\leq l}\langle S, A\rangle = \Sigma^{<l}\langle S, A\rangle \cup \Sigma^l\langle S, A\rangle$
  (traces de longueur inférieure ou égale à $l \in (\omega+1)$)

Nous appelerons $\Sigma^{<\omega}\langle S,A\rangle$ l'ensemble des traces finies, $\Sigma^{\omega}\langle S,A\rangle$ l'ensemble des traces infinies et $\Sigma^{\leq\omega}\langle S,A\rangle$ l'ensemble des traces sur S et A. L'ensemble des traces étant d'usage fréquent, nous poserons $\Sigma\langle S,A\rangle = \Sigma^{\leq\omega}\langle S,A\rangle$ et nous omettons le couple $\langle S,A\rangle$ si nous pouvons le déterminer sans ambiguité d'après le contexte.

La longueur d'une trace $p=\langle m,\Delta,a\rangle$ comptée en nombre d'états est $|p|=m$. La longueur de p comptée en nombre d'actions est $|\sharp|=m-1$. Le $i$ème état de p (compté à partir de zéro) est $p_i=\Delta_i$ pour $i\in|p|$. La $i$ème action de p (comptée à partir de zéro) est $\sharp_i=a_i$ pour $i\in|\sharp|$.

Le préfixe $p^{<m}$, $m\in((\omega+2)\setminus 0)$ (respectivement $p^{\leq m}$, $m\in(\omega+1)$) d'une trace $p=\langle m,\Delta,a\rangle$ est p si $m\geq m$ (respectivement $m+1\geq m$) sinon $\langle m,\Delta^{<m},a^{<m-1}\rangle$ (respectivement $\langle m+1,\Delta^{\leq m},a^{<m}\rangle$), où $f^{<m}$ (respectivement $f^{\leq m}$) est le préfixe (non vide) $\langle f_0,\ldots,f_{m-1}\rangle$ (respectivement $\langle f_0,\ldots,f_m\rangle$) d'une séquence $\langle f_0,\ldots,f_{m-1},f_m,\ldots\rangle$.

Le suffixe $p^{\geq m}$, $m+1<m$ (respectivement $p^{>m}$, $m<m$) d'une trace $p=\langle m,\Delta,a\rangle$ est $\langle m-(m+1),\Delta^{\geq m},a^{\geq m}\rangle$ (respectivement $\langle m-m,\Delta^{>m},a^{>m}\rangle$) où $f^{\geq m}$ (respectivement $f^{>m}$) est le suffixe (non vide) $\langle f_{m+1},\ldots\rangle$ (respectivement $\langle f_m,\ldots\rangle$) d'une séquence $\langle f_0,\ldots,f_m,f_{m+1},\ldots\rangle$.

La tranche $p^{<m,m'>}$ (respectivement $p^{\leq m,m'>}$, $p^{<m,m'\leq}$, $p^{\leq m,m'\leq}$) est $(p^{<m'})^{\geq m}$ (respectivement $(p^{\leq m'})^{\geq m}$, $(p^{<m'})^{>m}$, $(p^{\leq m'})^{>m}$).

La concaténation de $p\in\Sigma\langle S,A\rangle$ et $q\in\Sigma\langle S,A\rangle$ est $p^\frown q=p$ si $|p|=\omega$ sinon $|p|<\omega$ et si $p_{|\sharp|}=q_0$ alors $p^\frown q=r$ tel que $r^{<|p|}=p$ et $r^{\geq|p|}=q$ sinon $p_{|\sharp|}\neq q_0$ et $p^\frown q$ est indéfinie.

La concaténation de $p\in\Sigma\langle S,A\rangle$ et $q\in\Sigma\langle S,A\rangle$ via une action $a\in A$ est $p\xrightarrow{a}q=p$ si $|p|=\omega$ sinon $p\xrightarrow{a}q=r\in\Sigma\langle S,A\rangle$ tel que $r^{<|p|}=p$, $r^{\geq|p|}=q$ et $\sharp_{|p|}=a$.

Par abus de notation, nous notons s la trace de longueur 1 constituée du seul état s, c'est-à-dire $\langle 1,\{\langle 0,s\rangle\},0\rangle$. De ce fait une trace p finie peut se noter $p_0\xrightarrow{\sharp_0}p_1\cdots p_{|\sharp|-1}\xrightarrow{\sharp_{|\sharp|-1}}p_{|\sharp|}$ et une trace infinie par $p_0\xrightarrow{\sharp_0}p_1\cdots p_i\xrightarrow{\sharp_i}p_{i+1}\cdots$ et donc par abus de notation, nous écrirons $\langle p_i\xrightarrow{\sharp_i}p_{i+1}:i\in|\sharp|\rangle$ (par analogie avec l'écriture $\langle f_i:i\in I\rangle$ de la fonction F telle que $\forall i\in I.\,F(i)=f_i$).

Exemple

$p=p_0\xrightarrow{\sharp_0}p_1\xrightarrow{\sharp_1}p_2$

$q=q_0\xrightarrow{q_0}q_1\xrightarrow{q_1}q_2\xrightarrow{q_2}q_3\xrightarrow{q_3}q_4\cdots q_i\xrightarrow{q_i}q_{i+1}\cdots$

$|p|=3$, $|\sharp|=2$, $|q|=|q_\sharp|=\omega$

$p^{<0}$ est indéfini, $p^{<1}=p_0$, $p^{<2}=p_0\xrightarrow{\sharp_0}p_1$, $p^{<m}=p$ pour $m\geq 3$

$q^{<0}$ est indéfini, $q^{<1}=q_0$, $q^{<m}=q_0\xrightarrow{q_0}q_1\cdots\xrightarrow{q_{m-2}}q_{m-1}$ pour $m\geq 2$, $q^{<\omega}=q$

$p^{>0}=p$, $p^{>1}=p_1\xrightarrow{\sharp_1}p_2$, $p^{>2}=p_2$, $p^{>m}$ est indéfini pour $m\geq 3$

$q^{>m}=q_m\xrightarrow{q_m}q_{m+1}\cdots q_i\xrightarrow{q_i}q_{i+1}\cdots$ pour $m\geq 0$, $q^{>\omega}$ est indéfini

$q^{\langle 2,4\rangle}=q_2\xrightarrow{q_2}q_3$

$q\xrightarrow{a}p=q$

$p\xrightarrow{a}q=p_0\xrightarrow{\sharp_0}p_1\xrightarrow{\sharp_1}p_2\xrightarrow{a}q_0\xrightarrow{q_0}q_1\cdots q_i\xrightarrow{q_i}q_{i+1}\cdots$

□

2.1.2 SEMANTIQUE

La sémantique d'un langage associe à tout programme un ensemble de traces car l'exécution d'un programme (par exemple parallèle) est en général non déterministe. Un ensemble de traces peut également être nécessaire pour décrire les exécutions d'un programme déterministe si par exemple les états initiaux correspondant à des données différentes sont différents.

La sémantique d'un langage $\mathcal{LP}$ associe à tout programme Pr de $\mathcal{LP}$ un ensemble non vide $S[\![Pr]\!]$ d'états, un ensemble non vide $A[\![Pr]\!]$ d'actions et un ensemble $\Sigma[\![Pr]\!] \subseteq \Sigma\langle S[\![Pr]\!], A[\![Pr]\!]\rangle$ de traces.

Exemple 2.1.2-1

La sémantique d'un programme qui exécute de zéro à deux fois l'action a puis l'action b peut se décrire par la sémantique $\langle S, A, \Sigma\rangle$ où $S = \{0,1\}$, $A = \{a,b\}$ et $\Sigma = \{ 0 \xrightarrow{b} 1,\ 0 \xrightarrow{a} 0 \xrightarrow{b} 1,\ 0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 1 \}$

□

Exemple 2.1.2-2

La sémantique d'un programme qui exécute un nombre quelconque mais fini de fois l'action a puis l'action b peut se décrire par la sémantique $\langle S, A, \Sigma\rangle$ où $S = \{0,1\}$, $A = \{a,b\}$ et

$$\Sigma = \{\ 0 \xrightarrow{b} 1$$
$$0 \xrightarrow{a} 0 \xrightarrow{b} 1$$
$$\dots$$
$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \dots 0 \xrightarrow{a} 0 \xrightarrow{b} 1\ \}$$

□

Exemple 2.1.2-3

La sémantique d'un programme qui exécute un nombre fini de fois l'action a puis l'action b ou exécute un nombre infini de fois l'action a peut se décrire par la sémantique $\langle S, A, \Sigma\rangle$ où $S = \{0,1\}$, $A = \{a,b\}$ et

$$\Sigma = \{\ \dots$$
$$0 \xrightarrow{a} 0 \dots 0 \xrightarrow{a} 0 \xrightarrow{b} 1$$
$$\dots$$
$$0 \xrightarrow{a} 0 \dots 0 \xrightarrow{a} 0 \dots\ \}$$

□

Définition 2.1.2:1 Sémantiques

L'ensemble des sémantiques sur des ensembles $\mathcal{S}$ (d'états) et $\mathcal{A}$ (d'actions) est $\mathrm{Sem}\langle \mathcal{S}, \mathcal{A}\rangle = \{ \langle S, A, \Sigma\rangle : S \in \mathcal{S} \wedge A \in \mathcal{A} \wedge \Sigma \in \Sigma\langle S, A\rangle \}$

2.2 DEFINITION DES SYSTEMES DE TRANSITION

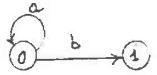
Etant donnés des ensembles S d'états et A d'actions, un système de transition formalise la notion d'état initiaux d'un programme (comme un sous-ensemble de S caractérisé par une fonction ε à valeurs de vérité (tt vrai, ff faux)) et de pas du programme correspondant à l'exécution d'une action $a \in A$ (comme une relation de transition t_a entre un état et ses successeurs possibles).

Définition 2.2:1 Systèmes de transition

L'ensemble des systèmes de transition sur des ensembles $\mathscr{S}$ (d'états) et $\mathscr{A}$ (d'actions) est $Tran\langle\mathscr{S},\mathscr{A}\rangle = \{\langle S,A,t,\varepsilon\rangle : S\in\mathscr{S} \wedge A\in\mathscr{A} \wedge t\in(A\to(S\times S\to\{tt,ff\})) \wedge \varepsilon\in(S\to\{tt,ff\})\}$.

Exemple 2.2-1

Un programme qui exécute l'action b ou bien exécute un nombre fini de fois l'action a puis l'action b ou bien un nombre infini de fois l'action a, peut-être informellement représenté par l'automate fini suivant



que nous formalisons par le système de transition $\langle S,A,t,\varepsilon\rangle$ où $S=\{0,1\}$, $A=\{a,b\}$, $t_a(\Delta,\Delta') = [\Delta=\Delta'=0]$, $t_b(\Delta,\Delta') = [\Delta=0 \wedge \Delta'=1]$ et $\varepsilon(\Delta) = [\Delta=0]$.

□

2.3 SYSTEME DE TRANSITION ENGENDRE PAR UNE SEMANTIQUE

Pour formaliser les notions d'"état initial" et de "pas de calcul" pour une sémantique $\langle S,A,\Sigma\rangle$, nous définissons le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle$ qu'elle engendre. (Pour alléger les notations, nous laisserons implicites certains paramètres quand ils peuvent être aisément déterminés d'après le contexte. Par exemple nous écrirons $\langle S,A,t,\varepsilon\rangle$ au lieu de $\langle S,A,t\langle S,A,\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle$).

Définition 2.3:1 Système de transition engendré par une sémantique

Les états initiaux engendrés par une sémantique $\langle S,A,\Sigma\rangle$ sont caractérisés par

$\varepsilon \in (S\to\{tt,ff\})$

$\varepsilon(s) = [\exists p\in\Sigma.\ p_1 = s]$

La relation de transition engendrée par une sémantique $\langle S,A,\Sigma\rangle$ est définie par :

$t \in (A\to(S\times S\to\{tt,ff\}))$

$t_a(s,s') = [\exists p\in\Sigma, i\in|p|.\ (p_i = s \wedge p_i = a \wedge p_{i+1} = s')]$

Le système de transition engendré par une sémantique $\langle S,A,\Sigma\rangle$ est $\langle S,A,t\langle S,A,\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle$ où $t\langle S,A,\Sigma\rangle$ et $\varepsilon\langle S,A,\Sigma\rangle$ sont respectivement la relation de transition et les états initiaux engendrés par $\langle S,A,\Sigma\rangle$.

Exemple

Les sémantiques données en exemples 2.1.2-1, 2.1.2-2 et 2.1.2-3 engendrent exactement le même système de transition 2.2-1.

□

2.4 SEMANTIQUE ENGENDREE PAR UN SYSTEME DE TRANSITION

La description des comportements possibles d'un programme par un système de transition est souvent plus simple à utiliser voire à comprendre que la description par un ensemble de traces. Pour répondre à la question de savoir dans quelles conditions la donnée d'une sémantique $\langle S, A, \Sigma \rangle$ d'un programme est équivalente à la donnée du système de transition $\langle S, A, t, \varepsilon \rangle$ engendré par cette sémantique, nous définirons l'ensemble des traces complètes engendrées par un système de transition quelconque.

Définition 2.4:1

La sémantique engendrée par un système de transition $\langle S, A, t, \varepsilon \rangle$ est $\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$ avec :

- $\Sigma^n\langle S, A, t, \varepsilon\rangle = \{p \in \Sigma^n\langle S, A\rangle : \varepsilon(p_0) \wedge \forall i \in [\![0, n-1[\![.\ t_{a_i}(p_i, p_{i+1}) \wedge \forall a \in A, \forall s \in S .\ \neg t_a(p_{n-1}, s)\}$
  (Traces complètes finies de longueur $n \in (\omega \cup \{\omega\})$)

- $\Sigma^\omega\langle S, A, t, \varepsilon\rangle = \{p \in \Sigma^\omega\langle S, A\rangle : \varepsilon(p_0) \wedge \forall i \in \omega .\ t_{a_i}(p_i, p_{i+1})\}$
  (Traces infinies)

- $\Sigma\langle S, A, t, \varepsilon\rangle = \bigcup_{n \in ((\omega+1) \cup \{\omega\})} \Sigma^n\langle S, A, t, \varepsilon\rangle$
  (Traces complètes)

Exemple 2.4-1

La sémantique engendrée par le système de transition donné en exemple 2.2-1 est définie en 2.1.2-3.

□

Dans la suite nous utiliserons les notations suivantes :

(en particulier $\Sigma^{<\omega}\langle S, A, t, \varepsilon\rangle$ est l'ensemble des traces finies engendrées par le système de transition $\langle S, A, t, \varepsilon\rangle$).

$\Sigma^{\leq \ell}\langle S, A, t, \varepsilon\rangle = (\Sigma^{<\ell}\langle S, A, t, \varepsilon\rangle \cup \Sigma^{\ell}\langle S, A, t, \varepsilon\rangle)$, (traces de longueur inférieure ou égale à $\ell \in (\omega+1)$).

2.5 RELATIONS ENTRE SEMANTIQUES ET ENTRE SYSTEMES DE TRANSITION

Notons que le système de transition $\langle S,A,t\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle, \varepsilon\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle\rangle$ engendré par la sémantique $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ engendrée par le système de transition $\langle S,A,t,\varepsilon\rangle$ est $\langle S,A,t,\varepsilon\rangle$ lui-même. - Par contre les exemples 2.3-1 et 2.4-1 montrent que la sémantique $\langle S,A,\Sigma\langle S,A,t\langle S,A\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ engendrée par le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ engendré par la sémantique $\langle S,A,\Sigma\rangle$ n'est en général pas cette sémantique $\langle S,A,\Sigma\rangle$ elle-même. Ceci nous amène donc à étudier dans ce paragraphe les relations qui peuvent exister entre sémantiques. Ensuite, au paragraphe 2.6, il nous sera possible d'énoncer une condition nécessaire et suffisante pour qu'une sémantique soit close (c'est-à-dire égale à la sémantique engendrée par le système de transition qu'elle engendre).

Nous incluons également dans ce paragraphe 2.5, la définition de relations entre sémantiques qu'il est nécessaire d'étudier pour justifier certaines méthodes de preuve de propriétés de programmes. En effet, pour certaines classes de propriétés, les démonstrations de correction se font plus facilement en faisant abstraction de certains aspects de la sémantique des programmes. On raisonne donc non pas sur la sémantique exacte du programme mais sur une sémantique approchée qui lui correspond selon une relation qui conserve la propriété à démontrer. Par exemple les preuves de propriétés d'invariance (correction partielle, exclusion mutuelle, ...) de programmes parallèles avec exécution équitable des processus concurrents se font beaucoup plus aisément en raisonnant sur le programme parallèle non équitable correspondant.

Pour formaliser cette méthode de preuve considérons (pour l'instant en simplifiant) qu'une propriété d'un programme est une relation entre une spécification $Sp \in \underline{Spec}$ et la sémantique $\langle S,A,\Sigma\rangle \in \underline{Sem}\langle\mathcal{D},\mathcal{A}\rangle$ du programme. Nous avons $P \in (\underline{Spec} \times \underline{Sem}\langle\mathcal{D},\mathcal{A}\rangle) \to \{tt,ff\}$.

La preuve se fait en raisonnant sur une propriété $P' \in ((\underline{Spec} \times \underline{Sem}\langle\mathcal{D},\mathcal{A}\rangle) \to \{tt,ff\})$ reliant une spécification Sp' et une sémantique approchée $\langle S',A',\Sigma'\rangle$ du programme.

Pour justifier cette méthode de preuve, il faut montrer que la sémantique approchée $\langle S',A',\Sigma'\rangle$ du programme est liée à la sémantique exacte du programme $\langle S,A,\Sigma\rangle$ par une <u>relation entre sémantiques</u> $R \in (\underline{Sem}\langle\mathcal{D},\mathcal{A}\rangle \times \underline{Sem}\langle\mathcal{D},\mathcal{A}\rangle) \to \{tt,ff\})$ qui conserve la propriété originale P à démontrer:

$$[P'(Sp', \langle S',A',\Sigma'\rangle) \wedge R(\langle S',A',\Sigma'\rangle, \langle S,A,\Sigma\rangle)] \Rightarrow P(Sp, \langle S,A,\Sigma\rangle)$$

(En général, la preuve est faite pour toutes spécifications Sp' et Sp liées par une certaine relation entre spécifications).

Enfin, une relation R entre sémantiques induit une <u>relation entre systèmes de transition</u>, également notée R et définie par:

$R \in ((\underline{Tran}\langle\mathcal{D},\mathcal{A}\rangle \times \underline{Tran}\langle\mathcal{D},\mathcal{A}\rangle) \to \{tt,ff\})$

$R(\langle S',A',t',\varepsilon'\rangle, \langle S,A,t,\varepsilon\rangle) = R(\langle S',A',\Sigma\langle S',A',t',\varepsilon'\rangle\rangle, \langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

De la même façon une relation R entre systèmes de transition induit une relation entre sémantiques, également notée R et définie par:

$R \in ((\underline{Sem}\langle\mathcal{D},\mathcal{A}\rangle \times \underline{Sem}\langle\mathcal{D},\mathcal{A}\rangle) \to \{tt,ff\})$

$R(\langle S',A',\Sigma'\rangle, \langle S,A,\Sigma\rangle) = R(\langle S',A',t\langle S',A',\Sigma'\rangle, \varepsilon\langle S',A',\Sigma'\rangle\rangle, \langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle)$

Il est donc intéressant d'étudier pour toute relation R entre sémantiques ou systèmes de transition la relation induite respectivement entre systèmes de transition et sémantiques. Donnons maintenant des exemples de relations, entre sémantiques et entre systèmes de transition, qui seront utilisées ultérieurement

2.5.1 INCLUSION DE SEMANTIQUES ET DE SYSTEMES DE TRANSITION

On utilise la relation d'inclusion entre sémantiques quand on fait des preuves de programmes relatives à un sur- ou sous-ensemble des traces d'exécution du programme. Par exemple, on peut quelquefois démontrer une propriété d'invariance (correction partielle, ...) ou de fatalité (terminaison, ...) d'un programme parallèle en ignorant les hypothèses d'exécution équitable des processus concurrents (tout processus indéfiniment activable est fatalement activé, ...). Ceci vient du fait que ces propriétés étant vraies pour une sémantique $\langle S', A', \Sigma' \rangle$ le sont également pour toute sémantique $\langle S', A', \Sigma \rangle$ telle que $\Sigma \subseteq \Sigma'$.

Plus généralement, la relation d'inclusion entre sémantiques est définie par $\langle S, A, \Sigma \rangle \leq \langle S', A', \Sigma' \rangle$ si et seulement si $[S \subseteq S' \wedge A \subseteq A' \wedge \Sigma \subseteq \Sigma']$.

Muni de cet ordre partiel réflexif, $\underline{sem} \langle \mathcal{P}, \mathcal{A} \rangle$ est un treillis complet dont l'infimum est $\langle 0,0,0 \rangle$ et le supremum $\langle \mathcal{P}, \mathcal{A}, \Sigma \langle \mathcal{P}, \mathcal{A} \rangle \rangle$, (où 0 désigne l'ensemble vide, cf. annexe I-2).

Le lemme qui suit caractérise la relation d'inclusion $\leq$ entre systèmes de transition induite par l'inclusion de sémantiques et qui est donc définie par :

$$[\langle S, A, t, \varepsilon \rangle \leq \langle S', A', t', \varepsilon' \rangle] \Leftrightarrow [S \subseteq S' \wedge A \subseteq A' \wedge \Sigma \langle S, A, t, \varepsilon \rangle \subseteq \Sigma \langle S', A', t', \varepsilon' \rangle]$$

Pour exprimer ce lemme nous définissons :

- $\underline{Acc}\ \langle S, A, t, \varepsilon \rangle(A) = [\exists p \in \Sigma \langle S, A, t, \varepsilon \rangle, i \in |p| \,.\, A = p_i]$
  (qui caractérise les états accessibles d'un système de transition)
- $\underline{Blo}\ \langle S, A, t, \varepsilon \rangle(A) = [\forall a \in \mathcal{A}, A' \in S \,.\, \neg t_a(A, A')]$
  (qui caractérise les états de blocage d'un système de transition)

Lemme 2.5.1 ∨ 1

$$[\langle S, A, t, \varepsilon \rangle \leq \langle S', A', t', \varepsilon' \rangle]$$
$$\Leftrightarrow [S \subseteq S' \wedge A \subseteq A' \wedge (\forall a \in \mathcal{A} \,.\, t_a 1 Acc \Rightarrow t'_a) \wedge (Blo \wedge Acc \Rightarrow Blo') \wedge (\varepsilon \Rightarrow \varepsilon')]$$

avec $Acc = \underline{Acc}\langle S, A, t, \varepsilon \rangle$, $Blo = \underline{Blo}\langle S, A, t, \varepsilon \rangle$ et $Blo' = \underline{Blo}\langle S', A', t', \varepsilon' \rangle$.

Démonstration

($\Rightarrow$) Si $\langle S, A, t, \varepsilon \rangle \leq \langle S', A', t', \varepsilon' \rangle$ alors par définition on a $\Sigma \langle S, A, t, \varepsilon \rangle \subseteq \Sigma \langle S', A', t', \varepsilon' \rangle$. Si $t_a 1 Acc(A, A')$ est vrai, il existe une trace p de $\Sigma \langle S, A, t, \varepsilon \rangle$ et $i \in |p|$ tels que $A = p_i$ et $t_a(A, A')$ et donc une trace p' de $\Sigma \langle S, A, t, \varepsilon \rangle$ telle que $(i+1) \in |p'| \wedge A = p'_i \wedge A' = p'_{i+1}$. Comme $p' \in \Sigma \langle S', A', t', \varepsilon' \rangle$, la définition 2.4:1 entraîne que $t'_a(A, A')$ est vrai. De même si $(Blo \wedge Acc)(A)$ est vrai, il existe une trace finie p de $\Sigma \langle S, A, t, \varepsilon \rangle$ de longueur n telle que $p_{n-1} = A \wedge \forall a \in \mathcal{A}, A' \in S \,.\, \neg t_a(A, A')$. Comme $p \in \Sigma \langle S', A', t', \varepsilon' \rangle$, la définition 2.4:1 entraîne que $\forall a \in \mathcal{A}, A' \in S \,.\, \neg t'_a(p_{n-1}, A')$ et donc $Blo'(A)$. Enfin si $\varepsilon(A)$ alors d'après 2.4:1 il existe $p \in \Sigma \langle S, A, t, \varepsilon \rangle$ telle que $p_0 = A$. Comme $p \in \Sigma \langle S', A', t', \varepsilon' \rangle$ nous avons $\varepsilon'(p_0)$ et donc $\varepsilon'(A)$.

($\Leftarrow$) Il faut démontrer que $\Sigma \langle S, A, t, \varepsilon \rangle \subseteq \Sigma \langle S', A', t', \varepsilon' \rangle$. Si $p \in \Sigma \langle S, A, t, \varepsilon \rangle$ nous avons $\varepsilon(p_0)$ et donc $\varepsilon'(p_0)$. Si $i \in |p|$ alors nous avons $Acc(p_i) \wedge t_{B_i}(p_i, p_{i+1})$ et par conséquent $t'_{B_i}(p_i, p_{i+1})$. Si p est une trace infinie, nous avons démontré que $p \in \Sigma \langle S', A', t', \varepsilon' \rangle$. Si p est une trace finie de longueur n, nous avons $(Acc \wedge Blo)(p_{n-1})$, donc $Blo'(p_{n-1})$, qui d'après 2.4:1, implique $p \in \Sigma \langle S', A', t', \varepsilon' \rangle$.

□

2.5.2 EQUIVALENCE DE SYSTEMES DE TRANSITION

La relation d'égalité entre sémantiques induit une relation d'équivalence entre systèmes de transition définie par :

$$[\langle S,A,t,\varepsilon\rangle \equiv \langle S',A',t',\varepsilon'\rangle] \Leftrightarrow [S=S' \wedge A=A' \wedge \Sigma\langle S,A,t,\varepsilon\rangle = \Sigma\langle S',A',t',\varepsilon'\rangle]$$

(Par dérogation à la convention de 2.5, nous notons $\equiv$ la relation induite, réservant $=$ à l'égalité de systèmes de transition).

Lemme 2.5.2.v1

$$\begin{array}{ll} & [\langle S,A,t,\varepsilon\rangle \equiv \langle S',A',t',\varepsilon'\rangle] \\ \Leftrightarrow & \\ & [S=S' \wedge A=A' \wedge (\forall a\in A.\ t_a 1 Acc = t'_a 1 Acc') \wedge (Blo \wedge Acc = Blo' \wedge Acc') \wedge \varepsilon=\varepsilon'] \\ \Rightarrow & \\ & [Acc = Acc'] \end{array}$$

avec $Acc = \underline{Acc}\langle S,A,t,\varepsilon\rangle$, $Acc' = \underline{Acc}\langle S',A',t',\varepsilon'\rangle$, $Blo = \underline{Blo}\langle S,A,t,\varepsilon\rangle$ et $Blo' = \underline{Blo}\langle S',A',t',\varepsilon'\rangle$

démonstration

Si $\langle S,A,t,\varepsilon\rangle \equiv \langle S',A',t',\varepsilon'\rangle$ alors $\Sigma\langle S,A,t,\varepsilon\rangle = \Sigma\langle S',A',t',\varepsilon'\rangle$ et donc par définition de $\underline{Acc}$, nous avons $\underline{Acc}\langle S,A,t,\varepsilon\rangle = \underline{Acc}\langle S',A',t',\varepsilon'\rangle$.

$\langle S,A,t,\varepsilon\rangle \equiv \langle S',A',t',\varepsilon'\rangle$ si et seulement si $\langle S,A,t,\varepsilon\rangle \subseteq \langle S',A',t',\varepsilon'\rangle$ et $\langle S',A',t',\varepsilon'\rangle \subseteq \langle S,A,t,\varepsilon\rangle$ et donc d'après le lemme 2.5.1v1, si et seulement si $S=S' \wedge A=A' \wedge \forall a\in A.(t_a 1 Acc \Rightarrow t'_a \wedge t'_a 1 Acc' \Rightarrow t_a) \wedge Blo \wedge Acc \Rightarrow Blo' \wedge Blo' \wedge Acc' \Rightarrow Blo \wedge \varepsilon=\varepsilon'$.
Pour tout $a\in A=A'$, nous avons $[t_a 1 Acc \Rightarrow t'_a \wedge t'_a 1 Acc' \Rightarrow t_a] \Rightarrow [t_a 1 Acc \Rightarrow t'_a 1 Acc = t'_a 1 Acc' \Rightarrow t_a 1 Acc' = t_a 1 Acc] = [t_a 1 Acc = t'_a 1 Acc'] \Rightarrow [t_a 1 Acc \Rightarrow t'_a \wedge t'_a 1 Acc' \Rightarrow t_a]$. De plus $[Blo \wedge Acc \Rightarrow Blo' \wedge Blo' \wedge Acc' \Rightarrow Blo] = [Blo \wedge Acc = Blo' \wedge Acc']$ car $Acc = Acc'$.
□

En cas de besoin, nous pourrons [illegible] au [illegible] nous servant que en [illegible] sur [illegible] [illegible].

2.5.3 CONCORDANCE ENTRE SEMANTIQUES ET ENTRE SYSTEMES DE TRANSITION A DES RELATIONS ENTRE ETATS ET/OU ACTIONS PRES

La relation de concordance entre sémantiques sera utilisée dans les justifications de méthodes de preuves pour éliminer des informations contenues dans les états ou actions. Il s'agit par exemple d'éliminer les variables auxiliaires introduites pour faciliter une démonstration ou d'identifier toutes les actions d'un même processus d'un programme parallèle.

Soient $rs \in (\mathcal{S}\times\mathcal{S} \to \{tt,ff\})$ une relation entre états et $ra \in (\mathcal{A}\times\mathcal{A} \to \{tt,ff\})$ une relation entre actions.

La relation de concordance entre traces aux relations rs entre états et ra entre actions près est définie par :

$$\simeq\langle rs,ra\rangle \in (\Sigma\langle\mathcal{S},\mathcal{A}\rangle \times \Sigma\langle\mathcal{S},\mathcal{A}\rangle \to \{tt,ff\})$$

$$\simeq\langle rs,ra\rangle(p,q) = [|p|=|q| \wedge \forall i\in|p|.rs(p_i,q_i) \wedge \forall i\in|p|.ra(p_i,q_i)]$$

La relation de concordance entre sémantiques aux relations rs entre états et ra entre actions près est définie par :

$$\simeq\langle rs,ra\rangle \in (Sem\langle\mathcal{S},\mathcal{A}\rangle \times Sem\langle\mathcal{S},\mathcal{A}\rangle \to \{tt,ff\})$$

$$\simeq\langle rs,ra\rangle(\langle S,A,\Sigma\rangle,\langle S',A',\Sigma'\rangle) = [S'=rs[S] \wedge A'=ra[A] \wedge \Sigma'=\simeq\langle rs,ra\rangle[\Sigma]]$$

La relation induite entre systèmes de transition est la relation de concordance entre systèmes de transition aux relations rs entre états et ra entre actions près qui est définie par :

$$\simeq\langle rs,ra\rangle \in (Tran\langle\mathcal{S},\mathcal{A}\rangle \times Tran\langle\mathcal{S},\mathcal{A}\rangle \to \{tt,ff\})$$

$$\simeq\langle rs,ra\rangle(\langle S,A,t,\varepsilon\rangle,\langle S',A',t',\varepsilon'\rangle) = \simeq\langle rs,ra\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle,\langle S',A',\Sigma\langle S',A',t',\varepsilon'\rangle\rangle)$$

Lemme 2.5.3~1

Si

$$(\exists A \in S.\ \varepsilon(A) \ \wedge\ rs(A,A')) = \varepsilon^{\#}(A') \qquad (a)$$
$$\wedge\ (ra(a,a') \wedge rs^{-1}(A'_0,A_0) \wedge t_a(A_0,A_1) \wedge rs(A_1,A'_1)) \Rightarrow t^{\#}_{a'}(A'_0,A'_1) \qquad (b)$$
$$\wedge\ (rs(A_0,A'_0) \wedge t^{\#}_{a'}(A'_0,A'_1)) \Rightarrow (\exists A_1 \in S,\ a \in A.\ rs(A_1,A'_1) \wedge ra(a,a') \wedge t_a(A_0,A_1)) \qquad (c)$$
$$\wedge\ (rs(A_0,A'_0) \wedge t_a(A_0,A_1)) \Rightarrow (\exists A'_1 \in S',\ a' \in A'.\ t^{\#}_{a'}(A'_0,A'_1)) \qquad (d)$$

alors

$$\Sigma\langle rs[S], ra[S], t^{\#}, \varepsilon^{\#}\rangle = \{q \in \Sigma\langle rs[S], ra[A]\rangle.\ \exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \simeq\langle rs, ra\rangle(p,q)\}$$

et donc

$$\simeq\langle rs, ra\rangle(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) \Leftrightarrow (\langle S',A',t',\varepsilon'\rangle \equiv \langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle)$$

Démonstration

Commençons par montrer que $\Sigma\langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle = \{q \in \Sigma\langle rs[S], ra[A]\rangle.\ \exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \simeq\langle rs, ra\rangle(p,q)\}$.

Montrons que si $\exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \simeq\langle rs, ra\rangle(p,q)$ alors $q \in \Sigma\langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle$. D'après la définition de $\simeq\langle rs, ra\rangle(p,q)$, 2.4.1, (a) et (b), nous avons $|p| = |q|$, $\varepsilon(p_0) \wedge rs(p_0,q_0) \Rightarrow \varepsilon^{\#}(q_0)$, $\forall i \in |p|.\ (ra(\beta_i, \gamma_i) \wedge rs(p_i,q_i) \wedge t_{\beta_i}(p_i,p_{i+1}) \wedge rs(p_{i+1},q_{i+1})) \Rightarrow t^{\#}_{\gamma_i}(q_i,q_{i+1})$ et donc $q \in \Sigma\langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle$ si p et donc q est une trace infinie. Si p est une trace finie de longueur n, nous avons $\forall a' \in A', A \subseteq S.\ \neg t^{\#}_{a'}(q_{n-1}, A')$ car sinon $rs(p_{n-1}, q_{n-1})$ et (c) impliqueraient $\forall A \in S, a \in A.\ t_a(p_{n-1}, A)$ en contradiction avec l'hypothèse que p est de longueur n.

Si $q \in \Sigma\langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle$ alors $\exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \simeq\langle rs, ra\rangle(p,q)$. Nous construisons p tel que $|p| = |q|$ comme suit : d'après (a), $\varepsilon^{\#}(q_0) \Rightarrow (\exists p_0 \in S.\ \varepsilon(p_0) \wedge rs(p_0,q_0))$. Disposant de p_i tel que $rs(p_i,q_i)$ et $i \in |q|$, nous avons $t^{\#}_{\gamma_i}(q_i,q_{i+1})$ et d'après (c), il existe p_{i+1} et β_i tels que $rs(p_{i+1},q_{i+1})$, $ra(\beta_i,\gamma_i)$ et $t_{\beta_i}(p_i,p_{i+1})$. Si q et donc p est infinie, ceci montre que $p \in \Sigma\langle S,A,t,\varepsilon\rangle$. Si q est finie de longueur n, il reste à montrer que $\forall a \in A, A \in S.\ \neg t_a(p_{n-1}, A)$. Dans le cas contraire si (p_{n-1}, q_{n-1}) est (par hypothèse inductive d'existence de p) tel que $t^{=}_{a'}(q_{n-1})$, on obtient une contradiction avec la définition dans d'une trace finie de longueur n.

Comme conséquence immédiate, nous obtenons $\simeq\langle rs, ra\rangle(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) \Leftrightarrow (\langle S',A',t',\varepsilon'\rangle \equiv \langle rs[S], ra[A], t^{\#}, \varepsilon^{\#}\rangle)$ en observant que $S' = rs[S]$ et $A' = ra[A]$.

□

Nous utiliserons principalement trois cas particuliers :

2.5.3.1 Concordance à une fonction des états près

Il s'agit du cas où rs est une fonction $f_S \in (S \to S')$ et ra la relation d'identité $\underline{1}$.

La sémantique concordante à $\langle S,A,\Sigma\rangle$ à la fonction $f_S \in (S \to S')$ des états près est donc $\simeq\langle f_S\rangle(\langle S,A,\Sigma\rangle) = \langle f_S[S], A, \{\langle m, f_S(A), a\rangle : \langle m,A,a\rangle \in \Sigma\}\rangle$ et f_S est étendue à $(\Sigma\langle S,A\rangle \to \Sigma\langle S',A'\rangle)$ par $\forall i \in |A|.\ f_S(A)_i = f_S(A_i)$.

2.5.3.2 Concordance à l'annulation des états près

– Quand les preuves de correction d'un programme portent uniquement sur les actions des traces, il est parfois possible d'éliminer les états en les identifiant à un état unique noté par convention $\underline{\Omega}$.

Nous définissons la sémantique concordante à $\langle S,A,\Sigma\rangle$ à l'annulation des états près comme la sémantique concordante à la fonction $f_S \in (S \to \{\underline{\Omega}\})$ près définie par $\forall A \in S.\ f_S(A) = \underline{\Omega}$.

– Si une sémantique $\langle S,A,\Sigma\rangle$ ne comporte qu'un seul état, nous pouvons identifier les traces $\langle m,A,a\rangle$ à des séquences d'actions a et l'ensemble Σ des traces à l'ensemble des séquences d'actions $\{a : \exists m, A.\ \langle m,A,a\rangle \in \Sigma\}$ quand elles ne sont pas vides sinon $\{\underline{a}\}$ où $\underline{a}$ désigne par convention une action unique. Dans ce cas cette sémantique peut donc être directement définie par un couple $\langle A,\Sigma\rangle$ où A est un ensemble d'actions et $\Sigma \in 1^{A^{\omega}} - \{\}$

- Remarquons que nous pouvons toujours, sans perte d'informations, ramener une sémantique $\langle S, A, \Sigma \rangle$ à une sémantique $\langle S', A', \Sigma' \rangle$ ne comportant qu'un seul état en choisissant $S' = \{\underline{a}\}$, $A' = S \times (A \cup \{\underline{a}\})$ où $\underline{a} \notin A$ (c'est-à-dire que les actions incluent l'état dans lequel elles sont effectuées et que nous rajoutons une action unique après le dernier état d'une trace finie) et $\Sigma' = \{\underline{a} \xrightarrow{\langle p_0, \sharp_0 \rangle} \underline{a} \cdots \underline{a} \xrightarrow{\langle p_{|\sharp|-1}, \sharp_{|\sharp|-1} \rangle} \underline{a} \xrightarrow{\langle p_{|\sharp|}, \underline{a} \rangle} \underline{a} : p \in \Sigma \wedge |p| < \omega\}$ $\cup \{\langle \underline{a} \xrightarrow{\langle p_i, \sharp_i \rangle} \underline{a} : i \in \omega \rangle : p \in \Sigma \wedge |p| = \omega\}$. Cet argument montre que nous pouvons toujours raisonner sur les actions, ce que nous faisons rarement.

2.5.3.3 Concordance à l'annulation des actions près

- Quand les preuves de correction d'un programme (comme la correction partielle) portent uniquement sur les états, il est parfois possible d'éliminer les actions. Il se peut également que les actions n'aient pas besoin d'être notées dans les traces parce que les états comportent des informations de contrôle suffisamment riches pourque la transition entre deux états ne puisse correspondre qu'à une seule action qu'il est possible de déterminer sans ambiguité à partir de ces informations de contrôle. Eliminer les actions revient à les identifier toutes à une action unique $\underline{a}$ en choisissant $S' = S$, $A' = \{\underline{a}\}$, $ns = \underline{1}$ et $na(a, a') = [a' := \underline{a}]$.

- Si une sémantique $\langle S, A, \Sigma \rangle$ ne comporte qu'une seule action, nous pouvons identifier les traces $\langle m, p, a \rangle$ à des séquences non vides d'états p et l'ensemble des traces Σ à $\{p : \exists m, a . \langle m, p, a \rangle \in \Sigma\}$. Cette sémantique peut donc être définie par un couple $\langle S, \Sigma \rangle$ où S est un ensemble d'états et $\Sigma \subseteq S^{*\omega}$.

- Remarquons que nous pouvons toujours ramener une sémantique $\langle S, \Sigma \rangle$ à une sémantique $\langle S', \Sigma' \rangle$ ne comportant qu'une seule action en choisissant $S' = S \times (A \cup \{\underline{a}\})$, où $\underline{a} \notin A$ (c'est-à-dire que les états incluent la dernière action effectuée ou $\underline{a}$ si c'est la première), $A' = \{\underline{a}\}$ et $\Sigma' = \{\langle p_0, \sharp_0 \rangle \xrightarrow{\underline{a}} \cdots \xrightarrow{\underline{a}} \langle p_{|\sharp|}, \underline{a} \rangle : p \in \Sigma \wedge |p| < \omega\} \cup \{\langle \langle p_i, \sharp_i \rangle \xrightarrow{\underline{a}} \langle p_{i+1}, \sharp_{i+1} \rangle : i \in \omega \rangle : p \in \Sigma \wedge |p| = \omega\}$. Cet argument montre que nous pouvons toujours raisonner uniquement sur des états, ce que nous faisons souvent.

2.5.4 REDUCTION DE SEMANTIQUES

Souvent, une preuve de correction d'un programme est simplifiée en ne tenant pas compte de certains états intermédiaires du programme. Par exemple, la compilation de langages de haut niveau traduit l'évaluation d'une expression arithmétique en une suite d'instructions machine comportant des points de contrôle et des registres intermédiaires. Sous certaines conditions (absence d'effets de bord, ...) il peut être plus simple de ne pas tenir compte de cette décomposition des calculs dans les preuves. Ceci revient à considérer que l'évaluation de l'expression est indivisible, autrement dit que les états intermédiaires du calcul sont inobservables.

Exemple

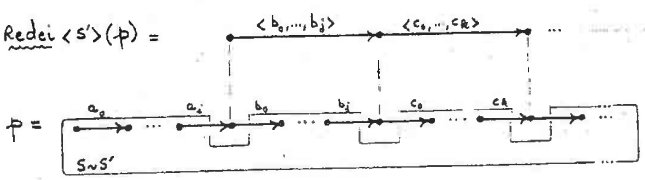
Soit la sémantique $S = \{0, 1, 2, 3\}$, $A = \{R := X\ ;\ ,\ R := R+1\ ;\ ,\ X := R\ ;\}$, $\Sigma = \{0 \xrightarrow{R:=X;} 1 \xrightarrow{R:=R+1;} 2 \xrightarrow{X:=R;} 3\}$. Par élimination de l'ensemble $\{1, 2\}$ des états inobservables, nous obtenons la sémantique $S' = \{0, 3\}$, $A' = \{R := X; R := R+1; X := R\ ;\}$, $\Sigma = \{0 \xrightarrow{R:=X;\ R:=R+1;\ X:=R;} 3\}$.

□

2.5.4.1 Réduction des états inobservables

Etant donnés s, A, s' $(s' \neq 0 \wedge s' \subseteq s)$ et $A' = A^{\#\omega}$, nous notons $\underline{\text{Redei}}\ \langle s' \rangle (p)$ lorsque $(s' \cap \{p_i : i \in |p|\}) \neq 0$, la trace de $\Sigma\langle s', A' \rangle$ dérivée de $p \in \Sigma\langle s, A \rangle$ par réduction des états inobservables de $s \vee s'$. Le schéma suivant donne l'intuition de la définition

$\underline{\text{Redei}}\ \langle s' \rangle (p) =$



Si $p = \langle m, A, a \rangle \in \Sigma\langle s, A \rangle$ alors $\underline{\text{Redei}}\ \langle s' \rangle (p)$ lorsque $(s' \cap A) \neq 0$ est la trace $p' = \langle m', A', a' \rangle \in \Sigma\langle s', A^{\#\omega} \rangle$ définie comme suit :
Si $i \leq m$ alors $\underline{\text{card}}\,(\{A_j \in s : j \in i \wedge A_j \in s'\})$ est le nombre d'états observables dans p de rang strictement inférieur à i (qui peut être ω quand $i = m = \omega$). En particulier $m' = \underline{\text{card}}\,(\{A_j \in s : j \in m \wedge A_j \in s'\})$ est le nombre d'états observables dans p. Nous avons $m' \neq 0$. Si $k \in m'$ alors $r(k) = \sup \{i : i \in m \wedge (\underline{\text{card}}\,(\{A_j \in s : j \in i \wedge A_j \in s'\}) = k)\}$ est le rang (compté à partir de zéro) du $k^{\text{ème}}$ état observable de p. Nous avons $A' \in (m' \to s')$ telle que $\forall k \in m'.\ A'(k) = A(r(k))$. Posons $a' \in ((m'-1) \to A')$ telle que $\forall k \in (m'-1).\ a'(k) = a^{\langle r(k), r(k+1) \rangle}$.

Etant donnés une sémantique $\langle s, A, \Sigma \rangle \in \underline{\text{Sem}}\langle \mathcal{S}, \mathcal{A} \rangle$ et un ensemble non vide d'états (dits observables) $s' \subseteq s$, la sémantique dérivée de $\langle s, A, \Sigma \rangle$ par réduction des états inobservables $s \vee s'$ est $\underline{\text{Redei}}\ \langle s' \rangle (\langle s, A, \Sigma \rangle) = \langle s', A^{\#\omega}, \underline{\text{Redei}}\ \langle s' \rangle [\Sigma] \rangle$.

La relation de réduction dérivée entre systèmes de transition est

$\underline{\text{Redei}}\ \langle s' \rangle (\langle s, A, t, \varepsilon \rangle, \langle s', A', t', \varepsilon' \rangle) = [\underline{\text{Redei}}\ \langle s' \rangle (\langle s, A, \Sigma\langle s, A, t, \varepsilon \rangle \rangle) = \langle s', A', \Sigma\langle s', A', t', \varepsilon' \rangle \rangle]$

Pour étudier les propriétés de cette relation, nous utiliserons la notation suivante :
si $t \in (A \to (s \times s \to \{\#, \#\#\}))$ alors nous étendons t à $((2^s)^3 \to (A^{<\omega} \to (s \times s \to \{\#, \#\#\})))$ par :

$t\,|\,s_d, s_i, s_f\,|_{\langle\rangle}\,(A, A') = [A = A \wedge A' \in s_f]$

$t\,|\,s_d, s_i, s_f\,|_{\langle a_0, \ldots, a_m \rangle}\,(A, A') = [\exists A \in (m+2 \to s).\ (A_0 = A \in s_d \wedge \forall j \in (m+1 \vee 0).\ A_j \in s_i \wedge A_{m+1} = A' \in s_f \wedge \forall j \in (m+1).\ t_{a_j}(A_j, A_{j+1}))]$

c'est-à-dire que nous passons de $A \in s_d$ à $A' \in s_f$ par les actions $a_0, \ldots, a_m$ sur des états intermédiaires dans s_i.

Lemme 2.5.4.1

(1) Si $\langle s, A, t, \varepsilon \rangle \in \underline{\text{Tran}}\langle \mathcal{S}, \mathcal{A} \rangle$ est un système de transition, $s' \subseteq s$ un ensemble non vide d'états et $\varepsilon^\# \in (s' \to \{\#, \#\#\})$, $t^\# \in (A^{\#\omega} \to (s' \times s' \to \{\#, \#\#\}))$ sont définis par :

$\varepsilon^\#(A') = [\exists A \in s, \alpha \in A^{<\omega}.\ \varepsilon(A) \wedge t\,|\,s \vee s', s \vee s', s'\,|_\alpha\,(A, A')]$ (a)

$t^\#_\alpha(A, A') = t\,|\,s', s \vee s', s'\,|_\alpha\,(A, A')$ (b)

alors

$\Sigma\langle s', A^{\#\omega}, t^\#, \varepsilon^\# \rangle \subseteq \underline{\text{Redei}}\ \langle s' \rangle [\Sigma\langle s, A, t, \varepsilon \rangle]$

et donc

$\underline{\text{Redei}}\ (\langle s, A, t, \varepsilon \rangle, \langle s', A', t', \varepsilon' \rangle) \Rightarrow [\langle s', A^{\#\omega}, t^\#, \varepsilon^\# \rangle \subseteq \langle s', A', t', \varepsilon' \rangle]$

(2) Si de plus

$\forall p \in \Sigma\langle s, A, t, \varepsilon \rangle, i \in |p|, \alpha \in A^{\#\omega}.\ t^\#_\alpha(p_i, A') \Rightarrow [\exists j > i.\ j \in |p| \wedge p_j \in s']$ (c)

alors

$\Sigma\langle s', A^{\#\omega}, t^\#, \varepsilon^\# \rangle = \underline{\text{Redei}}\ \langle s' \rangle [\Sigma\langle s, A, t, \varepsilon \rangle]$

et donc

$\underline{\text{Redei}}\ \langle s' \rangle (\langle s, A, t, \varepsilon \rangle, \langle s', A', t', \varepsilon' \rangle) \Rightarrow [\langle s', A', t', \varepsilon' \rangle \equiv \langle s', A^{\#\omega}, t^\#, \varepsilon^\# \rangle]$

Démonstration

(1) Si $q \in \Sigma\langle S', A^{*\omega}, t^{\#}, \varepsilon^{\#}\rangle$ alors nous pouvons construire $p \in \Sigma\langle S, A, t, \varepsilon\rangle$ tel que $q = \underline{\text{Redei}}\langle S'\rangle(p)$. D'après 2.4:1, nous avons $\varepsilon^{\#}(q_0)$ et donc d'après (a), il existe $n \in \omega$, $p_0, \dots, p_n \in S$, $\#_{0}, \dots, \#_{n-1} \in A$ tels que $\forall i \in n.\ p_i \in (S \cup S')$, $p_n = q_0 \in S'$, $\varepsilon(p_0)$ et $\forall i \in n.\ t_{\#_i}(p_i, p_{i+1})$. Posons $r(0) = n$ de sorte que $q_0 = p_{r(0)}$ et $\forall i \in r(0).\ p_i \notin S'$. Si p a été construit jusqu'au rang $r(k)$ tel que $q_k = p_{r(k)}$ et $k \in |q|$ alors d'après 2.4:1 nous avons $t^{\#}_{\underline{\#}_k}(q_k, q_{k+1})$ et donc d'après (b), il existe m avec $m+1 = |\underline{\#}_k|$, $p_{r(k)+1}, \dots, p_{r(k)+m+1} \in S$, $\#_{r(k)} = \underline{\#}_k(0)$, $\#_{r(k)+m} = \underline{\#}_k(m)$ tels que $\forall i \in ((m+1) \setminus 0).\ p_{r(k)+i} \in (S \cup S')$, $p_{r(k)+m+1} = q_{k+1} \in S'$ et pour $i = r(k), \dots, r(k)+m$, nous avons $t_{\#_i}(p_i, p_{i+1})$. Posons $r(k+1) = r(k)+m+1$ de sorte que $q_{k+1} = p_{r(k+1)}$ et $q_k = \#^{\langle r(k), r(k+1)\rangle}$. Si q est infinie alors par cette construction p l'est également et donc $p \in \Sigma\langle S, A, t, \varepsilon\rangle$. Si q est finie de longueur ℓ, poursuivons la construction comme suit : soit $\tau \in \langle S, A, t, \varepsilon'\rangle$ une trace finie ou infinie, avec $\varepsilon'(A) = [A = p_{r(\ell-1)}]$. Posons $p_{r(\ell-1)+i} = \tau_i$ pour $i \in |\tau|$ et $\#_{r(\ell-1)+i} = \tau_{\Rightarrow i}$ pour $i \in |\tau_{\Rightarrow}|$. Observons que $\forall i \in (|\tau| \setminus 0).\ \tau_i \notin S'$ car sinon pour le plus petit $i \in (|\tau| \setminus 0)$ tel que $\tau_i \in S'$ nous aurions $t^{\#}_{\langle \tau_0, \dots, \tau_{i-1}\rangle}(\tau_0, \tau_i)$ en contradiction avec le fait que $q_{\ell-1} = p_{r(\ell-1)} = \tau_0$ n'a pas de successeur puisque q est de longueur ℓ. Il reste à démontrer par récurrence sur k que $r(k) = \sup\{i : i \in |p| \wedge (\mathrm{card}(\{p_j \in S : j \in i \wedge p_j \in S'\}) = k)\}$. Comme $\forall j \in r(0).\ p_j \notin S'$ et $p_{r(0)} \in S'$, c'est vrai pour $k = 0$. Si c'est vrai pour k alors $\mathrm{card}(\{p_j \in S : j \in r(k) \wedge p_j \in S'\}) = k$, $p_{r(k)} \in S'$, $p_{r(k)+1}, \dots, p_{r(k+1)-1} \notin S'$ et $p_{r(k+1)} \in S'$ impliquent $\mathrm{card}(\{p_j \in S : j \in i \wedge p_j \in S'\}) = k+1$ pour $i = r(k)+1, \dots, r(k+1)$.

Observons que $\underline{\text{Redei}}\langle S'\rangle(\langle S, A, t, \varepsilon\rangle, \langle S', A', t', \varepsilon'\rangle)$ est équivalent à $A' = A^{*\omega} \wedge \underline{\text{Redei}}\langle S'\rangle[\Sigma\langle S, A, t, \varepsilon\rangle] = \Sigma\langle S', A', t', \varepsilon'\rangle$, ce qui implique $\Sigma\langle S', A^{*\omega}, t^{\#}, \varepsilon^{\#}\rangle \leq \Sigma\langle S', A', t', \varepsilon'\rangle$.

(2) A la suite de (1), montrons que si $\exists p \in \Sigma\langle S, A, t, \varepsilon\rangle$ tel que $q = \underline{\text{Redei}}\langle S'\rangle(p)$ alors $q \in \Sigma\langle S', A^{*\omega}, t^{\#}, \varepsilon^{\#}\rangle$. Comme $(S' \cap \{p_i : i \in |p|\}) \neq 0$, il existe $j \in |p|$ tel que $p_j \in S'$. $r(0)$ est le plus petit $j \in |p|$ tel que $p_j \in S'$ et nous avons $p_{r(0)} = q_0$ et donc $\varepsilon^{\#}(q_0)$. Si $k \in |q_{\Rightarrow}|$, nous avons $t \in S'$, $S \cup S'$, $S' \wedge \#^{\langle r(k), r(k+1)\rangle}(p_{r(k)}, p_{r(k+1)})$, $q_k = p_{r(k)}$, $q_{k+1} = p_{r(k+1)}$ et $\underline{\#}_k = \#^{\langle r(k), r(k+1)\rangle}$ ce qui implique que $t^{\#}_{\underline{\#}_k}(q_k, q_{k+1})$. D'après 2.4:1, nous avons donc $q \in \Sigma\langle S', A^{*\omega}, t^{\#}, \varepsilon^{\#}\rangle$ si q est infinie. Si q est finie de longueur ℓ, il faut montrer que $q_{\ell-1}$ n'a pas de successeur pour $t^{\#}$. Par l'absurde, supposons qu'il y en ait un, soit λ. Il existe donc $\alpha \in A^{*\omega}$ tel que $t^{\#}_{\alpha}(q_{\ell-1}, \lambda)$. Comme $q_{\ell-1} = p_{r(\ell-1)}$, il existe $j \in |p|$ tel que $r(\ell-1) < j < |p| \wedge p_j \in S'$ à suivre p.

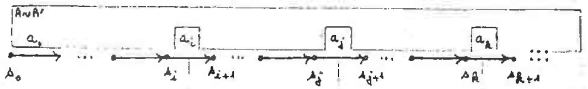
$r(\ell-1) = \sup\{i : i \in |p| \wedge (\mathrm{card}(\{p_j \in S : j \in i \wedge p_j \in S'\}) = \ell-1)\}$ et $p_{r(\ell-1)} \in S'$ d'où $\mathrm{card}(\{p_j \in S : j \in (r(\ell-1)+1) \wedge p_j \in S'\}) = \ell$. Par définition de $q = \underline{\text{Redei}}\langle S'\rangle(p)$ et $|q| = \ell$ nous avons $\ell = \mathrm{card}(\{p_j \in S : j \in |p| \wedge p_j \in S'\})$. Nous en déduisons la contradiction $\forall k.\ (r(\ell-1) < k < |p| \Rightarrow p_k \notin S')$.

Comme $\underline{\text{Redei}}\langle S'\rangle(\langle S, A, t, \varepsilon\rangle, \langle S', A', t', \varepsilon'\rangle) = \Sigma\langle S', A', t', \varepsilon'\rangle$ soit $A' = A^{*\omega} \wedge \underline{\text{Redei}}\langle S'\rangle[\Sigma\langle S, A, t, \varepsilon\rangle] = \Sigma\langle S', A', t', \varepsilon'\rangle$ soit $A' = A^{*\omega} \wedge \Sigma\langle S', A^{*\omega}, t^{\#}, \varepsilon^{\#}\rangle = \Sigma\langle S', A', t', \varepsilon'\rangle$ c'est-à-dire $\langle S', A', t', \varepsilon'\rangle \equiv \langle S', A^{*\omega}, t^{\#}, \varepsilon^{\#}\rangle$.

□

2.5.4.2 Réduction des actions inobservables

De manière analogue, nous pourrons définir la réduction des actions inobservables d'une trace. Etant donnés S, A, $S' = S^{*\omega}$ et A' $(A' \in A \wedge A' \neq 0)$, nous notons $\underline{\text{Redai}}\langle A'\rangle(p)$ lorsque $(A' \cap \{\#_i : i \in |\underline{\#}|\}) \neq 0$ la trace de $\Sigma\langle S', A'\rangle$ dérivée de $p \in \Sigma\langle S, A\rangle$ par réduction des actions inobservables de $A \cup A'$. L'idée se représente informellement comme suit :



et la formalisation est tout à fait similaire à ce qui précède.

2.6 FERMETURES DE SEMANTIQUES

Nous définissons maintenant des relations entre sémantiques au moyen d'opérateurs de fermeture sur le treillis complet $\underline{Sem}\langle\mathcal{O},\mathcal{A}\rangle$ muni de l'inclusion $\subseteq$. Nous étudions, quand elle est intéressante, la relation induite entre systèmes de transition.

Dans une première partie (2.6.1 à 2.6.4) nous nous intéressons à des relations entre sémantiques qui seront ultérieurement utilisées pour justifier certaines méthodes de preuve. Il s'agit de raisonner sur les préfixes des traces (2.6.1), les suffixes des traces (2.6.2), les états et actions accessibles (2.6.3) ou les traces équitables.

Dans une deuxième partie (2.6.5 à 2.6.8) nous introduisons successivement (en 2.6.5) la notion de sémantique fermée par fusion (Pratt [79]), (en 2.6.6) la notion de sémantique réduite par élimination des traces préfixes strictes, (en 2.6.7) la notion de fermeture d'une sémantique par limites (Abrahamson [80]) ce qui permet (en 2.6.8) de donner des conditions nécessaires et suffisantes pour qu'une sémantique $\langle S,A,\Sigma\rangle$ d'une part et la sémantique $\underline{Rtran}(\langle S,A,\Sigma\rangle)$ engendrée par le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle,\varepsilon\langle S,A,\Sigma\rangle\rangle$ engendré par $\langle S,A,\Sigma\rangle$ d'autre part soient $\subseteq$-comparables (cf. théorèmes 2.6.8∨2, 2.6.8∨3) ou égales (cf. théorème 2.6.8∨4). Ces résultats seront utilisés dans le paragraphe suivant pour définir des sémantiques à l'aide de systèmes de transition.

2.6.1 FERMETURE D'UNE SEMANTIQUE PAR PREFIXES

Certaines propriétés de sémantiques comme l'invariance se conservent par fermeture par préfixes.

La relation de préface ou facteur gauche sur $\Sigma^{\leq\omega}\langle S,A\rangle$, définie par $(p \mapsto q \Leftrightarrow \exists i\in|q|.\ p=q^{\leq i})$ est une relation d'ordre réflexive.

La fermeture par préfixes d'une sémantique $\langle S,A,\Sigma\rangle$ est la sémantique $\underline{Pref}(\langle S,A,\Sigma\rangle) = \langle S,A,\{p\in\Sigma^{\leq\omega}\langle S,A\rangle : \exists q\in\Sigma.\ p\mapsto q\}\rangle$

Pref est un opérateur de fermeture supérieure sur $\underline{Sem}\langle\mathcal{O},\mathcal{A}\rangle$ muni de l'inclusion $\subseteq$.

La relation induite sur les systèmes de transition n'est pas intéressante car une sémantique fermée par préfixes ne peut pas être engendrée par un système de transition, sauf si toutes les traces sont réduites à un seul état.

Il est quelquefois beaucoup plus facile de faire des preuves en raisonnant sur les préfixes finis ou traces incomplètes (c'est-à-dire sur des calculs "en cours") plutôt que sur des traces complètes (c'est-à-dire sur des calculs terminés ou infinis). Dans ce cas, nous pouvons raisonner sur la fermeture par préfixes finis de la sémantique:

La fermeture par préfixes finis d'une sémantique $\langle S,A,\Sigma\rangle$ est $\underline{Pref}^{<\omega}(\langle S,A,\Sigma\rangle) = \langle S,A,\{p\in\Sigma^{<\omega}\langle S,A\rangle : \exists q\in\Sigma.\ p\mapsto q\}\rangle$

$\underline{Pref}^{<\omega}$ n'est pas extensif (nous n'avons pas $\langle S,A,\Sigma\rangle \subseteq \underline{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)$ quand Σ contient une trace infinie), toutefois nous avons :

Lemme 2.6.1~1

$\underset{\sim}{Pref}^{<\omega}$ est un opérateur de préfermeture supérieure sur $\underset{\sim}{Sem}\langle \mathcal{S}, \mathcal{A} \rangle$.

De nouveau la relation induite sur les systèmes de transition est sans intérêt.

Nous utiliserons la remarque triviale que l'ensemble des préfixes d'un ensemble de traces est constitué de préfixes finis et des traces infinies :

Lemme 2.6.1~2

$$\underset{\sim}{Pref}(\langle S,A,\Sigma\rangle) = \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle) \cup \langle S,A,\Sigma\cap\Sigma^{\omega}\langle S,A\rangle\rangle$$

2.6.2 FERMETURE D'UNE SEMANTIQUE OU D'UN SYSTEME DE TRANSITION PAR SUFFIXES

Certaines preuves de programmes sont relatives à un état initial qui ne correspond pas forcément au point de départ de l'exécution du programme. Nous raisonnons alors sur une fermeture de la sémantique du programme par suffixes.

La relation de suffixe ou facteur droit sur $\Sigma^{<\omega}\langle S,A\rangle$ est définie par $p \twoheadrightarrow q \Leftrightarrow \exists i \in |q| \cdot p = q^{\geq i}$.

La fermeture par suffixes d'une sémantique $\langle S,A,\Sigma\rangle$ est la sémantique $\underset{\sim}{Suff}(\langle S,A,\Sigma\rangle) = \langle S,A,\{p \in \Sigma^{<\omega}\langle S,A\rangle : \exists q \in \Sigma \cdot p \twoheadrightarrow q\}\rangle$.

$\underset{\sim}{Suff}$ est un opérateur de fermeture supérieure sur $\underset{\sim}{Sem}\langle \mathcal{S}, \mathcal{A}\rangle$ muni de l'inclusion $\subseteq$.

La relation induite sur les systèmes de transition est définie par
$\underset{\sim}{Suff}(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) = [\underset{\sim}{Suff}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \langle S',A',\Sigma\langle S',A',t',\varepsilon'\rangle\rangle]$

Lemme 2.6.2~1

$$\underset{\sim}{Suff}(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) \Leftrightarrow [\langle S',A',t',\varepsilon'\rangle \equiv \langle S,A,t,\varepsilon^*\rangle]$$
avec $\varepsilon^*(\lambda) = [\exists \lambda' \in S. \ \varepsilon(\lambda') \wedge t^*(\lambda',\lambda)]$

en notant t^* la fermeture transitive réflexive de t définie comme suit :

$$t^*(\lambda,\lambda') = \bigcup_{n\geq 0} t^n(\lambda,\lambda') \qquad \text{avec} \qquad t^0(\lambda,\lambda') = [\lambda'=\lambda]$$
$$t^{n+1}(\lambda,\lambda') = [\exists a \in \Gamma, \lambda'' \in S. (t_a(\lambda,\lambda'') \wedge t^n(\lambda'',\lambda'))]$$

Démonstration

Nous avons $\{p : \exists q \in \Sigma\langle S,A,t,\varepsilon\rangle . \ p \rightarrow q\} = \Sigma\langle S,A,t,\varepsilon^*\rangle$ et donc

$\underline{Suff}(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) \Leftrightarrow [\langle S',A',\Sigma\langle S',A',t',\varepsilon'\rangle\rangle = \langle S,A,\Sigma\langle S,A,t,\varepsilon^*\rangle\rangle]$.

□

Suff est un opérateur de fermeture supérieure sur $\underline{Tran}\langle\mathcal{P},\mathcal{A}\rangle/_{\equiv}$ muni de l'inclusion.

2.6.3 FERMETURE D'UNE SEMANTIQUE OU D'UN SYSTEME DE TRANSITION PAR REDUCTION AUX ETATS ET/OU ACTIONS ACCESSIBLES

Une preuve de programme souvent se simplifie en raisonnant non pas sur des états quelconques du programme mais sur l'ensemble de ceux qui sont accessibles au cours d'un calcul (ou sur un sur-ensemble de ceux-ci caractérisé par un invariant). Ceci revient à raisonner sur la réduction de la sémantique du programme aux états (et actions) accessibles.

Etant donnée une sémantique $\langle S,A,\Sigma\rangle$ la réduction aux états et actions accessibles de cette sémantique est $\underline{Redeaa}(\langle S,A,\Sigma\rangle) = \langle \{s \in S : \exists p \in \Sigma, i \in |p| . (p_i = s)\}, \{a \in A : \exists p \in \Sigma, j \in |t_s| . (t_{sj} = a)\}, \Sigma\rangle$.

Observons que Redeaa est un opérateur de fermeture inférieure sur $\underline{Sem}\langle\mathcal{P},\mathcal{A}\rangle$ muni de l'inclusion $\subseteq$.

Nous définissons de même la réduction aux états accessibles Redea et la réduction aux actions accessibles Redaa.

La relation induite sur les systèmes de transition est

$\underline{Redeaa}(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) = [\underline{Redeaa}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \langle S',A',\Sigma\langle S',A',t',\varepsilon'\rangle\rangle]$

Lemme 2.6.3-1

$\underline{Redeaa}(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) \Leftrightarrow$

$[\langle S',A',t',\varepsilon'\rangle \equiv \langle \{s \in S : \exists s' \in S . (\varepsilon(s') \wedge t^*(s',s))\}, \{a \in A : \exists s,s',s'' \in S . (\varepsilon(s) \wedge t^*(s,s') \wedge t_a(s',s''))\}, t, \varepsilon\rangle]$

Remarquons que Redeaa est un cas particulier de Redat. En effet, $\underline{Redeaa}(\langle S,A,t,\varepsilon\rangle, \langle S',A',t',\varepsilon'\rangle) = \underline{Redat}\langle Acc\rangle(\langle S,A,t,\varepsilon\rangle, \langle S,A',t',\varepsilon'\rangle)$ (où $Acc = Acc\langle S,A,t,\varepsilon\rangle$).

2.6.4 FERMETURE D'UNE SEMANTIQUE PAR REDUCTION AUX TRACES EQUITABLES

Pour définir la sémantique de programmes parallèles avec hypothèse d'exécution équitable des processus, Lehman-Pnueli-Stavi [81], nous pouvons spécifier une sémantique non équitable puis éliminer les traces non équitables. Cette réduction aux traces équitables évite d'avoir à spécifier un contrôleur d'exécution (scheduler) particulier de manière explicite dans la sémantique, et laisse ouvertes d'autres implémentations possibles.

Nous écrivons $\underline{Enabled}(a,i,p,\Sigma)$ quand l'action a est activable au point $i \in |p|$ d'une trace p, c'est-à-dire qu'il existe une trace $q \in \Sigma$ ayant même préfixe que p jusqu'en i et a est activée en ce point :

$$\underline{Enabled}(a,i,p,\Sigma) = [i \in |p| \wedge \exists q \in \Sigma.(i \in |q| \wedge q^{\leq i} = p^{\leq i} \wedge q_i = a)]$$

La réduction d'une sémantique aux traces faiblement équitables pour un ensemble α d'actions conserve les traces finies et les traces infinies pour lesquelles aucune action de α n'est, au delà d'un certain point continuellement activable et jamais activée :

$$\underline{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\rangle) =$$
$$\langle S,A,\{p \in \Sigma : (|p|=\omega) \Rightarrow \neg(\exists a \in \alpha, i \in \omega.\ \forall j \geq i.(\underline{Enabled}(a,j,p,\Sigma) \wedge p_j \neq a))\}\rangle$$

$$\underline{Wfair}(\langle S,A,\Sigma\rangle) = \underline{Wfair}\langle A\rangle(\langle S,A,\Sigma\rangle)$$

Exemple

La réduction de la sémantique 2.1.2-3 aux traces faiblement équitables est la sémantique 2.1.2-2. La trace infinie $0 \xrightarrow{a} 0 \ldots 0 \xrightarrow{a} 0 \ldots$ n'est pas faiblement équitable pour $\{a,b\}$ car l'action b n'est jamais activée et toujours activable pour donner une trace $0 \xrightarrow{a} 0 \xrightarrow{b} \ldots$.

□

La réduction d'une sémantique aux traces fortement équitables pour un ensemble α d'actions conserve les traces finies et les traces infinies pour lesquelles aucune action de α n'est, au delà d'un certain point activable infiniment souvent et jamais activée :

$$\underline{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\rangle) =$$
$$\langle S,A,\{p \in \Sigma : (|p|=\omega) \Rightarrow \neg(\exists a \in \alpha, i \in \omega.(\forall j \geq i.\exists k \geq j.\underline{Enabled}(a,k,p,\Sigma)) \wedge (\forall j \geq i.\ p_j \neq a))\}\rangle$$

$$\underline{Sfair}(\langle S,A,\Sigma\rangle) = \underline{Sfair}\langle A\rangle(\langle S,A,\Sigma\rangle)$$

Remarquons que l'équité forte entraîne l'équité faible.

Lemme 2.6.4~1

(1) $\underline{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\rangle) \subseteq \underline{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\rangle) \subseteq \langle S,A,\Sigma\rangle$

(2) $\underline{Wfair}\langle\alpha\rangle$ et $\underline{Sfair}\langle\alpha\rangle$ sont des opérateurs de fermeture inférieure sur $\underline{Sem}\langle\mathcal{O},\mathcal{A}\rangle$ muni de l'inclusion $\subseteq$.

La relation induite sur les systèmes de transition n'a aucun intérêt car en général $\underline{Wfair}\langle\alpha\rangle(\Sigma\langle S,A,t,\varepsilon\rangle)$ ne peut pas être engendré par un système de transition sur S et A.

Pour justifier les méthodes de preuve de propriétés d'invariance de programmes parallèles équitables, nous utiliserons le fait que tout préfixe fini d'une trace engendrée par un système de transition est préfixe d'une trace équitable relativement à tout ensemble fini d'actions et réciproquement :

Lemme 2.6.4~2

Si $\underline{card}(\alpha) < \omega$ alors

- $\underline{Pref}^{<\omega} \circ \underline{Wfair}\langle\alpha\rangle(\langle O,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underline{Pref}^{<\omega}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$
- $\underline{Pref}^{<\omega} \circ \underline{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underline{Pref}^{<\omega}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

Démonstration

Nous avons $\underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) \subseteq \underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ $\subseteq \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ d'après le lemme 2.6.4.v1.1 et le fait que $\underset{\sim}{Pref}^{<\omega}$ est monotone pour $\subseteq$.

Pour montrer que $\underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) \subseteq \underset{\sim}{Pref}^{<\omega} \circ \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ nous considérons un préfixe fini p de longueur m d'une trace de $\Sigma\langle S,A,t,\varepsilon\rangle$ que nous prolongeons en une trace σ de $\Sigma\langle S,A,t,\varepsilon\rangle$ dont nous montrons qu'elle est fortement équitable. Pour la base de la construction, choisissons $\sigma^{<m} = p$ de sorte que σ soit construit jusqu'au point $m-1$. Dans la construction de σ, nous utilisons une file d'attente f qui contient toujours une fois et une seule tout élément de α et qui par hypothèse est donc finie. Notons f_ℓ la valeur de cette file au point ℓ de la construction. Initialement f_{m-1} contient toutes les actions de α dans un ordre quelconque. Supposons que nous ayons construit σ jusqu'au point ℓ et défini f_ℓ. Si σ_ℓ n'a pas de successeur pour t alors la trace σ est finie et donc fortement équitable, ce qui termine la démonstration. Sinon, il existe une action b activable en σ_ℓ. Si aucune des actions activables en σ_ℓ n'appartient à f_ℓ, nous choisissons $f_{\ell+1} = f_\ell$, $\sigma_{\ell} = b$ et $\sigma_{\ell+1}$ un élément quelconque de s tel que $t_{\sigma_\ell}(\sigma_\ell, \sigma_{\ell+1})$ soit vrai. Sinon il existe une action de f_ℓ activable en σ_ℓ. Dans ce cas, nous choisissons σ_ℓ comme étant l'action activable de f_ℓ la plus près de la tête de la file (toutes les actions devant σ_ℓ dans f_ℓ (s'il y en a) n'étant donc pas activables en σ_ℓ). Nous choisissons $\sigma_{\ell+1}$ quelconque tel que $t_{\sigma_\ell}(\sigma_\ell, \sigma_{\ell+1})$ soit vrai et $f_{\ell+1}$ comme étant f_ℓ à la différence que σ_ℓ a été déplacée en queue de la file d'attente. Pour achever cette démonstration, il suffit de montrer que si par cette construction, nous obtenions une trace σ infinie alors elle est fortement équitable. En effet, dans le cas contraire il existerait une action a de α qui est activable infiniment souvent et jamais activée au delà d'un point i et donc au point $j = \sup^+\{m, i\}$ de σ. Pour chaque entier k pour lequel a est activable, σ_k précède a dans f_k. Or ceci n'est possible qu'un nombre fini de fois.

□

2.6.5 FERMETURE D'UNE SEMANTIQUE PAR FUSIONS

En général, les évolutions possibles de l'exécution à partir d'un état d'un programme ne dépendent pas de la manière dont cet état a été atteint. Cette condition s'exprime par le fait que la sémantique du programme est fermée par fusions c'est-à-dire que tout préfixe fini d'une trace (terminé par un état A) peut se prolonger par tout suffixe (commençant par A) d'une trace :

L'extension d'une sémantique $\langle S,A,\Sigma\rangle$ par fusions est $\underset{\sim}{Efus}(\langle S,A,\Sigma\rangle) = \langle S,A,\{p\hat{}q : p\in\Sigma^{<\omega}\langle S,A\rangle \wedge q\in\Sigma^{\leq\omega}\langle S,A\rangle \wedge \exists p',q'\in\Sigma.(p \mapsto p' \wedge q \twoheadrightarrow q')\}\rangle$

Exemple

L'extension de la sémantique $\langle\{0,1\},\{a,b\},\{0\xrightarrow{b}1, 0\xrightarrow{a}0\xrightarrow{b}1\}\rangle$ par fusions est la sémantique 2.1.2-1 c'est-à-dire $\langle\{0,1\},\{a,b\},\{0\xrightarrow{b}1, 0\xrightarrow{a}0\xrightarrow{b}1, 0\xrightarrow{a}0\xrightarrow{a}0\xrightarrow{b}1\}\rangle$.

□

Observons que $\underset{\sim}{Efus}$ est un opérateur monotone et extensif sur $\langle \underset{\sim}{Sem}\langle\mathcal{O},\mathcal{A}\rangle, \subseteq\rangle$, mais n'est pas idempotent. D'où la définition suivante :

La fermeture d'une sémantique $\langle S,A,\Sigma\rangle$ par fusions est $\underset{\sim}{Ffus}(\langle S,A,\Sigma\rangle)$ où $\underset{\sim}{Ffus}$ est le plus petit opérateur de fermeture sur $\underset{\sim}{Sem}\langle\mathcal{O},\mathcal{A}\rangle$ plus grand ou égal à $\underset{\sim}{Efus}$. (plus petit et plus grand étant compris par rapport à l'extension point par point de $\subseteq$).

Exemple

La fermeture par fusions de la sémantique que 2.1.2-1 est la sémantique 2.1.2-2.

□

Lemme 2.6.5∿1

(1) $\underset{\sim}{Ffus} = \bigcup_{n\geq 0} \underset{\sim}{Efus}^n$

(2) $\underset{\sim}{Efus} \circ \underset{\sim}{Ffus} = \underset{\sim}{Ffus}$

Démonstration

Comme $\underset{\sim}{Sem}\langle \mathcal{P}, \mathcal{O}\rangle$ est un treillis complet pour $\leq$ et $\underset{\sim}{Efus}$ est monotone et extensif, le théorème 4.3, son corollaire 4.4-1 et la définition 1.13 dans Cousot - Cousot [79] impliquent que $\underset{\sim}{Ffus}(\langle S,A,\Sigma\rangle)$ est la limite de la séquence $X_0 = \langle S,A,\Sigma\rangle$, $X_\delta = \underset{\sim}{Efus}(X_{\delta-1})$ si δ est un ordinal successeur et $X_\delta = \bigcup_{\alpha<\delta} X_\alpha$ si α est un ordinal limite. Par définition de $\underset{\sim}{Efus}$, X_δ est de forme $\langle S,A,\Sigma_\delta\rangle$ avec $(\alpha\leq\beta)\Rightarrow(\Sigma_\alpha \subseteq \Sigma_\beta)$. Pour montrer que la limite est atteinte pour $\delta=\omega$, il suffit de montrer que $\Sigma_{\omega+1} = \{p\hat{}q : p\in\Sigma^{<\omega}\langle S,A\rangle \wedge q\in\Sigma^{\leq\omega}\langle S,A\rangle \wedge (\exists p',q'\in \bigcup_{i<\omega}\Sigma_i .\ p\twoheadrightarrow p' \wedge q\twoheadrightarrow q')\} \subseteq \Sigma_\omega = \bigcup_{k<\omega}\Sigma_k = \bigcup_{k<\omega}\{p\hat{}q : p\in\Sigma^{<\omega}\langle S,A\rangle \wedge q\in\Sigma^{\leq\omega}\langle S,A\rangle \wedge (\exists p',q'\in \bigcup_{i<k}\Sigma_i .\ p\twoheadrightarrow p' \wedge q\twoheadrightarrow q')\}$. Nous avons bien $(\exists p',q'\in\bigcup_{i<\omega}\Sigma_i)\Rightarrow(\exists i,j\in\omega .\ p'\in\Sigma_i \wedge q'\in\Sigma_j)\Rightarrow(\exists i,j\in\omega .\ p',q'\in\Sigma_{\sup(\{i,j\})})\Rightarrow(\exists k\in\omega .\ p',q'\in\Sigma_k)$ car $\Sigma_i \subseteq \Sigma_{\sup(\{i,j\})}$ et $\Sigma_j \subseteq \Sigma_{\sup(\{i,j\})}$.

□

La relation induite sur les systèmes de transition est l'identité car :

Lemme 2.6.5∿2

$$\underset{\sim}{Ffus}\left(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle\right) = \langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$$

Démonstration

Il suffit de montrer que $\underset{\sim}{Efus}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ et d'appliquer le lemme 2.6.5∿1.1.

□

Nous utiliserons la propriété suivante :

Lemme 2.6.5∿3

(1) $\underset{\sim}{Efus} \circ \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

(2) $\underset{\sim}{Ffus} \circ \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

(3) $\underset{\sim}{Efus} \circ \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

(4) $\underset{\sim}{Ffus} \circ \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$

Démonstration

(1), (3) : Si $p = p\hat{}p'$ et $q = q'\hat{}q$ sont des traces équitables de $\Sigma\langle S,A,t,\varepsilon\rangle$ alors $p\hat{}q$ est une trace de $\Sigma\langle S,A,t,\varepsilon\rangle$ qui est équitable car sinon une action est au delà de p toujours ou infiniment souvent activable et jamais activée dans q donc dans q.

(2) : Par récurrence, nous avons $\underset{\sim}{Efus}^n \circ \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \underset{\sim}{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ et le résultat dérive de 2.6.5∿1.1. Idem pour (4).

□

2.6.6 REDUCTION D'UNE SEMANTIQUE PAR ELIMINATION DES TRACES PREFIXES STRICTS

En général, si l'exécution d'un programme peut se poursuivre à partir d'un certain état, alors elle doit se poursuivre. Cette condition correspond à l'hypothèse habituelle que les calculs progressent à une vitesse non nulle ou encore qu'il n'y a pas de blocages (pannes) dus à un agent extérieur. Cette condition s'exprime par le fait que la sémantique du programme est réduite par élimination des traces préfixes strictes c'est-à-dire qu'il n'existe pas de trace qui soit préfixe strict d'une autre trace.

La réduction d'une sémantique $\langle S, A, \Sigma \rangle$ par élimination des traces préfixes strictes est $\mathrm{Retps}(\langle S, A, \Sigma \rangle) = \langle S, A, \{p \in \Sigma : \forall q \in \Sigma \ (p \to q) \Rightarrow (p = q)\} \rangle$

Exemple

$\mathrm{Retps}(\langle \{0\}, \{a\}, \{0, 0 \xrightarrow{a} 0\} \rangle) = \langle \{0\}, \{a\}, \{0 \xrightarrow{a} 0\} \rangle$

$\mathrm{Retps}(\langle \{0\}, \{a\}, \{0, 0 \xrightarrow{a} 0, 0 \xrightarrow{a} 0 \xrightarrow{a} 0\} \rangle) = \langle \{0\}, \{a\}, \{0 \xrightarrow{a} 0 \xrightarrow{a} 0\} \rangle$

□

L'opérateur Retps est réductif, idempotent mais n'est pas monotone pour $\subseteq$ comme le montre le contre-exemple ci-dessus.

La relation induite sur les systèmes de transition est l'identité car :

Lemme 2.6.6 v1

$\mathrm{Retps}(\langle S, A, \Sigma \langle S, A, t, \epsilon \rangle \rangle) = \langle S, A, \Sigma \langle S, A, t, \epsilon \rangle \rangle$

Observons que si $\langle S, A, \Sigma' \rangle = \mathrm{Retps}(\langle S, A, \Sigma \rangle)$ alors Σ et Σ' ont même ensemble

Lemme 2.6.6 v2

$\mathrm{Retps} \circ \mathrm{Wfair}\langle\alpha\rangle(\langle S, A, \Sigma \langle S, A, t, \epsilon \rangle \rangle) = \mathrm{Wfair}\langle\alpha\rangle(\langle S, A, \Sigma \langle S, A, t, \epsilon \rangle \rangle)$

$\mathrm{Retps} \circ \mathrm{Sfair}\langle\alpha\rangle(\langle S, A, \Sigma \langle S, A, t, \epsilon \rangle \rangle) = \mathrm{Sfair}\langle\alpha\rangle(\langle S, A, \Sigma \langle S, A, t, \epsilon \rangle \rangle)$

2.6.7 FERMETURE D'UNE SEMANTIQUE PAR LIMITES

Certaines sémantiques ont la propriété que certaines traces d'exécution peuvent être suivies pendant un temps fini arbitrairement long mais pas pendant un temps infini.

Exemple

Pour tout $m>0$, la sémantique 2.1.2-2 offre la possibilité d'exécuter m fois l'action a (puis l'action b) mais il est impossible d'exécuter l'action a un nombre infini de fois.

□

Dans le cas contraire, la sémantique est fermée par limites c'est-à-dire que si tous les préfixes finis d'une trace infinie peuvent être suivis par une exécution alors la trace infinie est un calcul légitime :

$$\underline{Flim}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma \cup \{p\in\Sigma^{\omega}\langle S,A\rangle : \forall m\in\omega.\ \exists q\in\Sigma.\ p^{\leq m}\rightarrow q\}\rangle$$

Exemple

La fermeture par limites de la sémantique 2.1.2-2 est la sémantique 2.1.2-3.

□

Lemme 2.6.7-1

$\underline{Flim}$ est un opérateur de fermeture supérieure sur $\langle \underline{Sem}\langle\mathcal{P},\hat{\rightarrow}\rangle, \subseteq\rangle$

Démonstration

$\underline{Flim}$ est extensif et monotone et croissant. Soit $L(\Sigma) = \{p\in\Sigma^{\omega}\langle S,A\rangle : \forall m\in\omega.\ \exists q\in\Sigma.\ p^{\leq m}\rightarrow q\}$. Pour montrer

Si $p\in L(L(\Sigma))$ alors $\forall m\in\omega.\ \exists q\in L(\Sigma).\ p^{\leq m}\rightarrow q$ et donc $\exists q'\in\Sigma.\ p^{\leq m}\rightarrow q'$ et donc $p\in L(\Sigma)$.

□

La relation induite sur les systèmes de transition est l'identité car :

Lemme 2.6.7-2

$$\underline{Flim}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) = \langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$$

Observons que la fermeture par limites d'une sémantique n'introduit pas de nouveaux préfixes finis de traces et par conséquent deux sémantiques ayant mêmes fermetures par limites ont mêmes préfermetures par préfixes finis (la réciproque étant fausse) :

Lemme 2.6.7-3

(1) $\underline{Pref}^{<\omega}\circ\underline{Flim} = \underline{Pref}^{<\omega}$

(2) $[\underline{Flim}(\langle S_1,A_1,\Sigma_1\rangle) = \underline{Flim}(\langle S_2,A_2,\Sigma_2\rangle)] \Rightarrow\!\!\!\!\!\!\not\quad\ [\underline{Pref}^{<\omega}(\langle S_1,A_1,\Sigma_1\rangle) = \underline{Pref}^{<\omega}(\langle S_2,A_2,\Sigma_2\rangle)]$

Démonstration

(1) Comme $\underline{Flim}$ est extensif et $\underline{Pref}^{<\omega}$ monotone, nous avons $\underline{Pref}^{<\omega}(\langle S,A,\Sigma\rangle) \subseteq \underline{Pref}^{<\omega}(\underline{Flim}(\langle S,A,\Sigma\rangle))$. Si $\langle S',A',\Sigma'\rangle = \underline{Pref}^{<\omega}(\underline{Flim}(\langle S,A,\Sigma\rangle))$ et $r\in\Sigma'$ alors r est le préfixe fini d'une trace de Σ ou bien le préfixe fini d'une trace $p\in\Sigma^{<\omega}\langle S,A\rangle$ telle que si $n<\omega$, $\exists q\in\Sigma.\ p^{\leq n}\rightarrow q$. Comme $r = p^{\leq m}$, nous en déduisons que r est encore un préfixe fini d'une trace de Σ.

(2,⇒) Si $\underline{Flim}(\langle S_1,A_1,\Sigma_1\rangle) = \underline{Flim}(\langle S_2,A_2,\Sigma_2\rangle)$ alors $\underline{Pref}^{<\omega}\circ\underline{Flim}(\langle S_1,A_1,\Sigma_1\rangle) = \underline{Pref}^{<\omega}\circ\underline{Flim}(\langle S_2,A_2,\Sigma_2\rangle)$ donc d'après (1) ci-dessus 2.6.7-3.1, nous avons $\underline{Pref}^{<\omega}(\langle S_1,A_1,\Sigma_1\rangle) = \underline{Pref}^{<\omega}(\langle S_2,A_2,\Sigma_2\rangle)$

(2,⇏) Le contre-exemple est $S_1=S_2=\{A\}$, $A_1=A_2=\{a\}$, $\Sigma_1 = \{A\xrightarrow{a}A,\ A\xrightarrow{a}A\xrightarrow{a}A,\ A\xrightarrow{a}A\xrightarrow{a}A\xrightarrow{a}A\}$ $\Sigma_2 = \{A\xrightarrow{a}A\xrightarrow{a}A,\ A\xrightarrow{a}A\xrightarrow{a}A\xrightarrow{a}A\}$

Toutefois deux sémantiques ayant mêmes ensembles de traces finies et mêmes préfermetures par préfixes finis ont même fermeture par limites :

Lemme 2.6.7~4

$$[(\Sigma_1 \cap \Sigma^{<\omega}\langle s_1,A_1\rangle) = (\Sigma_2 \cap \Sigma^{<\omega}\langle s_2,A_2\rangle) \wedge \underline{Pref}^{<\omega}(\langle s_1,A_1,\Sigma_1\rangle) = \underline{Pref}^{<\omega}(\langle s_2,A_2,\Sigma_2\rangle)]$$
$$\Rightarrow [\underline{Flim}(\langle s_1,A_1,\Sigma_1\rangle) = \underline{Flim}(\langle s_2,A_2,\Sigma_2\rangle)]$$

Démonstration

Posons $L(\Sigma) = \{p \in \Sigma^\omega\langle s,A\rangle : \forall m \in \omega.\ \exists q \in \Sigma.\ p^{\leq m} \mapsto q\}$. Nous observons que $\underline{Pref}^{<\omega}(\langle s_1,A_1,\Sigma_1\rangle) = \underline{Pref}^{<\omega}(\langle s_2,A_2,\Sigma_2\rangle)$ entraîne que $s_1 = s_2$, $A_1 = A_2$ et $L(\Sigma_1) = L(\Sigma_2)$. Nous avons aussi $\Sigma \cap \Sigma^\omega\langle s,A\rangle \subseteq L(\Sigma)$. Le lemme dérive alors du fait que $\underline{Flim}(\langle s,A,\Sigma\rangle) = \Sigma \cup L(\Sigma) = (\Sigma \cap \Sigma^{<\omega}\langle s,A\rangle) \cup L(\Sigma)$.

□

Comme conséquence, nous obtenons que la fermeture par limites de la réduction aux traces équitables (pour un nombre fini d'actions) d'une sémantique engendrée par un système de transition est cette sémantique :

Lemme 2.6.7~5

Si $card(\alpha) < \omega$ alors

(1) $\underline{Flim} \circ \underline{W}fair_{\langle\alpha\rangle}(\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle) = \langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle$

(2) $\underline{Flim} \circ \underline{S}fair_{\langle\alpha\rangle}(\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle) = \langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle$

Démonstration

Par définition, $\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle$, $\underline{W}fair_{\langle\alpha\rangle}(\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle)$ et $\underline{S}fair_{\langle\alpha\rangle}(\langle s,A,\Sigma\langle s,A,t,\varepsilon\rangle\rangle)$ sont [illegible] d'appliquer les lemmes 2.6.7~1, 2.6.7~4 et 2.6.7~3.

□

Le résultat suivant portant sur la composition de fermetures d'une sémantique par fusions puis par limites sera utilisé ultérieurement pour caractériser les sémantiques engendrées par un système de transition :

Lemme 2.6.7~6

(1) $\underline{Efus} \circ \underline{Flim} \circ \underline{Ffus} = \underline{Flim} \circ \underline{Ffus}$

(2) $\underline{Ffus} \circ \underline{Flim} \circ \underline{Ffus} = \underline{Flim} \circ \underline{Ffus}$

(3) $\underline{Flim} \circ \underline{Ffus}$ est un opérateur de fermeture supérieure sur $\langle \underline{Sem}\langle\mathcal{O},\mathcal{A},t\rangle, \subseteq\rangle$

Démonstration

(1) Posons $\langle s,A,\Sigma_1\rangle = \underline{Ffus}(\langle s,A,\Sigma\rangle)$, $\langle s,A,\Sigma_2\rangle = \underline{Flim}(\langle s,A,\Sigma_1\rangle) = \langle s,A,\Sigma_1 \cup L(\Sigma_1)\rangle$ où $L(\Sigma) = \{p \in \Sigma^\omega\langle s,A\rangle : \forall m \in \omega.\ \exists q \in \Sigma.\ p^{\leq m} \mapsto q\}$, $\langle s,A,\Sigma_3\rangle = \underline{Efus}(\langle s,A,\Sigma_2\rangle)$. Pour démontrer que $\Sigma_3 = \Sigma_2$, il suffit de démontrer que $\Sigma_3 \subseteq \Sigma_2$ car $\underline{Efus}$ est extensive pour $\subseteq$. Si $p \in \Sigma_3$ alors p est de la forme $r^\wedge q$ avec $R = (r^\wedge r') \in \Sigma_2$ et $Q = (q'^\wedge q) \in \Sigma_2$. Comme $\Sigma_2 = \Sigma_1 \cup L(\Sigma_1)$, quatre cas sont à considérer :

- $R \in \Sigma_1$ et $Q \in \Sigma_1$: d'après le lemme 2.6.5~2 et la définition de Σ_1, nous avons $\underline{Efus}(\langle s,A,\Sigma_1\rangle) = \langle s,A,\Sigma_1\rangle$ et donc $(r^\wedge r') \in \Sigma_1$ et $(q'^\wedge q) \in \Sigma_1$, nous en déduisons $p = r^\wedge q \in \Sigma_1 \subseteq \Sigma_2$.

- $R \in L(\Sigma_1)$ et $Q \in \Sigma_1$: il existe $r'' \in \Sigma^{\leq\omega}\langle s,A\rangle$ telle que $(r^\wedge r'') \in \Sigma_1$, ce qui ramène au cas précédent.

- $R \in \Sigma_1$ et $Q \in L(\Sigma_1)$: soit i $(= |r_{\leq s}|)$ le rang de $p_{|r_{\leq s}|}$ dans p et donc dans R de sorte que $p_i = R_i$. Comme $R \in \Sigma_1$, il existe $L \in \Sigma_1$ telle que $R^{\leq m} \mapsto L$ pour $m \in |R|$. En particulier $\forall m \leq i.\ p^{\leq m} \mapsto L \in \Sigma_1$ car $p^{\leq m} = R^{\leq m}$. Comme $Q \in L(\Sigma_1)$, c'est une trace infinie et donc p l'est également. Par définition de $L(\Sigma_1)$, nous avons $\forall m \in \omega.\ \exists L \in \Sigma_1.\ q^{\leq m} \mapsto L$. Quand m est strictement plus grand que le rang j de q_0 dans Q, nous avons $L = ((q'^\wedge q)^{\leq m-i} \wedge \ell) \in \Sigma_1$ soit $(q'^\wedge q^{\leq m-j-1} \wedge \ell) \in \Sigma_1$, avec $\ell \in \Sigma^{\leq\omega}\langle s,A\rangle$. Comme $\underline{Efus}(\langle s,A,\Sigma_1\rangle) = \langle s,A,\Sigma_1\rangle$, nous en déduisons $(r^\wedge q^{\leq m-j-1} \wedge \ell) \in \Sigma_1$ soit $s^{\leq m+1-j} \wedge \ell) \in \Sigma_1$. En particulier quand $n = (m+1-j) > i$ soit $m > j$, $\exists L \in \Sigma_1.\ p^{\leq n} \mapsto L$ et donc $p \in L(\Sigma_1) \subseteq \Sigma_2$.

- $R \in L(\Sigma_1)$ et $Q \in L(\Sigma_1)$: même raisonnement que précédemment car $R \in L(\Sigma_1)$ implique $\exists i \in \mathbb{N}.\ R^{\leq m}$ [illegible]

(2) D'après (1) et par récurrence $\underline{Efus}^n \circ \underline{Flim} \circ \underline{Efus} = \underline{Flim} \circ \underline{Efus}$ et d'après le lemme 2.6.5-1, nous en déduisons $\underline{Efus} \circ \underline{Flim} \circ \underline{Efus} = \underline{Flim} \circ \underline{Efus}$.

(3) $\underline{Flim}$ et $\underline{Efus}$ étant des opérateurs de fermeture, il reste à montrer l'idempotence qui résulte de (2).

□

2.6.8 RETRACTION D'UNE SEMANTIQUE PAR TRANSITIONS, SEMANTIQUE CLOSE

Dans une preuve de correction d'un programme, nous cherchons souvent à éviter les raisonnements sur les traces d'exécution d'une sémantique $\langle S,A,\Sigma\rangle$ et à les remplacer par des raisonnements sur le système de transition $\langle S,A,t,\varepsilon\rangle$ que cette sémantique engendre. Cette simplification ne se justifie que si nous pouvons montrer qu'une preuve relative à la rétraction par transitions $\underline{Rtran}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ de la sémantique $\langle S,A,\Sigma\rangle$ peut remplacer la preuve relative à $\langle S,A,\Sigma\rangle$. Nous étudions les propriétés de l'opérateur $\underline{Rtran}$ qui nous servirons pour de telles justifications.

Définition 2.6.8:1

La rétraction par transitions d'une sémantique $\langle S,A,\Sigma\rangle$ est la sémantique engendrée par le système de transition engendré par cette sémantique :

$\underline{Rtran} \in (\underline{Sem}\langle\mathcal{S},\mathcal{A}\rangle \to \underline{Sem}\langle\mathcal{S},\mathcal{A}\rangle)$

$\underline{Rtran}(\langle S,A,\Sigma\rangle) = \langle S, A, \Sigma\langle S, A, t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle\rangle$

Exemples

La rétraction par transitions de la sémantique $\langle\{0\},\{a\},\{0 \xrightarrow{a} 0\}\rangle$ est $\langle\{0\},\{a\},\{0 \xrightarrow{a} 0 \ldots 0 \xrightarrow{a} 0 \ldots\}\rangle$. Les sémantiques 2.1.2-1, 2.1.2-2 et 2.1.2-3 ont toutes pour rétraction par transitions la sémantique 2.1.2-3.

□

$\underline{Rtran}$ est une rétraction (monotone et idempotent) sur $\langle \underline{Sem}\langle\mathcal{S},\mathcal{A}\rangle, \subseteq\rangle$ mais n'est pas extensive. Retour sur [illegible] exemple 2-dessus... Toutefois nous justifierons le résultat suivant :

Théorème 2.6.8~1

Pref∘Rtran est une fermeture supérieure sur $\langle Sem\langle \mathcal{O}, \mathcal{A}\rangle, \subseteq\rangle$

Démonstration

Pref et Rtran étant monotones et idempotents, la composition l'est également. Il reste à montrer que si $\langle S,A,\Sigma_1\rangle = Pref \circ Rtran(\langle S,A,\Sigma\rangle)$ alors $\Sigma \subseteq \Sigma_1$. C'est évident car si $p \in \Sigma$, nous avons $\epsilon(p_0)$ et $\forall i \in |p|.\ t_{p_i}(p_i, p_{i+1})$ où $\langle S,A,t,\epsilon\rangle$ est le système de transition engendré par $\langle S,A,\Sigma\rangle$. Par conséquent, p est préfixe d'une trace de $\Sigma\langle S,A,t,\epsilon\rangle$ et donc $p \in \Sigma_1$ par définition de Pref.
□

Une sémantique et sa rétraction par transitions n'étant en général pas comparables (et en particulier, pas $\subseteq$-comparables), le raisonnement sur une sémantique n'est pas équivalent au raisonnement sur le système de transition qu'elle engendre.

Exemple

Nous ne pouvons pas démontrer que toutes les traces des sémantiques 2.1.2-1 ou 2.1.2-2 sont finies en raisonnant sur le système de transition 2.3-1 qu'elles engendrent pour la raison que celui-ci engendre la trace infinie $0 \xrightarrow{a} 0 \ldots 0 \xrightarrow{a} 0 \ldots$.
□

Nous pouvons caractériser les cas où une sémantique et sa rétraction par transitions sont $\subseteq$-comparables comme suit :

Théorème 2.6.8~2

$$[\langle S,A,\Sigma\rangle \subseteq Rtran(\langle S,A,\Sigma\rangle)] \Leftrightarrow [\langle S,A,\Sigma\rangle = Retps(\langle S,A,\Sigma\rangle) \wedge \langle S,A,\Sigma\rangle = Efus(\langle S,A,\Sigma\rangle)]$$

Démonstration

($\Rightarrow$) Par l'absurde supposons $\langle S,A,\Sigma\rangle \subseteq Rtran(\langle S,A,\Sigma\rangle)$ et $Retps(\langle S,A,\Sigma\rangle) \neq \langle S,A,\Sigma\rangle$. Comme Retps est réductif, $\exists p,q \in \Sigma.\ p \mapsto q \wedge p \neq q$. Si p était une trace infinie, $p \mapsto q$ entraînerait $p=q$, donc p est une trace finie. Comme $\langle S,A,\Sigma\rangle \subseteq Rtran(\langle S,A,\Sigma\rangle)$, p est une trace engendrée par le système de transition $\langle S,A,t,\epsilon\rangle$ engendré par $\langle S,A,\Sigma\rangle$ et donc $p_{|p|}$ est un état de blocage pour t. Par ailleurs, comme $p \mapsto q$ et $p \neq q$, nous avons $(|p|+1) \in |q|$ et donc par définition de t il vient $t_{q_{|p|}}(q_{|p|}, q_{|p|+1})$, en contradiction avec $q_{|p|} = p_{|p|}$ d'après $p \mapsto q$.

Posons $\langle S,A,\Sigma_e\rangle = Efus(\langle S,A,\Sigma\rangle)$ et $\langle S,A,\Sigma_t\rangle = Rtran(\langle S,A,\Sigma\rangle)$. Par l'absurde supposons $\langle S,A,\Sigma\rangle \subseteq Rtran(\langle S,A,\Sigma\rangle)$ et $Efus(\langle S,A,\Sigma\rangle) \neq \langle S,A,\Sigma\rangle$. Comme Efus est extensif, $\exists P \in \Sigma_e.\ P \notin \Sigma$ qui implique $\exists P \in \Sigma_e.\ P \notin \Sigma_t$ puisque $\langle S,A,\Sigma\rangle \subseteq Rtran(\langle S,A,\Sigma\rangle)$. Nous avons donc $[\exists p \in \Sigma^{<\omega}\langle S,A\rangle, q \in \Sigma^{\leq\omega}\langle S,A\rangle, p',q' \in \Sigma.\ (p \mapsto p' \wedge q \mapsto q' \wedge P = p\hat{}q)] \wedge P \notin \Sigma_t$ qui implique (puisque $\Sigma \subseteq \Sigma_t$ et $q \mapsto q'$) $[\exists p,q'' \in \Sigma^{<\omega}\langle S,A\rangle, q \in \Sigma^{\leq\omega}\langle S,A\rangle, p',q' \in \Sigma_t.\ (p \mapsto p' \wedge q' = q''\hat{}q' \wedge P = p\hat{}q)] \wedge P \notin \Sigma_t$. Si $\langle S,A,t,\epsilon\rangle$ est le système de transition engendré par $\langle S,A,\Sigma\rangle$, nous obtenons $[\exists p,q'' \in \Sigma^{<\omega}\langle S,A\rangle, q \in \Sigma^{\leq\omega}\langle S,A\rangle, p',q' \in \Sigma_t.\ (\epsilon(p'_0) \wedge \forall i \in |p|.\ t_{p'_i}(p'_i, p'_{i+1}) \wedge \forall i \in |q|.\ t_{q'_{(|p|+i)}}(q'_{(|p|+i)}, q'_{(|p|+i+1)})] \wedge P \notin \Sigma_t$ qui implique $[\epsilon(P_0) \wedge \forall i \in |P|.\ t_{P_i}(P_i, P_{i+1})] \wedge P \notin \Sigma_t$ qui est faux.

($\Leftarrow$) Soit $\langle S,A,\Sigma_t\rangle = Rtran(\langle S,A,\Sigma\rangle)$. Par l'absurde, supposons $\langle S,A,\Sigma\rangle = Efus(\langle S,A,\Sigma\rangle)$, $\langle S,A,\Sigma\rangle = Retps(\langle S,A,\Sigma\rangle)$ et $\exists P \in \Sigma.\ P \notin \Sigma_t$. Par définition de Σ_t et le théorème 2.6.8~1, P est préfixe strict d'une trace φ de Σ_t. P est fini sinon nous aurions $\varphi = P$. $\varphi \notin \Sigma$ car $\langle S,A,\Sigma\rangle = Retps(\langle S,A,\Sigma\rangle)$. Ceci implique puisque $Efus(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle$, $\neg[\exists p \in \Sigma^{<\omega}\langle S,A\rangle, q \in \Sigma^{\leq\omega}\langle S,A\rangle, p',q' \in \Sigma.\ (p \mapsto p' \wedge q \mapsto q' \wedge \varphi = (p\hat{}q) \in \Sigma)]$ ainsi que $\neg[\exists p = P \in \Sigma^{<\omega}\langle S,A\rangle, q \in \Sigma^{\leq\omega}\langle S,A\rangle, P,q' \in \Sigma.\ (P \mapsto P \wedge q \mapsto q' \wedge \varphi = (P\hat{}q) \in \Sigma)] \Rightarrow \neg[\exists q \in \Sigma^{<\omega}\langle S,A\rangle, q' \in \Sigma.\ (q \mapsto q' \wedge \varphi = (P\hat{}q) \in \Sigma)] \Rightarrow [\forall q \in \Sigma^{<\omega}\langle S,A\rangle, q' \in \Sigma.\ ((q \mapsto q' \wedge \varphi = P\hat{}q) \Rightarrow \varphi \notin \Sigma)]$ qui est faux car $\langle S,A,\Sigma\rangle = Efus(\langle S,A,\Sigma\rangle)$.
□

Lemme 2.6.8 ∾3

$$[\ \underline{\mathrm{Rtran}}(\langle S,A,\Sigma\rangle) \subseteq \langle S,A,\Sigma\rangle\ ]$$
$$\Leftrightarrow$$
$$[\ \underline{\mathrm{Retps}} \circ \underline{\mathrm{Efus}}\ (\langle S,A,\Sigma\rangle) \subseteq \langle S,A,\Sigma\rangle \quad \wedge \quad \underline{\mathrm{Flim}} \circ \underline{\mathrm{Efus}}(\langle S,A,\Sigma\rangle) = \underline{\mathrm{Efus}}(\langle S,A,\Sigma\rangle)\ ]$$

Démonstration

($\Rightarrow$) Posons $\langle S,A,\Sigma_t\rangle = \underline{\mathrm{Rtran}}$, $\langle S,A,\Sigma_e\rangle = \underline{\mathrm{Efus}}(\langle S,A,\Sigma\rangle)$ et supposons $\Sigma_t \subseteq \Sigma$. Nous avons $\Sigma \subseteq \Sigma_e$ car $\underline{\mathrm{Efus}}$ est extensif.

Une trace de la sémantique $\underline{\mathrm{Retps}} \circ \underline{\mathrm{Efus}}(\langle S,A,\Sigma\rangle)$ est une trace de la forme $(p^\wedge q) \in \Sigma_e$ avec $p \mapsto p' \in \Sigma$ et $q' \mapsto q \in \Sigma$ et qui n'est pas préfixe strict dans Σ_e. Par définition de $\underline{\mathrm{Rtran}}$, nous observons que $p^\wedge q$ est préfixe d'une trace $r \in \Sigma_t \subseteq \Sigma \subseteq \Sigma_e$. Comme $p^\wedge q$ n'est pas préfixe strict dans Σ_e, nous avons $(p^\wedge q) = r$ et par conséquent $(p^\wedge q) \in \Sigma$.

Pour tout $p \in \Sigma^m\langle S,A\rangle$, nous avons $\forall m \in \omega.\ \exists q \in \Sigma_e.\ p^{\varepsilon m} \mapsto q$ qui entraine $p \in \Sigma_e$ et donc $\underline{\mathrm{Flim}}(\langle S,A,\Sigma_e\rangle) = \langle S,A,\Sigma_e\rangle$.

($\Leftarrow$) Pour la réciproque, soient $\langle S,A,\Sigma_e\rangle = \underline{\mathrm{Efus}}(\langle S,A,\Sigma\rangle)$, $\langle S,A,\Sigma_r\rangle = \underline{\mathrm{Retps}}(\langle S,A,\Sigma_e\rangle)$ et $\langle S,A,t,\varepsilon\rangle$ le système de transition engendré par la sémantique $\langle S,A,\Sigma\rangle$.

Montrons d'abord que pour tous $p \in \Sigma\langle S,A,t,\varepsilon\rangle$, $i \in |p|$, $q \in \Sigma_e$. $p^{\varepsilon i} \mapsto q$, nous avons $\exists q' \in \Sigma_r.\ p^{\varepsilon i} \mapsto q'$. Posons $q^0 = q$. Si la trace q^0 de Σ_e est préfixe strict de Σ_e, elle se prolonge en une trace q^1 de Σ_e telle que $p^{\varepsilon i} \mapsto q^1$, qui à nouveau si elle est préfixe strict, se prolonge en une trace q^2 de Σ_e telle que $p^{\varepsilon i} \mapsto q^2$. Si cette construction s'arrête après $n+1$ pas, nous obtenons une trace q^n de Σ_e telle que $p^{\varepsilon i} \mapsto q^n$, qui n'est pas préfixe strict et qui est donc élément de Σ_r. Dans ce cas nous choisissons $q' = q^n$. Si nous obtenons une suite infinie q^k, $j \geq 0$ de Σ_e telles que $p^{\varepsilon i} \mapsto q^j$ et $(j<j') \Rightarrow (q^j)^{\varepsilon i} = (q^{j'})^{\varepsilon i} \wedge q^j \neq q^{j'}$. Comme $\underline{\mathrm{Flim}}(\langle S,A,\Sigma_e\rangle) = \langle S,A,\Sigma_e\rangle$ la trace limite q' définie par $q'^{\varepsilon i} = p^{\varepsilon i}$, $a'_j = a^j_j$ pour $j \geq i$ et $a'_j = q^0_j$ pour $j < i$ appartient à Σ_e. Enfin [illegible]

Montrons maintenant que pour tout $p \in \Sigma\langle S,A,t,\varepsilon\rangle$, $i \in |p|$, nous avons

Si $i=0$ alors $p \in \Sigma\langle S,A,t,\varepsilon\rangle$ implique $\varepsilon(p_0)$ et donc par définition de ε, $\exists r \in \Sigma.\ r_0 = p_0$ de sorte que $\exists r \in \Sigma.\ p^{\varepsilon 0} \mapsto r$. Comme $\Sigma \subseteq \Sigma_e$, nous déduisons du lemme ci-dessus l'existence de $q^0 \in \Sigma_r$ tel que $p^{\varepsilon 0} \mapsto q^0$.

Supposons par hypothèse d'induction pour $i \in |p|$ et $i \neq 0$ l'existence de $q^{i-1} \in \Sigma_r$ tel que $p^{\varepsilon i-1} \mapsto q^{i-1}$. Par définition de t, $\exists q' \in \Sigma, j \in |q'|.\ p_{i-1} = q'_{j-1} \wedge p_{i-1} = q'_{j-1} \wedge p_i = q'_j$. Nous en déduisons que $(p^{\varepsilon i} \wedge q'^{\geq j}) \in \Sigma_e$ et donc l'existence de $q^i \in \Sigma_r$ tel que $p^{\varepsilon i} \mapsto q^i$.

Si $p \in \Sigma\langle S,A,t,\varepsilon\rangle$ est finie alors nous avons montré qu'il existe $q^i \in \Sigma_r$ tel que $i = (|p|-1)$ et $p^{\varepsilon i} \mapsto q^i$. Comme $p \in \Sigma\langle S,A,t,\varepsilon\rangle$, p_i est un état de blocage et donc $q^i = p^{\varepsilon i}$ car sinon nous aurions $(p^{\varepsilon i} \wedge q^{\geq i}) \in \Sigma$ et donc $t_{q_i}(p_i, q^i_{i+1})$ puisque $\langle S,A,t,\varepsilon\rangle$ est engendré par $\langle S,A,\Sigma\rangle$. Nous en déduisons $p = p^{\varepsilon i} = q^i \in \Sigma$. Sinon p est infinie et $\underline{\mathrm{Flim}}(\langle S,A,\Sigma_e\rangle) = \langle S,A,\Sigma_e\rangle$ de sorte que $p \in \Sigma_e$ car $\forall i \in \omega.\ \exists q^i \in \Sigma_r \subseteq \Sigma_e$. Comme p est infinie et $p \in \Sigma_e$, nous en déduisons $p \in \Sigma_r \subseteq \Sigma$.

□

Le cas particulier où une sémantique est rétractée par transitions ou close est important car alors $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ engendre exactement Σ et tout raisonnement sur les traces Σ du programme peut se ramener à un raisonnement sur le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ (au prix de manière triviale en engendrant Σ explicitement à partir de ce système de transition). Ceci conduit à

Définition 2.6.8 : 3

$$[\ \langle S,A,\Sigma\rangle \text{ est } \underline{\mathrm{close}}\ ] \Leftrightarrow [\ \underline{\mathrm{Rtran}}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle\ ]$$

Nous pouvons caractériser les sémantiques closes par les faits que

(α) le comportement futur de l'exécution dépend seulement de l'état courant qui a été atteint et non de la façon dont il a été atteint,

(β) le blocage de l'exécution en un état ne dépend que de cet état,

(δ) les limites des comportements finis sont des comportements infinis acceptables :

Théorème 2.6.8∿4

[$\langle S,A,\Sigma\rangle$ est close]

$\Leftrightarrow$

[$\underline{Efus}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle \wedge \underline{Retps}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle \wedge \underline{Flim}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle$]

Démonstration

($\Rightarrow$) Lemmes 2.6.5∿2, 2.6.6∿1 et 2.6.7∿2. ($\Leftarrow$) Lemmes 2.6.8∿2 et 2.6.8∿3.

□

(Emerson[80] a obtenu un résultat similaire avec une notion différente de sémantique utilisant des traces incomplètes). Essayons maintenant de caractériser l'opérateur Rtran

Lemme 2.6.8∿5

[$\underline{Retps}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle \wedge \underline{Efus}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle$] $\Rightarrow$ [$\underline{Rtran}(\langle S,A,\Sigma\rangle)=\underline{Flim}(\langle S,A,\Sigma\rangle)$]

Démonstration

Si $\underline{Retps}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle$ et $\underline{Efus}(\langle S,A,\Sigma\rangle)=\langle S,A,\Sigma\rangle$ alors d'après 2.6.8∿2, nous avons $\langle S,A,\Sigma\rangle \subseteq \underline{Rtran}(\langle S,A,\Sigma\rangle)$ et donc par monotonie $\underline{Flim}(\langle S,A,\Sigma\rangle) \subseteq \underline{Flim} \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ = $\underline{Flim} \circ \underline{Rtran}(\langle S,A,\Sigma\rangle) = \underline{Rtran}(\langle S,A,\Sigma\rangle)$ d'après [illegible] 2.6.8∿[illegible] donc $\underline{Flim}(\langle S,A,\Sigma\rangle) \subseteq \underline{Rtran}(\langle S,A,\Sigma\rangle)$

$\underline{Flim} \circ \underline{Efus} \circ \underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$ est égal à $\underline{Flim} \circ \underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$ d'après le lemme 2.6.7∿6.1 soit $\underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$ car d'après 2.6.7∿1, Flim est idempotent, soit enfin $\underline{Efus} \circ \underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$ en appliquant à nouveau 2.6.7∿6.1. D'autre part, $\underline{Retps} \circ \underline{Efus} \circ \underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$ = $\underline{Retps} \circ \underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle) \subseteq \underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$ d'après 2.6.7∿6.1 et le fait que Retps soit réductif. D'après le lemme 2.6.8∿3, nous en déduisons que $\underline{Rtran}(\underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)) \subseteq \underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$. Comme Flim et Efus sont $\subseteq$-extensives et Rtran $\subseteq$-monotone, nous avons $\underline{Rtran}(\langle S,A,\Sigma\rangle) \subseteq \underline{Rtran}(\underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle))$ et donc par transitivité $\underline{Rtran}(\langle S,A,\Sigma\rangle) \subseteq \underline{Flim} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$.

□

La rétraction par transitions de la sémantique équitable (pour un nombre fini d'actions) engendrée par un système de transition est exactement la sémantique engendrée par ce système de transition :

Théorème 2.6.8∿6

Si card(α) $<\omega$ alors

(1) $\underline{Rtran} \circ \underline{Wfair}\langle\alpha\rangle \circ \underline{Rtran} = \underline{Rtran}$

(2) $\underline{Rtran} \circ \underline{Sfair}\langle\alpha\rangle \circ \underline{Rtran} = \underline{Rtran}$

Démonstration

Nous avons $\underline{Retps} \circ \underline{Wfair}\langle\alpha\rangle \circ \underline{Rtran}(\langle S,A,\Sigma\rangle) = \underline{Wfair}\langle\alpha\rangle \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ d'après 2.6.6∿2 et $\underline{Efus} \circ \underline{Wfair}\langle\alpha\rangle \circ \underline{Rtran}(\langle S,A,\Sigma\rangle) = \underline{Wfair}\langle\alpha\rangle \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ d'après 2.6.5∿3.1 donc d'après 2.6.8∿5, $\underline{Rtran} \circ \underline{WDfair}\langle\alpha\rangle \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ est égal à $\underline{Flim} \circ \underline{Ffus} \circ \underline{Wfair}\langle\alpha\rangle \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ et donc à $\underline{Flim} \circ \underline{Wfair}\langle\alpha\rangle \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ d'après le lemme 2.6.5∿2.2 soit $\underline{Rtran} \circ \underline{Efus}(\langle S,A,\Sigma\rangle)$ d'après le lemme 2.6.7∿6.1. La preuve est similaire pour 1.

Si une sémantique est fermée par fusions, elle a même préfixes finis que sa rétraction par transitions :

Lemme 2.6.8n7

$$[\underline{Efus}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle] \Leftrightarrow [\underline{Pref}^{<\omega} \circ \underline{Rtran}(\langle S,A,\Sigma\rangle) = \underline{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)]$$

Démonstration

($\leq$) Posons $\langle S,A,\Sigma_1\rangle = \underline{Rtran}(\langle S,A,\Sigma\rangle)$. Soient $p \in \Sigma_1$, $i \in |p|$ de sorte que $p^{\leq i}$ est un préfixe fini de p. Montrons qu'il est aussi préfixe fini d'une trace de Σ. $p^{\leq 0} = p_0$ est préfixe fini d'une trace de Σ car par définition de Σ_1, nous avons $\varepsilon(p_0)$ qui implique $\exists p' \in \Sigma.\ p'_0 = p_0$. Supposons maintenant que $p^{\leq i'}$, $i' \in i$ est un préfixe fini d'une trace de Σ. Donc il existe une trace $q \in \Sigma.\ q^{\leq i'} = p^{\leq i'}$. D'autre part, p étant une trace de Σ_1, elle est engendré par le système de transition $\langle S,A,t,\varepsilon\rangle$ engendré par $\langle S,A,\Sigma\rangle_1$ et nous avons $t_{\sharp i'}(p_{i'}, p_{i'+1})$ qui implique d'après 2.3.1 que $\exists \tau \in \Sigma,\ j \in |\tau| .\ (\tau_j = p_{i'} \wedge \tau_{j+1} = p_{i'+1})$. $\langle S,A,\Sigma\rangle$ étant fermée par fusions, la trace $p^{\leq i'} \xrightarrow{\sharp i'} p_{i'+1} = \tau_{j+1} \xrightarrow{\tau_{j+1}} \tau^{j+1}$ est aussi une trace de Σ et $p^{\leq i'+1}$ en est un préfixe fini. Finalement $p^{\leq i}$ est préfixe fini d'une trace de Σ.

($\geq$) $\langle S,A,\Sigma\rangle \leq \underline{Pref} \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ d'après 2.6.8n1. $\underline{Pref}^{<\omega}$ étant monotone, nous avons $\underline{Pref}^{<\omega}(\langle S,A,\Sigma\rangle) \leq \underline{Pref}^{<\omega} \circ \underline{Pref} \circ \underline{Rtran}(\langle S,A,\Sigma\rangle) = \underline{Pref}^{<\omega} \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$.

□

2.7 SPECIFICATION D'UNE SEMANTIQUE A L'AIDE D'UN SYSTEME DE TRANSITION

Comme nous chercherons à ramener les preuves de programmes relatives à une sémantique $\langle S,A,\Sigma\rangle$ à des preuves relatives au système de transition $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ qu'elle engendre, il est naturel de chercher à spécifier la sémantique $\langle S,A,\Sigma\rangle$ à l'aide d'un système de transition $\langle S,A,t,\varepsilon\rangle$ aussi proche que possible de $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$.

Dans le cas d'une sémantique close (cf. 2.7.1) ces deux systèmes de transition sont équivalents.

Dans le cas d'une sémantique $\langle S,A,\Sigma\rangle$ non close (cf. 2.7.2) les traces de Σ sont préfixes des traces engendrées par $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ de sorte que cette sémantique $\langle S,A,\Sigma\rangle$ peut être spécifiée par un système de transition et une condition sur les préfixes des traces qu'il engendre (cf. 2.7.1.1). Nous montrons également que cette sémantique non close $\langle S,A,\Sigma\rangle$ est engendrée à une concordance près par un système de transition $\langle S^*,A^*,t^*,\varepsilon^*\rangle$ qui induit un contrôleur surveillant l'exécution des programmes (cf. 2.7.1.2). Nous étudions ensuite (cf 2.7.3) le cas particulier important d'une sémantique non close réduite par élimination des traces préfixes strictes et fermée par fusions. Dans ce cas Σ est inclus dans l'ensemble des traces engendrées par $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ de sorte que la sémantique $\langle S,A,\Sigma\rangle$ peut alors être spécifiée par un système de transition et une condition sur les traces qu'il engendre (cf. 2.7.3.1) ou bien comme précédemment par concordance avec une sémantique close (cf. 2.7.3.2).

2.7.1 SPECIFICATION D'UNE SEMANTIQUE CLOSE A L'AIDE DU SYSTEME DE TRANSITION QUI L'ENGENDRE

Les sémantiques closes sont caractérisées par le théorème 2.6.8v4 et peuvent d'après la définition 2.6.8:2 être définies par le système de transition qui les engendre exactement.

Exemple 2.7.1-1

(1) La sémantique 2.1.2-3 est engendrée par l'automate 2.3-1.

(2) La sémantique des programmes séquentiels (de style programmes PASCAL) est réduite par élimination des traces préfixes strictes et fermée par fusions et limites et peut donc d'après le théorème 2.6.8v4 être définie par un système de transition.

(3) Il en va de même pour les programmes asynchrones pour lesquels aucune hypothèse d'équité à l'exécution n'est faite. Ceci correspond aux hypothèses que chaque processus est exécuté par un processeur qui calcule à une vitesse indéterminée qui peut être nulle (quand le processeur tombe en panne) et que ces processeurs ne peuvent pas tous rester en panne indéfiniment. De ce fait, si l'exécution peut globalement progresser alors elle progressera fatalement (grâce à un processus exécuté sur un processeur qui n'est pas indéfiniment en panne). Formellement ceci s'exprime par le fait que la sémantique est réduite par élimination des traces préfixes strictes

De plus, comme les vitesses relatives des processeurs sont inconnues (parce que les pannes sont imprévisibles ou bien parce que les processeurs calculent à des vitesses différentes sans être asynchronisés), l'état suivant un état donné ne dépend pas de la façon dont celui-ci a été atteint. Formellement ceci s'exprime par le fait que la sémantique est fermée par [illegible]

Ensuite, [illegible] peut également être exprimée [illegible]

2.7.2 SPECIFICATION D'UNE SEMANTIQUE NON CLOSE

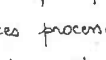
Dans le cas général, deux méthodes peuvent être utilisées pour spécifier une sémantique à l'aide d'un système de transition :

2.7.2.1 Spécification par un système de transition et une condition sur les préfixes des traces qu'il engendre

D'après le théorème 2.6.8v1, nous avons toujours $\langle S,A,\Sigma\rangle \subseteq \underline{Pref} \circ \underline{Rtrans}(\langle S,A,\Sigma\rangle)$. Il est donc toujours possible de spécifier la sémantique $\langle S,A,\Sigma\rangle$ par le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ qu'elle engendre et une condition sur les préfixes des traces engendrées par ce système de transition pour éliminer les préfixes parasites.

Exemple

La sémantique $S=\{0\}$, $A=\{a\}$, $\Sigma=\{0 \xrightarrow{a} 0,\ 0 \xrightarrow{a} 0 \xrightarrow{a} 0\}$ peut être spécifiée par l'automate $\langle S,A,t,\varepsilon\rangle$ schématisé par :



et la restriction aux préfixes de longueur deux ou trois des traces qu'il engendre :

$\Sigma = \{p : \exists q \in \Sigma\langle S,A,t,\varepsilon\rangle . (p \leftarrow q \ \wedge\ 2 \leq |p| \leq 3)\}$ □

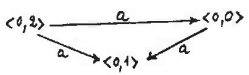
2.7.2.2 Spécification par concordance avec une sémantique close

Pour faire des preuves sur une sémantique non close $\langle S,A,\Sigma\rangle$, il n'est [illegible] sur le système de transition qu'elle engendre [illegible] il est toujours possible de trouver une sémantique close $\langle S^\#,A^\#,\varepsilon^\#\rangle$ qui engendre avec la sémantique non close $\langle S,A,\Sigma\rangle$ une relation entre états

et actions près. Par conséquent les preuves relatives à la sémantique mon close $\langle S,A,\Sigma\rangle$ peuvent se faire, à une relation entre états et actions près, sur le système de transition $\langle S^\#,A^\#,t^\#,\varepsilon^\#\rangle$ engendré par cette sémantique close $\langle S^\#,A^\#,\Sigma^\#\rangle$. Il est donc toujours possible de raisonner sur des systèmes de transition.

Exemple

La sémantique $\langle\{0\},\{a\},\{0\xrightarrow{a}0,\ 0\xrightarrow{a}0\xrightarrow{a}0\}\rangle$ peut être définie par la sémantique engendrée par le système de transition représenté par l'automate



à la concordance r_S entre états et r_A entre actions près définies par :

$r_S(\langle A,m\rangle,s') = [s=s']$

$r_A(a,a') = [a=a']$

□

Montrons qu'il est toujours possible de se restreindre au cas simple où la concordance entre sémantiques est définie à une fonction $\rho^\#\in(S^\#\to S)$ entre états près, (cf. 2.5.3.1) :

Théorème 2.7.2.2 v1

Pour toute sémantique $\langle S,A,\Sigma\rangle$, il existe une sémantique close $\langle S^\#,A^\#,\Sigma^\#\rangle$ et une fonction $\rho^\#\in(S^\#\to S)$ entre états telle que $\langle S,A,\Sigma\rangle$ dérive de $\langle S^\#,A^\#,\Sigma^\#\rangle$ à $\rho^\#$ près, c'est-à-dire :

$\simeq\langle\rho^\#\rangle(\langle S^\#,A^\#,\Sigma^\#\rangle) = \langle S,A,\Sigma\rangle$

Démonstration

$\langle S^\#,A^\#,\Sigma^\#\rangle$ peut toujours être choisie comme la sémantique engendrée par le système de transition $\langle S^\#,A,t^\#,\varepsilon^\#\rangle$ défini par :

$S^\# = \Sigma\times\omega$

$t^\#_a(\langle p,i\rangle,\langle p',i'\rangle) = [(i+1)\in|p| \wedge a=p_i \wedge p'=p \wedge i'=i+1]$

$\varepsilon^\#(\langle p,i\rangle) = [i=0]$

avec

$\rho^\#(\langle p,i\rangle) = p_i$

□

En général, nous nous intéressons à des programmes exécutables par des machines et par conséquent la sémantique $\langle S^\#,A^\#,\Sigma^\#\rangle$ dérive de Rtran$(\langle S,A,\Sigma\rangle)$ par inclusion d'un contrôleur (scheduler) surveillant l'exécution des programmes.

Observons en effet, que toute sémantique $\langle S,A,\Sigma\rangle$ peut être définie par un système de transition $\langle S,A,t,\varepsilon\rangle$ et un contrôleur $\langle S^c,A,t^c,\varepsilon^c\rangle$ de sorte que $\langle S,A,\Sigma\rangle$ concorde avec la sémantique engendrée par $\langle S^\#,A,t^\#,\varepsilon^\#\rangle$ (où $S^\# = S\times S^c$, $t^\#_a(\langle s,s^c\rangle,\langle s',s^{c\prime}\rangle) = [t_a(s,s') \wedge t^c_a(s^c,s^{c\prime})]$ et $\varepsilon^\#(\langle s,s^c\rangle) = [\varepsilon(s) \wedge \varepsilon^c(s^c)]$) à une correspondance $\rho^\#\in(S^\#\to S)$ près telle que $\rho^\#(\langle s,s^c\rangle) = s$. L'objet de l'agent extérieur $\langle S^c,A,t^c,\varepsilon^c\rangle$ est de contrôler l'initialisation et les transitions du système $\langle S,A,t,\varepsilon\rangle$. En pratique, on ne s'intéresse qu'au cas où ces fonctions et relations $t,t^c,\varepsilon,\varepsilon^c$ sont calculables.

2.7.3 SPECIFICATION D'UNE SEMANTIQUE NON CLOSE REDUITE PAR ELIMINATION DES TRACES PREFIXES STRICTS ET FERMEE PAR FUSIONS

Nous considérons maintenant le cas assez fréquent d'une sémantique qui n'est pas close mais sans traces préfixes strictes et fermée par fusions.

Exemple 2.7.3-1

(1) L'automate 2.2-1 engendre la sémantique 2.1.2-3 qui, réduite aux traces faiblement équitables pour $<\{a,b\}>$ est la sémantique 2.1.2-2 qui n'est pas fermée par limites puisque sa fermeture par limites est précisement la sémantique 2.1.2-3, d'après le lemme 2.6.7-5.1. La sémantique 2.1.2-3 n'est pas close mais elle est fermée par fusions et réduite par élimination des traces préfixes strictes.

(2) Il en va de même pour les programmes parallèles avec hypothèse d'exécution faiblement équitable des processus. Ceci correspond à l'idée que les processus sont exécutés par des processeurs différents dont aucun ne tombe indéfiniment en panne de sorte que les processus peuvent s'exécuter à des vitesses différentes mais aucun processus toujours prêt ne doit attendre indéfiniment. Cette hypothèse d'équité faible a (comme le montre l'exemple 2.7.3-1.1 ci-dessus) pour conséquence que la sémantique correspondante n'est pas fermée par limites.

De plus, comme les processeurs parallèles calculent à des vitesses différentes sans priorités, tous les entremêlements possibles des opérations effectuées par chaque processeur sont réalisables de sorte que l'état suivant un état donné ne dépend pas de la façon dont cet état donné a été atteint. Formellement ceci s'exprime par le fait que la sémantique est fermée par fusions.

Enfin si l'exécution peut globalement progresser (un processeur au moins n'étant pas arrêté), alors elle progressera [illegible] (un processeur au moins avec vitesse de calcul non nulle). Par conséquent, la sémantique est réduite par élimination des traces préfixes strictes.

Notons par rapport au cas général une simplification des méthodes (2.7.1 et 2.7.2.2) de spécification de sémantiques non closes à l'aide de systèmes de transitions :

2.7.3.1 Spécification par un système de transition et une condition sur les traces qu'il engendre

Lorsque $\underline{\text{Rétps}}(<S,A,\Sigma>) = <S,A,\Sigma>$ et $\underline{\text{Efus}}(<S,A,\Sigma>) = <S,A,\Sigma>$ et $<S,A,\Sigma>$ n'est pas close, on a $\Sigma \subset \Sigma^{<S,A,t<S,A,\Sigma>,\varepsilon<S,A,\Sigma>>}$ d'après le théorème 2.6.8-2 et la définition 2.6.8-2. Dans ce cas, il est donc possible de spécifier la sémantique $<S,A,\Sigma>$ par le système de transition $<S,A,t<S,A,\Sigma>,\varepsilon<S,A,\Sigma>>$ qu'elle engendre et une condition sur les traces engendrées par ce système de transition pour éliminer les traces parasites, c'est-à-dire celles de $(\Sigma^{<S,A,t<S,A,\Sigma>,\varepsilon<S,A,\Sigma>>} \setminus \Sigma)$.

Exemples 2.7.3.1-1

(1) La sémantique 2.1.2-2 peut se définir comme étant la sémantique engendrée par l'automate 2.2-1 avec la restriction aux traces finies.

(2) La sémantique des programmes parallèles avec hypothèse d'exécution faiblement équitable des processus est fermée par fusions, réduite par élimination des traces préfixes strictes mais pas fermée par limites (2.7.2.2-1.2). En associant une action différente à chaque processus (actions qui seront le nom du processus), nous pourrons spécifier cette sémantique $<S,A,\Sigma>$ à l'aide du système de transition $<S,A,t<S,A,\Sigma>,\varepsilon<S,A,\Sigma>>$ et d'une condition sur les traces de la sémantique $\underline{\text{Rtrau}}(<S,A,\Sigma>)$ ainsi définie pour éliminer les traces non équitables. Cette condition s'exprime [illegible] à l'opérateur de fermeture inférieure $\underline{\text{Wfair}}\langle x\rangle$ où α est l'ensemble des noms de processus.

Quand nous utilisons cette méthode, nous définissons d'abord un système de transition $\langle S, A, t, \varepsilon\rangle$ puis une condition C sur les traces pour obtenir la sémantique $\langle S, A, \{p \in \Sigma\langle S,A,t,\varepsilon\rangle : C(p)\}\rangle$. Pour faire des preuves, nous chercherons à faire des raisonnements utilisant un système de transition plutôt que des traces. Il est possible d'utiliser $\langle S,A,t,\varepsilon\rangle$ ou le système de transition qui engendre $R_{trace}(\langle S,A,\{p\in\Sigma\langle S,A,t,\varepsilon\rangle : C(p)\}\rangle)$. Ces deux systèmes de transition sont en général différents. Il se pose alors le problème de savoir s'il est plus facile de spécifier la sémantique $\langle S,A,\Sigma\rangle$ par un moyen quelconque et d'en déduire le système de transition $\langle S,A,t\langle S,A,\Sigma\rangle, \varepsilon\langle S,A,\Sigma\rangle\rangle$ ou si à l'inverse il vaut mieux commencer par spécifier un système de transition $\langle S,A,t,\varepsilon\rangle$ et en déduire $\langle S,A,\Sigma\rangle$ par restriction de la sémantique $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$. Sans pouvoir apporter de réponse générale à ce problème, une réponse est possible (ces deux systèmes de transition sont équivalents) dans chacun des cas particuliers que nous traitons.

Exemple 2.7.3.1-2

Le théorème 2.6.8.v.6 montre que dans le cas de programmes parallèles avec hypothèse d'exécution faiblement ou fortement équitable, il revient au même de spécifier la sémantique puis d'en déduire le système de transition ou de spécifier le système de transition puis d'en déduire la sémantique. Cette seconde solution est donc généralement retenue car un système de transition qui définit un pas de calcul est plus simple à spécifier qu'une sémantique qui définit une suite de pas de calcul.

□

2.7.3.2 Spécification par concordance avec une sémantique close

La technique est la même que dans le cas général:

Exemple 2.7.3.2-1

Nous pouvons définir la sémantique 2.1.2-2, version faiblement équitable de 2.1.2-3, en ajoutant un contrôleur à l'automate 2.2-1 comme suit :

$S^{\#} = \{0,1\} \times \omega$

$A^{\#} = \{a, b\}$

$t_a^{\#}(\langle A,m\rangle,\langle A',m'\rangle) = [A=A'=0 \wedge m'=m-1]$, $t_b^{\#}(\langle A,m\rangle,\langle A',m'\rangle) = [A=0 \wedge A'=1]$

$\varepsilon^{\#}(\langle A,m\rangle) = [A=0]$.

et en définissant la fonction $c^{\#}(\langle A,m\rangle) = A$.

□

Exemple 2.7.3.2-2

Nous pouvons définir la réduction aux traces faiblement équitables $W_{fair}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ d'une sémantique $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ engendrée par un système de transition $\langle S,A,t,\varepsilon\rangle$, en ajoutant un contrôleur au système de transition $\langle S,A,t,\varepsilon\rangle$ comme suit :

$S^{\boxtimes} = (A \to \omega) \times S$

$t^{\boxtimes}(\langle m,A\rangle,\langle m',A'\rangle) = [\exists k \in A.\ t_k(A,A') \wedge (\bigvee [m_k>0 \wedge m_k<m_k \wedge \forall j\in(A\cup k).\ m'_j=m_j] [\forall j\in A.((\forall A''\in S.\ \neg t_j(A,A')) \vee m_j=0) \wedge m'_j>0)])]$

$\varepsilon^{\boxtimes}(\langle m,A\rangle) = \varepsilon(A)$

et par concordance à la fonction $f_A^{\boxtimes}$ des états près, définie par

$f_A^{\boxtimes}(\langle m,A\rangle) = A$

Observons que le contrôleur organise l'exécution en reprises. Au début de la reprise, est déterminé le nombre maximal m_k de fois l'action $k \in A$ aura

le nombre maximum d'activations dans la reprise décroit strictement pour k et reste inchangé pour les autres actions. Une nouvelle reprise peut commencer dans l'état s si toute action $k \in A$ n'est pas activable dans l'état s ou a été activée au moins une fois dans la reprise précédente).

Lemme 2.7.3.2-2v1

$$\alpha\langle f_A^{\natural}\rangle(\langle S^{\natural}, A, \Sigma\langle S^{\natural}, A, t^{\natural}, \varepsilon^{\natural}\rangle\rangle) = \underset{\sim}{W}\text{fair}(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle)$$

Démonstration

Montrons que si $p^{\natural} \in \Sigma\langle S^{\natural}, A, t^{\natural}, \varepsilon^{\natural}\rangle$ alors $f_A^{\natural}(p^{\natural})$ est une trace faiblement équitable de $\Sigma\langle S, A, t, \varepsilon\rangle$. C'est évident si $p^{\natural}$ est une trace finie. Sinon, il existe $m \in (\omega \to (A \to \omega))$ et $p \in (\omega \to S)$ tels que $\forall i \in \omega.\ (p_i^{\natural} = \langle m_i, p_i\rangle)$. Supposons que $p = f_A^{\natural}(p^{\natural})$ ne soit pas faiblement équitable pour A, de sorte que $\exists k \in A, i \in \omega.\ \forall j \geq i.\ (\exists a \in S.\ t_k(p_j, a) \wedge \neg t_k(p_j, p_{j+1}))$. Dans une reprise, la somme des m_{jk} pour $k \in A$ décroit strictement à chaque pas, de sorte qu'aucune reprise ne peut être infinie. En particulier, la reprise à laquelle "i appartient" doit se terminer en $i' \geq i$. Au début de la reprise suivante, nous avons $m_{(i'+1)k} > 0$ et les m_{jk} ne sont pas modifiés pendant cette reprise puisque l'action k n'est jamais activée. Quand cette dernière reprise se termine en $i'' > i'$, nous avons $\exists a \in S.\ t_k(p_{i''}, a) \wedge m_{i''k} = m_{(i'+1)k} > 0$ en contradiction avec la définition de $t^{\natural}$.

Il reste à montrer que si p est une trace de $\Sigma\langle S, A, t, \varepsilon\rangle$, faiblement équitable pour A, nous pouvons construire $m \in (|p| \to (A \to \omega))$ tel que $p^{\natural}$ définie par $(|p^{\natural}| = |p| \wedge \forall i \in |p|.\ p_i^{\natural} = \langle m_i, p_i\rangle)$ (et donc telle que $f_A^{\natural}(p^{\natural}) = p$) appartienne à $\Sigma\langle S^{\natural}, A, t^{\natural}, \varepsilon^{\natural}\rangle$.

Si $|p| = 1$, posons $m_{0k} = 0$

Si $1 < |p| < \omega$, posons $m_{ik} = \sum_{j<i}^{|p|-2} (t_k(p_j, p_{j+1}) \to 1 | 0)$

Si $|p| = \omega$, alors grâce à l'hypothèse d'équité faible, nous pouvons définir $\gamma \in (\omega \to \omega)$ par

$\gamma_0 = 0$ et $\gamma_{i+1} = \wedge\{j \in \omega : \forall k \in m.\ [(\forall l \geq \gamma_i.\ \forall a \in s.\ \neg t_k(p_l, a)) \vee (\exists l \in \omega.\ \gamma_i \leq l < j \wedge t_k(p_l, p_{l+1}))]\}$ de sorte que toutes les actions qui ne sont pas continuellement bloquées après γ_i sont activées au moins une fois entre γ_i et γ_{i+1}. Nous définissons $\eta \in (\omega \to \omega)$ tel que η_i est le γ_{j+1} tel que $\gamma_j \leq i < \gamma_{j+1}$. Posons $m_{ik} = \sum_{j<i}^{\eta_i} (t_k(p_j, p_{j+1}) \to 1 | 0)$.

□

□

2.8 EXEMPLE DE DEFINITION DE LA SEMANTIQUE D'UN LANGAGE DE PROGRAMMATION

Nous définissons la sémantique d'un langage que nous utiliserons ultérieurement comme exemple.

Les programmes ($\mathcal{P}_r$) peuvent être séquentiels ($\mathcal{P}_s$), parallèles asynchrones ($\mathcal{P}_{pa}$), parallèles communicant par envois de messages sur rendez-vous ($\mathcal{P}_{pc}$), parallèles faiblement équitables ($\mathcal{P}_{pw}$) ou parallèles synchronisés par sémaphores ($\mathcal{P}_{ps}$)

$\mathcal{P}_r \rightarrow \mathcal{P}_s \mid \mathcal{P}_{pa} \mid \mathcal{P}_{pc} \mid \mathcal{P}_{pw} \mid \mathcal{P}_{ps}$

Après avoir défini la syntaxe de ces programmes, nous en définissons la sémantique. Pour cela nous associons à tout programme P_r syntaxiquement correct un système de transition $\langle S[\![P_r]\!], A[\![P_r]\!], t[\![P_r]\!], \varepsilon[\![P_r]\!]\rangle$ qui est spécifié par induction sur la syntaxe du programme.
La sémantique des programmes séquentiels, parallèles asynchrones et parallèles communicants est close. Elle peut donc être définie comme la sémantique engendrée par ce système de transition, (cf. 2.7.1).
La sémantique des programmes parallèles faiblement équitables et celle des programmes avec sémaphores est réduite par élimination des traces préfixes strictes, fermée par fusion mais n'est pas close. Elle sera définie par élimination des traces non équitables de l'ensemble des traces engendrées par ce système de transitions (cf. 2.7.3.1) ou bien à une fonction des états près en incorporant un contrôleur dans ce système de transition (cf. 2.7.3.2).

2.8.1 PROGRAMMES SEQUENTIELS

Un programme séquentiel est une suite de commandes exécutées en séquence. Une commande peut être nulle, une affectation, une affectation aléatoire ou une composition alternative ou itérative de commandes. Chaque commande est précédée et suivie par une étiquette qui n'apparaît qu'une seule fois dans le programme et qui sert uniquement à désigner des points de contrôle du programme. Pour simplifier, les déclarations de types et de variables sont omises.

2.8.1.1 Syntaxe

Soient $\mathcal{L}, \mathcal{V}, \mathcal{E}$ et $\mathcal{B}$ des ensembles non vides donnés, respectivement d'étiquettes, de variables, d'expressions et d'expressions booléennes dont nous omettrons la syntaxe pour simplifier.

Les ensembles $\mathcal{P}_s$ de programmes séquentiels et $\mathcal{C}$ de commandes séquentielles sont définis par la syntaxe suivante :

$\mathcal{P}_s \rightarrow \mathcal{L}_0 : \mathcal{C}_0 ; \ldots ; \mathcal{L}_{m-1} : \mathcal{C}_{m-1} ; \mathcal{L}_m :$ $\quad (m>0)$

$\mathcal{C} \rightarrow$ skip $\mid \mathcal{V} := \mathcal{E} \mid \mathcal{V} := ? \mid$ if $\mathcal{B}$ then $\mathcal{P}_s$ else $\mathcal{P}_s$ fi $\mid$ while $\mathcal{B}$ do $\mathcal{P}_s$ od

(if $\mathcal{B}$ then $\mathcal{P}_s$ fi est l'abréviation de if $\mathcal{B}$ then $\mathcal{P}_s$ else skip fi)

Soient N une chaîne terminale dérivant du non-terminal $\mathcal{N}$, α_i des chaînes éventuellement vides et N_i des chaînes terminales non vides dérivant respectivement des non-terminaux $\mathcal{N}_i$, nous utiliserons la notation $N = \alpha_0 N_0 \ldots \alpha_m N_m \alpha_{m+1}$ pour $\exists \alpha_{0,-1}, \alpha_{m+1} \cdot (\ldots N = \alpha_0 N_0 \ldots \alpha_m N_m \alpha_{m+1} \ldots)$, le quantificateur étant conventionnellement placé pour lier toutes les occurences des $\alpha_{0,-1}, \alpha_{m+1}$ dans la [illegible] où il est utilisé.

Cette notation sera utilisée pour exprimer des conditions de contexte sur les programmes comme par exemple qu'une étiquette ne peut

apparaître plus d'une fois dans un programme :

$\forall Pr \in \mathcal{P}r.\ (Pr \equiv \alpha L_1 : \beta L_2 : \delta) \Rightarrow (L_1 \neq L_2)$

2.8.1.2 Sémantique

2.8.1.2.1 Etats

Un état d'un programme séquentiel Ps est une paire <état de contrôle, état mémoire>. L'état de contrôle est similaire à un compteur de programme. Il est représenté par une étiquette L qui désigne un point unique du programme. Supposons que les variables prennent leurs valeurs dans un domaine $\mathcal{D}$ (que nous ne spécifions pas pour simplifier). L'état mémoire M est un élément de $\mathcal{M} = (\mathcal{V} \to \mathcal{D})$ c'est-à-dire une fonction totale de l'ensemble $\mathcal{V}$ des variables dans l'ensemble $\mathcal{D}$ des valeurs des variables.

Formellement, l'ensemble $S[\![Ps]\!]$ des états du programme Ps est :

$$S[\![Ps]\!] = \{L \in \mathcal{L} :\ Ps \equiv \alpha L : \beta\} \times \mathcal{M}$$

2.8.1.2.2 Actions

N'ayant pas besoin dans la suite de nommer les actions qui sont exécutées par les programmes séquentiels, nous définissons :

$A[\![Ps]\!] = \{a\}$

où a est l'action unique que nous emploierons dans la suite.

2.8.1.2.3 Etats initiaux

Les états initiaux d'un programme séquentiel Ps sont caractérisés par :

$\varepsilon[\![Ps]\!] \in (S[\![Ps]\!] \to \{tt, ff\})$

$\varepsilon[\![Ps]\!](\langle L, M\rangle) = [Ps \equiv L : \alpha]$

2.8.1.2.4 Relation de transition

La relation de transition $t[\![Ps]\!]$ entre un état $\langle L, M\rangle$ du programme séquentiel Ps et ses successeurs $\langle L', M'\rangle$ (s'il en existe) est définie par :

$t[\![Ps]\!] \in ((S[\![Ps]\!] \times S[\![Ps]\!]) \to \{tt, ff\})$

$t[\![Ps]\!](\langle L, M\rangle, \langle L', M'\rangle) = [\underline{cond}[\![Ps]\!](L, L')(M) \wedge \underline{succ}[\![Ps]\!](L)(M, M')]$

$\langle L', M'\rangle$ est l'état successeur de $\langle L, M\rangle$ si et seulement si l'exécution d'un pas du programme séquentiel Ps au point de contrôle L dans l'état mémoire M a pour effet de déplacer le contrôle en L' (tel que $\underline{cond}[\![Ps]\!](L, L')(M)$ est vrai) et de changer l'état mémoire en M' (tel que $\underline{succ}[\![Ps]\!](L)(M, M')$ est vrai).

L'état mémoire ne peut être modifié que par une commande d'affectation. Une affectation aléatoire $V := ?$ affecte une valeur quelconque de $\mathcal{D}$ à la variable V. Une affectation $V := E$ affecte à V la valeur $\mathbb{E}[\![E]\!](M)$ de l'expression E dans l'état mémoire M à la variable V. Pour simplifier, la sémantique $\mathbb{E} \in (\mathcal{E} \to ((\mathcal{V} \to \mathcal{D}) \to \mathcal{D}))$ des expressions n'est pas spécifiée.

$$\underline{succ}[\![Ps]\!](L)(M, M') = (Ps \equiv \alpha L :\ V := E;\ \beta \Rightarrow \forall W \in (\mathcal{V} \sim V).\ [M'(W) = M(W)] \wedge M'(V) = \mathbb{E}[\![E]\!](M)$$

$$| (Ps \equiv \alpha L :\ V := ?;\ \beta \Rightarrow \forall W \in (\mathcal{V} \sim V).\ [M'(W) = M(W)]$$

$| M', M'')$

Pour définir le contrôle nous supposons donnée la sémantique $\mathbb{B} \in (\mathcal{B} \to ((\mathcal{V} \to \mathcal{D}) \to \{tt, ff\}))$ des expressions booléennes telle que $\mathbb{B}[\![B]\!](M)$ soit la valeur de l'expression booléenne B dans l'état mémoire M. $\mathbb{E}[\![E]\!]$ et $\mathbb{B}[\![B]\!]$ sont des fonctions partielles pour rendre compte des erreurs possibles à l'exécution. La relation de transition $\underline{cond}[\![Ps]\!](L,L')(M)$ entre l'état de contrôle L et son successeur L' dans l'état mémoire M est définie par cas comme suit :

$\underline{cond}[\![Ps]\!](L,L')(M) =$

$[$

$[$ $Ps \equiv \alpha L : \underline{skip}\ ; L' : \beta$

$\vee$ $Ps \equiv \alpha L : V := E ; L' : \beta \ \wedge\ M \in \underline{dom}(\mathbb{E}[\![E]\!])$

$\vee$ $Ps \equiv \alpha L : V := ? ; L' : \beta$

$\vee$ $Ps \equiv \alpha L : \underline{else}\ Ps'\ \underline{fi} ; L' : \beta$

$\vee$ $Ps \equiv \alpha L : \underline{fi} ; L' : \beta$

$]$

$\vee$

$[($ $Ps \equiv \alpha L : \underline{if}\ B\ \underline{then}\ L' : \beta$

$\vee$ $Ps \equiv \alpha L : \underline{while}\ B\ \underline{do}\ L' : \beta$

$\vee$ $Ps \equiv \alpha\ \underline{while}\ B\ \underline{do}\ L' : I_0 ; \dots ; L_{m-1} : I_{m-1} ; L : \underline{od} ; \beta$

$)$

$\wedge$ $M \in \underline{dom}(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M)$

$]$

$\vee$

$[($ $Ps \equiv \alpha L : \underline{if}\ B\ \underline{then}\ Ps'\ \underline{else}\ L' : \beta$

$\vee$ $Ps \equiv \alpha L : \underline{while}\ B\ \underline{do}\ Ps'\ \underline{od} ; L' : \beta$

$\vee$ $Ps \equiv \alpha\ \underline{while}\ B\ \underline{do}\ L_0 : I_0 ; \dots ; L_{m-1} : I_{m-1} ; L : \underline{od} ; L' : \beta$

$)$

$\wedge$ $M \in \underline{dom}(\mathbb{B}[\![B]\!]) \wedge \neg \mathbb{B}[\![B]\!](M)$

$]$

$]$

Par exemple, si le contrôle est au point L avant d'exécuter une boucle while ou après avoir exécuté son corps alors le contrôle passe au point L' qui désigne la première instruction de son corps si l'état mémoire est bien défini et ainsi le contrôle sort de la boucle si l'état est bien défini et faux et le programme s'arrête en L (qui n'a pas de successeur) si l'état n'est pas défini (ce qui constitue une erreur à l'exécution).

2.8.1.2.5 Traces

La sémantique d'un programme séquentiel $Ps \in \mathcal{P}s$ est :

$\langle S[\![Ps]\!], A[\![Ps]\!], \Sigma < S[\![Ps]\!], A[\![Ps]\!], t[\![Ps]\!], \varepsilon[\![Ps]\!] \rangle$

2.8.1.3 Exemple

Le programme séquentiel suivant calcule z^n quand $n \geq 0$:

```
0:
    P:=1;
1:
    while N>0 do
2:      P:=z×P;
3:      N:=N-1;
4:
    od;
5:
```

et sera utilisé pour illustrer quelques méthodes de preuve.

2.8.2 PROGRAMMES PARALLELES ASYNCHRONES

2.8.2.1 Syntaxe

Un programme parallèle asynchrone Ppa est composé d'un prélude séquentiel Ps, suivi de processus séquentiels $[\![Pra_0 \| \ldots \| Pra_{m-1}]\!]$ partageant des variables globales communes et se déroulant en parallèle de manière asynchrone, suivis d'un postlude séquentiel Ps' :

$$Ppa \longrightarrow Ps\ [\![Pra_0 \| \ldots \| Pra_{m-1}]\!];\ Ps' \qquad (m \geq 1)$$

Les processus asynchrones Pra_i, $i \in m$ sont des listes de commandes asynchrones étiquetées :

$$Pra \longrightarrow \mathcal{L}_0 : Ca_0 ; \ldots ; \mathcal{L}_{m-1} : Ca_{m-1}, \mathcal{L}_m : \qquad (m \geq 1)$$

Chaque commande asynchrone est une commande séquentielle (cf. 2.8.1.1) ou bien une liste de commandes séquentielles Ps exécutées de manière indivisible, ce que nous notons $\langle Ps \rangle$:

$$Ca \longrightarrow \underline{skip} \mid V := E \mid V :=? \mid \underline{if}\ B\ \underline{then}\ Pra_1\ \underline{else}\ Pra_2\ \underline{fi} \mid$$
$$\underline{while}\ B\ \underline{do}\ Pra\ \underline{od} \mid \langle Ps \rangle$$

2.8.2.2 Sémantique

2.8.2.2.1 Etats

Un état d'un programme $Ppa = Ps\ [\![Pra_0 \| \ldots \| Pra_{m-1}]\!];\ Ps'$ est une paire <état de contrôle, état mémoire> où l'état mémoire affecte des valeurs aux variables partagées et l'état de contrôle est une étiquette au sein du contrôle est dans le prélude Ps ou le postlude Ps' ou un tuple [illegible] qui désignent respectivement l'état de

contrôle des processus $Pra_0, \ldots, Pra_{m-1}$ qui se déroulent en parallèle. Formellement nous définissons :

$$S[\![Ppa]\!] = \left(\{ L \in \mathcal{L} : Ps \equiv \alpha L : \beta \vee Ps' \equiv \alpha L : \beta \} \cup \prod_{i \in m} \{ L \in \mathcal{L} : Pra_i \equiv \alpha L : \beta \} \right) \times \mathcal{M}$$

2.8.2.2.2 Actions

Les actions du programme $Ps[\![Pra_0 \| \ldots \| Pra_{m-1}]\!]; Ps'$ sont $\pounds, 0, \ldots, m-1, \pounds'$ correspondant respectivement à un pas dans le prélude, le processus Pra_i et le postlude :

$$A[\![Ppa]\!] = \{ \pounds, \pounds' \} \cup m$$

2.8.2.2.3 Etats initiaux

L'exécution du programme commence au point d'entrée du prélude dans un état mémoire quelconque :

$$\varepsilon[\![Ppa]\!] \in (S[\![Ppa]\!] \to \{tt, ff\})$$
$$\varepsilon[\![Ppa]\!](A) = [\exists L \in \mathcal{L}, M \in \mathcal{M} .\ A = \langle L, M \rangle \wedge Ppa \equiv L : \alpha]$$

2.8.2.2.4 Relation de transition

La relation de transition

$$t[\![Ppa]\!] \in (A[\![Ppa]\!] \to ((S[\![Ppa]\!] \times S[\![Ppa]\!]) \to \{tt, ff\}))$$

est définie par cas :

- L'exécution du prélude et du postlude est purement séquentielle :

$$t[\![Ppa]\!]_{\sharp}(\langle L,M\rangle,\langle L',M'\rangle) = [Ppa \equiv Ps[\![Pra_0 \| \dots \| Pra_{m-1}]\!]; Ps' \wedge t[\![Ps]\!](\langle L,M\rangle,\langle L',M'\rangle)]$$
$$t[\![Ppa]\!]_{\sharp'}(\langle L,M\rangle,\langle L',M'\rangle) = [Ppa \equiv Ps[\![Pra_0 \| \dots \| Pra_{m-1}]\!]; Ps' \wedge t[\![Ps']\!](\langle L,M\rangle,\langle L',M'\rangle)]$$

- Les exécutions des processus parallèles commencent simultanément :

$$t[\![Ppa]\!]_{\sharp}(\langle L,M\rangle,\langle\langle L_0,\dots,L_{m-1}\rangle,M'\rangle) = [Ppa \equiv \alpha L_i[\![L_0:\alpha_0 \| \dots \| L_{m-1}:\alpha_{m-1}]\!]; Ps' \wedge M'=M]$$

- Nous rendons compte de l'exécution parallèle des processus par un mélange nondéterministe d'exécutions d'actions atomiques. Chaque pas du programme consiste donc à exécuter de manière indivisible une action atomique d'un processus Pra_i tandis que les autres processus Pra_j, $j \in (m \setminus i)$ n'évoluent pas. L'exécution d'une action atomique indivisible correspond à l'évaluation d'une affectation ou d'un test d'une commande séquentielle (cf. 2.7.1.2.4) ou encore à l'exécution d'une liste de commandes séquentielles $\{Ps\}$ en exclusion mutuelle. Formellement :

$$t[\![Ppa]\!]_i(\langle\langle L_0,\dots,L_{m-1}\rangle,M\rangle,\langle\langle L'_0,\dots,L'_{m-1}\rangle,M'\rangle) =$$
$$[Ppa \equiv Ps[\![Pra_0 \| \dots \| Pra_{m-1}]\!]; Ps' \wedge \forall_j \in (m \setminus i).\ L'_j = L_j \wedge$$
$$(t[\![Pra_i]\!](\langle L_i,M\rangle,\langle L'_i,M'\rangle) \vee (Pra_i \equiv \alpha L_i:\{Ps''\};\beta \wedge Ps'' \equiv L:\delta L: \wedge\ t[\![Ps']\!]^*(\langle L,M\rangle,\langle L',M'\rangle)$$

- L'exécution de la commande parallèle se termine quand tous les processus ont terminé leur exécution :

$$t[\![Ppa]\!]_{\sharp}(\langle\langle\langle L_0,\dots,L_{m-1}\rangle,M\rangle,\langle L,M'\rangle) = [Ppa \equiv Ps[\![\alpha_0;L_0: \| \dots \| \alpha_{m-1};L_{m-1}:]\!]; L:\beta \wedge M'=M]$$

2.8.2.2.5 Traces

ce qui correspond à une exécution parallèle des processus sans aucune hypothèse d'équité. Par conséquent, l'exécution du programme

B := true ; [[B := false || while B do skip od]]

peut ne jamais se terminer si à chaque pas de l'exécution le second processus est activé tandis que le premier processus est toujours en attente.

2.8.2.3 Exemple

Le programme parallèle asynchrone suivant calcule 2^m pour $m \geq 0$:

```
0:  [[
11:       P1 := 1;
12:       while N>1 do
13:             ⟨ 131:    N := N-1;
                  132:    P1 := 2×P1;
                  133: ⟩;
14:       od;
15:
    ||
21:       P2 := 1;
22:       while N>1 do
23:             ⟨ 231:    N := N-1;
                  232:    P2 := 2×P2;
                  233: ⟩;
24:       od;
25:
    ]];
1:  if N=0 then P := P1×P2 else P := 2×P1×P2 fi;
```

(Remarquons que seule la commande d'affectation N:=N-1 doit être atomique. Aucune condition d'équité n'est nécessaire sur l'exécution des processus car le calcul peut être entièrement fait par l'un des processus).

2.8.3 PROGRAMMES PARALLELES COMMUNICANTS

2.8.3.1 Syntaxe

Pour décrire des programmes parallèles distribués où les processus se synchronisent et communiquent par des commandes d'envoi et de réception de messages sur rendez-vous, nous utilisons une variante de CSP (Hoare[78]) où les commandes de communication utilisent des canaux au lieu du nom des autres processus. Par simplicité également nous n'utiliserons pas de variables locales aux processus mais des variables globales (qui ne peuvent être ni consultées ni modifiées par les autres processus) :

$$Ppc \longrightarrow Ps \,[\![\, Prc_0 \,\|\, \ldots \,\|\, Prc_{m-1} \,]\!]\, ; Ps' \qquad (m>1)$$

Un processus est une liste séquentielle de commandes :

$$Prc \longrightarrow \mathcal{L}_0 : Cc_0 \,;\, \ldots \,;\, \mathcal{L}_{m-1} : Cc_{m-1} \,;\, \mathcal{L}_m : \qquad (m>1)$$

chaque commande étant une commande asynchrone (cf. 2.8.2.1) ou un envoi de message sur rendez-vous, une réception de message sur rendez-vous ou encore une commande alternative qui permet de sélectionner une communication parmi plusieurs alternatives. Soit Ch un ensemble de canaux, nous avons :

$$Cc \longrightarrow \underline{skip} \mid V := E \mid V := ? \mid \underline{if}\ B\ \underline{then}\ Prc_1\ \underline{else}\ Prc_2\ \underline{fi} \mid \underline{while}\ B\ \underline{do}\ Prc\ \underline{od} \mid \langle Ps \rangle \mid Ch!E \mid Ch?V \mid \underline{se}\ Alt_0\ \underline{or}\ \ldots\ \underline{or}\ Alt_e\ \underline{es}$$

$$Alt \longrightarrow B\,;\, Ch!E\ \underline{then}\ Prc \mid B\,;\, Ch?V\ \underline{then}\ Prc$$

2.8.3.2 Sémantique

Hoare n'ayant fait aucune hypothèse d'équité sur l'exécution des programmes CSP, la sémantique des programmes parallèles communicants est similaire à celle des programmes asynchrones sauf en ce qui concerne les commandes de communication. Dans la suite du paragraphe, nous avons :

$Ppc \equiv Ps \llbracket Prc_0 \| \ldots \| Prc_{m-1} \rrbracket ; Ps'$

2.8.3.2.1 Etats

Comme en 2.8.2.2.1, nous définissons :

$$S\llbracket Ppc \rrbracket = \left(\{L \in \mathcal{L} : Ps \equiv \alpha L : \beta \vee Ps \equiv \alpha L : \beta\} \cup \bigcup_{i \in m} \{L \in \mathcal{L} : Prc_i \equiv \alpha L : \beta\}\right) \times \mathcal{M}$$

2.8.3.2.2 Actions

Les actions du programme $Ps \llbracket Prc_0 \| \ldots \| Prc_{m-1} \rrbracket ; Ps'$ sont $\sharp, 0, \ldots, m-1, \sharp'$ correspondant respectivement à un pas dans le prélude, le processus Prc_i et le postlude comme en 2.8.2.2.2, plus l'ensemble des canaux intervenant dans le programme

$$A\llbracket Ppc \rrbracket = \{\sharp, \sharp'\} \cup m \cup \{\underline{Ch} \in \mathcal{C}h : Ppc \equiv \alpha Ch \beta\}$$

2.8.3.2.3 Etats initiaux

Comme en 2.8.2.2.3, nous définissons :

$$\mathcal{E}\llbracket Ppc \rrbracket \in [S\llbracket Ppc \rrbracket \to \mathbb{B}]$$

$$\mathcal{E}\llbracket Prc \rrbracket(\lambda) = [\exists L \in \mathcal{L}^{\mathcal{C}}\ M \in \mathcal{M}_{\mathcal{C}} . \ \lambda = \langle L, M \rangle \wedge Ppc \equiv L : \alpha]$$

2.8.3.2.4 Relation de transition

L'exécution d'un pas d'un programme parallèle communicant est similaire à celle d'un programme asynchrone sauf en ce qui concerne les commandes de communication. Par rapport à 2.8.2.2.4, nous rajoutons donc à la relation de transition un cas qui correspond à l'exécution d'une communication :

$$\begin{aligned}
&t\llbracket Ppc \rrbracket_{\underline{Ch}}(\langle\langle L_0, \ldots, L_{m-1} \rangle, M \rangle, \langle\langle L'_0, \ldots, L'_{m-1} \rangle, M' \rangle) = \\
&\quad [\, Ppc \equiv Ps \llbracket Prc_0 \| \ldots \| Prc_{m-1} \rrbracket ; Ps' \ \wedge \\
&\quad \exists i, j \in m . \ (Prc_i \equiv \alpha L_i : Ch ! E ; L'_i : \beta \\
&\qquad \vee (Prc_i \equiv \alpha L_i : \underline{ae}\ Alt_0\ \underline{or} \ldots \underline{or}\ Alt_{\ell-1}\ \underline{ea} \ \wedge \exists k \in \ell . \\
&\qquad\quad Alt_k \equiv B_1 ; Ch ! E\ \underline{then}\ L'_i : \eta \ \wedge M \in \underline{dom}(\mathbb{B}\llbracket B_1 \rrbracket) \wedge \mathbb{B}\llbracket B_1 \rrbracket(M))) \\
&\qquad \wedge M \in \underline{dom}(\mathbb{E}\llbracket E \rrbracket) \\
&\qquad \wedge (Prc_j \equiv \gamma L_j : Ch ? V ; L'_j : \delta \\
&\qquad\quad \vee (Prc_j \equiv \gamma L_j : \underline{ae}\ Alt'_0\ \underline{or} \ldots \underline{or}\ Alt'_p\ \underline{ea} \ \wedge \exists q \in p . \\
&\qquad\qquad Alt'_q \equiv B_2 ; Ch ? V\ \underline{then}\ L'_j : \xi \ \wedge M \in \underline{dom}(\mathbb{B}\llbracket B_2 \rrbracket) \wedge \mathbb{B}\llbracket B_2 \rrbracket(M))) \\
&\qquad \wedge \forall k \in (m \setminus \{i, j\}) . \ L'_k = L_k \\
&\qquad \wedge M'(V) = \mathbb{E}\llbracket E \rrbracket(M) \ \wedge \ \forall W \in (\mathcal{V} \setminus V) . \ (M'(W) = M(W)) \,]
\end{aligned}$$

Nous rajoutons également le cas correspondant à la sortie d'une commande alternative :

$$\begin{aligned}
&t\llbracket Ppc \rrbracket_i(\langle\langle L_0, \ldots, L_{m-1} \rangle, M \rangle, \langle\langle L'_0, \ldots, L'_{m-1} \rangle, M' \rangle) = \\
&\quad [Ppc \equiv Ps \llbracket Prc_0 \| \ldots \| Prc_{m-1} \rrbracket ; Ps' \ \wedge \ \forall j \in (m \setminus i) . \ L'_j = L_j \ \wedge \\
&\quad (t\llbracket Prc_i \rrbracket(\langle L_i, M \rangle, \langle L'_i, M' \rangle) \\
&\qquad \vee (Prc_i \equiv \alpha L_i : \sharp\, Ps''\, \sharp ; L'_i : \beta \ \wedge \ Ps'' \equiv L : \gamma ; L' : \ \wedge \ t\llbracket Ps'' \rrbracket^*(\langle L, M \rangle, \langle L', M' \rangle)) \\
&\qquad \vee (Prc_i \equiv \chi\ \underline{ae}\ Alt_0\ \underline{or} \ldots \underline{or}\ \beta L_i : \underline{or} \ldots \underline{or}\ Alt_{\ell-1}\ \underline{ea} ; L'_i : \gamma \ \wedge \ M' = M)) \,]
\end{aligned}$$

2.8.3.2.5 Traces

La sémantique d'un programme parallèle communicant Ppc est :

$\langle S[\![Ppc]\!], A[\![Ppc]\!], \Sigma < S[\![Ppc]\!], A[\![Ppc]\!], t[\![Ppc]\!], \epsilon[\![Ppc]\!] >>$

2.8.3.3 Exemples

Exemple 2.8.3.3-1

Le programme parallèle suivant réalise une section critique. Remarquons que le troisième processus joue le rôle de contrôleur. (Ayant défini (avec les notations Pascal) le type signal = (any) et la la variable Amy : signal, nous écrivons P!Amy (respectivement P?Amy) pour la transmission (respectivement reception) d'un signal).


```
0:  [[ 11:  while true do
       12:      P! any ;
       13:      V! any ;
       14:  od ;
       15:
    ||
       21:  while true do
       22:      P! any ;
       23:      V! any ;
       24:  od;
       25;
    ||
       31:  while true do
       32:      P? Amy ;
       33:      V? Amy;
       34:  od ;
       35:
    ]];
1:
```

Remarquons que l'entrée dans une section critique donnée n'est pas fatale.

Par contre, la version suivante du programme est équitable. (Nous codons la valeur du sémaphore et le contenu de la file d'attente comme suit :

- $s=0$ le sémaphore est ouvert
- $s=i$ le sémaphore est fermé. Il a été franchi par le processus i, $i=1,2$ et ensuite le processus $\tilde{i}$ n'a pas demandé à le franchir (où nous notons $\tilde{1}=2$ et $\tilde{2}=1$)
- $s=i\tilde{i}$ le sémaphore est fermé. Il a été franchi par le processus i et ensuite le processus $\tilde{i}$ a demandé à le franchir).

```
0:
[[ 11:
   while true do
12:    Sem! ('P', 1);        {demande d'entrée en section critique}
13:    Go1? Any ;            {autorisation d'entrée en section critique}
14:    Sem! ('V', 1);        {sortie de la section critique}
15:
   od;
16:
||
21:
   while true do
22:    Sem! ('P', 2);
23:    Go2? Any;
24:    Sem! ('V', 2);
25:
   od;
26:
||
31:
   S:= 0;
   while true do
      Sem? M;
      if (M=('P',1) ∧ S=0) ∨ (M=('V',2) ∧ S=21) then Go1! Any; S:=1;
      else if (M=('P',2) ∧ S=0) ∨ (M=('V',1) ∧ S=12) then Go2! Any; S:=2;
      else if (M=('P',1) ∧ S=2) then S:= 21 ;
      else if (M=('P',2) ∧ S=1) then S:=12 ;
      else {(M=('V',1) ∧ S=1) ∨ (M=('V',2) ∧ S=2)} S:=0 ;
   od;
]],
1:
```


Exemple 3.8.3.3.2

Le programme suivant (Hoare [78]) réalise la transmission de messages d'un producteur vers un consommateur via un tampon pour régulariser les flux ; les variables sont du type suivant :

```
type signal = (any);
N : 1.. maxint;
output, Input : array [0..N-1] of type-messages;
Buffer        : array [0..9] of type-messages;
I, Im, Out, J : integer;
Pe, Ft, Te    : boolean;
PTm, TCm      : channel type-messages;
PTs, TCs      : channel signal;
Any           : signal;
```

```
0:
[[ {Producteur :} 11:
                  I:=0;
               12:
                  while I<>N do
               13:
                     PTm ! output[I];
               14:
                     I:= I+1;
               15:
                  od;
               16:
                  PTs ! Any ;
               17:
```

```
||
{Tampon}  201:   Im:=0;  Out:=0;  Pe:= false;  Ft:= false;
          202:   while not Ft do
          203:     as ¬Pe ∧ Im<Out+10 ; PTm? Buffer[Im mod 10] then
          204:          Im := Im+1;
          205:     or  Out<Im;  TCm ! Buffer [Out mod 10] then
          206:          Out:= Out+1;
          207:     or  ¬Pe;  PTA? Any  then
          208:          Pe:= true;
          209:     or   Pe ∧ (Im=Out);  TCA ! Any  then
          210:          Ft := true ;
          211:     sa;
          212:   od;
          213:

||
{Consommateur:} 31:   J:=0;  Te:= false;
          32:   while ¬Te  do
          33:     as  true ;  TCm ? Imput[J]  then
          34:          J:= J+1;
          35:     or   true ;  TCA ? Any  then
          36:          Te:= true ;
          37:     sa;
          38:   od;
          39:

||,
1:
```


2.8.4 PROGRAMMES PARALLELES FAIBLEMENT EQUITABLES

2.8.4.1 Syntaxe

Pour simplifier nous adoptons la même syntaxe que pour les programmes asynchrones :

$\mathcal{P}pw \longrightarrow \mathcal{P}pa$

2.8.4.2 Sémantique

La sémantique d'un programme parallèle faiblement équitable peut être définie de manière équivalente (cf. 2.7.3.2)

- comme la restriction $\underset{\sim}{W}fair(\langle s[\![\mathcal{P}pw]\!], A[\![\mathcal{P}pw]\!], \Sigma\langle s[\![\mathcal{P}pw]\!], A[\![\mathcal{P}pw]\!], t[\![\mathcal{P}pw]\!], \varepsilon[\![\mathcal{P}pw]\!]\rangle\rangle$ aux traces faiblement équitables de la sémantique des programmes parallèles
- par concordance avec la sémantique close $\langle s[\![\mathcal{P}pw]\!]^{\natural}, A[\![\mathcal{P}pw]\!], \Sigma\langle s[\![\mathcal{P}pw]\!]^{\natural}, A[\![\mathcal{P}pw]\!]\ t[\![\mathcal{P}pw]\!]^{\natural}, \varepsilon[\![\mathcal{P}pw]\!]^{\natural}\rangle\rangle$ à la fonction $fs^{\natural}$ entre états près.

2.8.4.3 Exemple

Soit $F \in (\mathbb{Z} \to \mathbb{Z})$ un opérateur sur les entiers. S'il possède un point fixe alors l'exécution du programme parallèle faiblement équitable suivant se termine avec une valeur finale p de P telle que $F(p)=p$.
Le premier processus cherche un point fixe parmi les entiers strictement négatifs et le deuxième processus cherche parmi les entiers positifs ou nuls. L'équité faible est donc nécessaire pour garantir la terminaison ;

```
0:
   S1 := true ;  S2 := true
1: [[
      10:
         L := -1;
      11:
         while S1 ∧ S2 do
      12:
            if F(L) = L then
      13:
               S2 := false;
      14:
            else
      15:
               L := L-1;
      16:
            fi;
      17:
         od;
      18:
   ||
      20:
         H := 0;
      21:
         while S1 ∧ S2 do
      22:
            if F(H) = H then
      23:
               S2 := false;
      24:
            else
      25:
               H := H+1;
      26:
            fi;
      27:
         od;
      28:
   ]];
2: if ¬S1 then
3:    P := L;
4: else if ¬S2 then
5:    P := H;
6: fi;
7:
```


2.8.5 PROGRAMMES PARALLELES SYNCHRONES

De nombreuses solutions ont été proposées pour synchroniser des processus parallèles partageant des données communes. En général, elles peuvent s'implanter à l'aide de la notion primitive de sémaphores (Dijkstra [68]). C'est la solution que nous avons retenue comme exemple en raison de sa généralité et de sa relative simplicité.

2.8.5.1 Syntaxe

La syntaxe des programmes parallèles synchrones

$$\mathcal{P}ps \longrightarrow \mathcal{P}s\ [\![\ \mathcal{P}rs_0 \parallel \ldots \parallel \mathcal{P}rs_{m-1}\ ]\!];\ \mathcal{P}s' \qquad (m>1)$$

ne diffère de la syntaxe des programmes parallèles asynchrones que par le fait que les processus synchrones

$$\mathcal{P}rs \longrightarrow \mathcal{L}_0 : \mathcal{C}s_0\ ;\ \ldots\ ;\ \mathcal{L}_{m-1} : \mathcal{C}s_{m-1}\ ;\ \mathcal{L}_m : \qquad (m>1)$$

sont des listes de commandes synchrones étiquetées. Ces commandes synchrones peuvent être une commande séquentielle ou bien une opération $\underline{\mathrm{p}}$ ou $\underline{\mathrm{v}}$ sur un sémaphore. Soit $\mathcal{S}e \subseteq \mathcal{V}$ un ensemble non vide de variables sémaphores, nous avons

$$\mathcal{C}s \longrightarrow \underline{\mathrm{skip}} \mid \mathcal{V} := \mathcal{E} \mid \mathcal{V} := ? \mid \underline{\mathrm{if}}\ \mathcal{B}\ \underline{\mathrm{then}}\ \mathcal{P}rs_1\ \underline{\mathrm{else}}\ \mathcal{P}rs_2\ \underline{\mathrm{fi}} \mid$$
$$\underline{\mathrm{while}}\ \mathcal{B}\ \underline{\mathrm{do}}\ \mathcal{P}rs\ \underline{\mathrm{od}} \mid \underline{\mathrm{p}}(\mathcal{S}e) \mid \underline{\mathrm{v}}(\mathcal{S}e)$$

Comme pour les autres variables, les déclarations de sémaphores ($\underline{\mathrm{Sem}}$ Se $\underline{\mathrm{init}}$ A_0) sont laissées implicites. Nous supposerons cependant qu'elles initialisent à tout sémaphore $Se \in \mathcal{S}e$ une valeur initiale entière $\underline{\mathrm{Isem}}(Se)$ (notée A_0 ci-dessus).

Les conditions de contexte que les sémaphores n'apparaissent pas ailleurs que dans les commandes $\underline{P}$ et $\underline{V}$ et que les autres variables ne peuvent pas apparaître dans ces instructions sont habituelles :

$$([(Pps \equiv \forall \in \beta \wedge E \equiv \alpha' \forall \beta') \vee (Pps \equiv \alpha V := \beta) \vee (Pps \equiv \alpha B \beta \wedge B \equiv \alpha' \forall \beta')] \Rightarrow \forall \notin Se)$$
$$\wedge ([(Pps \equiv \forall \underline{P}(\forall) \beta) \vee (Pps \equiv \alpha \underline{V}(\forall) \beta)] \Rightarrow \forall \in Se)$$

2.8.5.2 Sémantique

De très nombreuses définitions opérationnelles de la sémantique des sémaphores ont été proposées. Des différences peuvent légères entre ces définitions entraînent parfois de subtiles différences de comportement pour les programmes qui les utilisent. La définition que nous proposons se veut fidèle à la définition originale de Dijkstra [68]. D'un point de vue pratique, on pourrait objecter que l'utilisation de stratégies de gestion des processus en attente plus souples que les files d'attente sont utiles, mais Habermann [72] montre que les sémaphores de Dijkstra avec files d'attente strictes permettent d'implanter les autres stratégies.

Dans la suite du paragraphe, nous posons $Pps \equiv Ps \,[\![Pro_0 \| \ldots \| Pro_{m-1}]\!]; Ps'$

2.8.5.2.1 Etats

La différence avec les états des programmes parallèles asynchrones (cf 2.8.2.2.1) est qu'à tout sémaphore $Se \in \mathcal{S}e \subseteq \mathcal{V}$ est associé une valeur entière et une file d'attente de processus. Pour simplifier, nous supposons $\Xi \equiv \mathbb{Z}$ et représentons la file d'attente par une séquence (éventuellement vide) d'éléments de m :

$$C[\![Pps]\!] = \{L \in \mathcal{L} : Ps \equiv \alpha L : \beta \vee Ps' \equiv \alpha L : \beta\} \cup \bigcup_{i \in m} \{L \in \mathcal{L} : Pro_i \equiv \alpha L : \beta\}$$
$$S[\![Pps]\!] = C[\![Pps]\!] \times \mathcal{M} \times (\mathcal{S}e \to m^{<\omega})$$

2.8.5.2.2 Actions

Les actions du programme parallèle synchrone $Pps \equiv Ps \,[\![Pro_0 \| \ldots \| Pro_{m-1}]\!]; Ps'$ sont :

- $\underline{P}$ correspondant à l'exécution d'un pas du prélude Ps
- $\underline{P}'$ correspondant à l'exécution d'un pas du postlude Ps'
- i correspondant à l'exécution d'un pas du processus Pro_i, $i \in m$ (autre qu'une opération sur un sémaphore)

La commande $\underline{P}(Se)$ donne lieu à deux actions possibles, à savoir la mise en attente (généralement suivie d'un réveil par un autre processus exécutant un $\underline{V}(Se)$) ou le passage du sémaphore :

- $\underline{w}(Se,i)$ correspond à l'exécution de la commande $\underline{P}(Se)$ par le processus Pro_i qui provoque la mise en attente de ce processus devant le sémaphore Se.
- $\underline{p}(Se,i)$ correspond à l'exécution de la commande $\underline{P}(Se)$ par le processus Pro_i et au passage de ce sémaphore par ce processus

La commande $\underline{V}(Se)$ donne lieu à deux actions possibles selon qu'il y avait ou non un processus en attente qu'il faut réveiller :

- $\underline{v}(Se,i)$ le processus Pro_i exécute une commande $\underline{V}(Se)$ qui libère le sémaphore alors qu'aucun autre processus n'était en attente sur ce sémaphore
- $\underline{vp}(Se,i,j)$ le processus Pro_i exécute une commande $\underline{V}(Se)$ qui libère le sémaphore et permet au processus Pro_j qui était en attente de le passer.

Nous avons donc :

$$A[\![P_{P,\underline{P}'}]\!] = \{\underline{P}, \underline{P}'\} \cup m \cup \{\underline{w}(Se,i), \underline{p}(Se,i), \underline{v}(Se,i), \underline{vp}(Se,i,j) : Se \in \mathcal{S}e \wedge i,j \in m \wedge i \neq j\}$$

2.8.5.2.3 Etats initiaux

Dans un état initial le contrôle est au début du programme. Les sémaphores $Se \in \mathcal{Se}$ ont la valeur initiale $\underline{Isem}(Se)$ (définie par une déclaration, ...) et la file d'attente associée est vide :

$$\mathcal{E}[\![Pps]\!](\Delta) = [\exists L \in \mathcal{L}, M \in \mathcal{M}, Q \in (\mathcal{Se} \to m^{<\omega}). \quad (\Delta = <L,M,Q> \wedge Pps \equiv L:\alpha \wedge \forall Se \in \mathcal{Se}. (M(Se) = \underline{Isem}(Se) \wedge Q(Se) = <>))]$$

2.8.5.2.4 Relation de transition

Pour les actions $\underline{p}$, $\underline{p}'$ et $0,\dots,m-1$ la relation de transition est similaire à celle des programmes parallèles asynchrones (cf. 2.8.2.2.4, la valeur et la file d'attente des sémaphores n'étant pas modifiées par ces actions). Il convient d'ajouter quatre cas correspondant à l'exécution d'une commande sur un sémaphore :

$$t[\![Pps]\!]_{\underline{v}(Se,i)}(\ll L_0,\dots,L_{m-1}>, M, Q>, \ll L'_0,\dots,L'_{m-1}>, M', Q'>) =$$

$$[Pps \equiv Ps[\![Pro_0 \| \dots \| Pro_{m-1}]\!]; Ps' \wedge Pro_i \equiv \alpha L_i : \underline{p}(Se); \beta \wedge M(Se) \leq 0 \wedge \forall \ell \in |Q(Se)|. Q(Se)_\ell \neq i \wedge \forall k \in m. L'_k = L_k \wedge M' = M \wedge Q'(Se) = (Q(Se)^\frown <i>) \wedge \forall Sm \in (\mathcal{Se} \vee Se). Q'(Sm) = Q(Sm)]$$

(Le processus Pro_i exécute $\underline{p}(Se)$ alors que le sémaphore a une valeur négative ou nulle et que le processus Pro_i n'est pas dans la file d'attente. Le processus Pro_i entre dans la file d'attente et le contrôle reste aux mêmes points du programme).

$$t[\![Pps]\!]_{\underline{v}(Se,i)}(\ll L_0,\dots,L_{m-1}>, M, Q>, \ll L'_0,\dots,L'_{m-1}>, M', Q'>) =$$

$$[Pps \equiv Ps[\![Pro_0 \| \dots \| Pro_{m-1}]\!]; Ps' \wedge Pro_i \equiv \alpha L_i : \underline{v}(Se); L'_i : \beta \wedge Q(Se) = <> \wedge \forall k \in (m \vee i). L'_k = L_k \wedge M'(Se) = M(Se) - 1 \wedge \forall l \in (\mathcal{Se} \vee Se). M'(l) = M(l) \wedge Q' = Q]$$

(Le processus Pro_i exécute $\underline{p}(Se)$ alors que le sémaphore a une valeur strictement positive. Le contrôle du processus passe donc la commande $\underline{p}(Se)$ et la valeur du sémaphore diminue de 1).

$$t[\![Pps]\!]_{\underline{v}(Se,i)}(\ll L_0,\dots,L_{m-1}>, M, Q>, \ll L'_0,\dots,L'_{m-1}>, M', Q'>) =$$

$$[Pps \equiv Ps[\![Pro_0 \| \dots \| Pro_{m-1}]\!]; Ps' \wedge Pro_i \equiv \alpha L_i : \underline{v}(Se); L'_i : \beta \wedge Q(Se) = <> \wedge \forall k \in (m \vee i). L'_k = L_k \wedge M'(Se) = M(Se) + 1 \wedge \forall v \in (\mathcal{Se} \vee Se). M'(v) = M(v) \wedge Q' = Q]$$

(Le processus Pro_i exécute la commande $\underline{v}(Se)$ alors qu'aucun autre processus n'est en attente sur ce sémaphore. Le contrôle de Pro_i franchit la commande et la valeur du sémaphore est augmentée de 1).

$$t[\![Pps]\!]_{\underline{vp}(Se,i,j)}(\ll L_0,\dots,L_{m-1}>, M, Q>, \ll L'_0,\dots,L'_{m-1}>, M', Q'>) =$$

$$[Pps \equiv Ps[\![Pro_0 \| \dots \| Pro_{m-1}]\!]; Ps' \wedge Pro_i \equiv \alpha L_i : \underline{v}(Se); L'_i : \beta \wedge Q(Se) = (<j>^\frown Q'(Se)) \wedge Pro_j \equiv \alpha' L'_j : \underline{p}(Se); L'_j : \beta' \wedge \forall k \in (m \vee \{i,j\}). L'_k = L_k \wedge M' = M \wedge \forall Sm \in (\mathcal{Se} \vee Se). Q'(Sm) = Q(Sm)]$$

(Le processus Pro_i exécute la commande $\underline{v}(Se)$ alors que le processus Pro_j est en tête de la file d'attente sur ce sémaphore en attente d'exécution de $\underline{p}(Se)$. De manière atomique le processus Pro_j quitte la file d'attente et les processus Pro_i et Pro_j franchissent simultanément les commandes respectives $\underline{v}(Se)$ et $\underline{p}(Se)$).

2.8.5.2.5 Traces

Dans une implantation de programmes parallèles avec sémaphores sur [illegible] monoprocesseur, le contrôleur ("scheduler") synchronisant les [illegible] sur les sémaphores a la main réquisitionnée au moyen d'interruptions [illegible], pour [illegible] aucune opération sur les sémaphores n'est exécutée

De manière plus abstraite, cette hypothèse d'équité faible se traduit par le fait que si le contrôle est devant une commande $\sharp(se)$ (auquel cas l'une des deux actions $\underline{w}(se,i)$ ou $\sharp(se,i)$ est toujours exécutable) alors fatalement le processus correspondant sera mis dans la file d'attente (par exécution de l'action $\underline{w}(se,i)$) ou franchira la commande $\sharp(se)$ (par exécution de l'action $\sharp(se,i)$). De même, si le contrôle d'un processus est devant une commande $\underline{v}(se)$ (auquel cas l'une des actions $\underline{v}(se,i)$ ou $\underline{v\sharp}(se,i,j)$, $i\neq j$ est toujours exécutable) alors fatalement le processus franchira la commande $\underline{v}(se)$ (par exécution de l'une des actions $\underline{v}(se,i)$ ou $\underline{v\sharp}(se,i,j)$).

Plus formellement, nous définissons l'ensemble réduit d'actions

$$Ar[\![Pps]\!] = \{\sharp,\sharp'\} \cup m \cup \{\sharp(se,i), \underline{v}(se,i) : se\in\mathcal{Se} \wedge i\in m\}$$

et la correspondance $fa \in (A[\![Pps]\!] \to Ar[\![Pps]\!])$ par cas :

$fa(\underline{w}(se,i)) = fa(\sharp(se,i)) = \sharp(se,i)$ si $se\in\mathcal{Se}$, $i\in m$

$fa(\underline{v}(se,i)) = fa(\underline{v\sharp}(se,i,j)) = \underline{v}(se,i)$ si $se\in\mathcal{Se}$, $i,j\in m$

$fa(x) = x$ si x est $\sharp,\sharp'$ ou $i\in m$

Les traces p de $\Sigma[\![Pps]\!]$ sont les traces engendrées par le système de transition associé à Pps (et appartenant donc à $\Sigma<S[\![Pps]\!], A[\![Pps]\!], t[\![Pps]\!], \varepsilon[\![Pps]\!]>$) dont l'image $\underline{\alpha}<fa>(p)$, après avoir renommé les actions par fa, est faiblement équitable pour $Ar[\![Pps]\!]$:

$$\Sigma[\![Pps]\!] = \{p\in\Sigma<S[\![Pps]\!], A[\![Pps]\!], t[\![Pps]\!], \varepsilon[\![Pps]\!]> : \underline{\alpha}<fa>(p) \in \underset{fair}{\mathcal{W}}<Ar[\![Pps]\!]>(\underline{\alpha}<fa>(\Sigma<S[\![Pps]\!], A[\![Pps]\!], t[\![Pps]\!], \varepsilon[\![Pps]\!]>))\}$$

2.8.5.2.6 Propriétés de la sémantique des sémaphores

Nous nous contentons de démontrer des propriétés d'invariance et d'équité des sémaphores

démonstration ou en utilisant des arguments informels.

Nous commençons par démontrer des lemmes préliminaires :

Le premier lemme exprime que si la valeur initiale du sémaphore est positive ou nulle alors il est toujours vrai en cours d'exécution du programme que si la valeur du sémaphore est positive ou nulle alors la file est vide et que si la valeur du sémaphore est nulle alors la file contient (au plus) m processus deux à deux différents :

Lemme 2.8.5.2.6 v1

$$\forall p\in\Sigma[\![Pps]\!], j\in|p|, L\in C[\![Pps]\!], M\in\mathcal{M}, \varphi\in(\mathcal{Se}\to m^{<\omega}), se\in\mathcal{Se}.$$
$$[p_j = <L,M,\varphi> \wedge \underline{Isem}(se)\geq 0]$$
$$\to [(|\varphi(se)|=0 \wedge M(se)\geq 0) \vee (0<|\varphi(se)|\leq m \wedge M(se)=0 \wedge \forall k,k'\in|\varphi(se)|.(k\neq k'\Rightarrow \varphi(se)_k\neq\varphi(se)_{k'}))]$$

Démonstration

Par induction sur $j\in|p|$. Si $j=0$ alors $\in[\![Pps]\!](p_0)$, $p_0 = <L,M,\varphi>$ et $\underline{Isem}(se)\geq 0$ entraînent $\varphi(se)=<>$ donc $|\varphi(se)|=0$ et $M(se)=\underline{Isem}(se)\geq 0$. Si la propriété est vraie pour $p_j = <L,M,\varphi>$ et $j+1\in|p|$, démontrons que par définition de $t[\![Pps]\!]_{t_j}$, la propriété reste vraie pour $p_{j+1} = <L',M',\varphi'>$. Si l'action t_j est $\sharp,\sharp'$ ou $i\in m$ c'est évident car $M'(se)=M(se)$ et $\varphi'(se)=\varphi(se)$. Si $t_j = \underline{w}(se,i)$ alors nous avons $M(se)\leq 0$ et $\forall k\in|\varphi(se)|.\varphi(se)_k\neq i$ et donc par hypothèse d'induction $0<|\varphi(se)|<m$ et $M(se)=0$ et les éléments de $\varphi(se)$ sont deux à deux différents. Comme $\varphi'(se)=\varphi(se)^\frown<i>$ et $M'(se)=M(se)$, nous en déduisons que $1\leq\varphi'(se)\leq m$, $M'(se)=0$ et les éléments de $\varphi'(se)$ sont deux à deux différents. Si $t_j = \sharp(se,i)$ alors $M(se)>0$ et donc par hypothèse d'induction $|\varphi(se)|=0$. Nous avons également $M'(se)=M(se)+1$ et $\varphi'(se)=\varphi(se)$ donc nous en déduisons $M'(se)\geq 0$ et $|\varphi'(se)|=0$. Si $t_j = \underline{v}(se,i)$ alors $|\varphi(se)|=0$, $M'(se)=M(se)+1$ et $\varphi'(se)=\varphi(se)$ et donc $|\varphi'(se)|=0 \wedge M'(se)>0$. Si $t_j = \underline{v\sharp}(se,i,k)$ alors $|\varphi(se)|>0$ et donc $M(se)=0$. Comme $M'(se)=M(se)$ et $|\varphi'(se)|<|\varphi(se)|$ nous en déduisons $0\leq|\varphi'(se)|<m$, $M'(se)=0$ et les éléments de $\varphi'(se)$ sont

deux à deux différents puisque ceux de $\varphi(se)$ le sont par hypothèse d'induction.

□

Le lemme suivant montre que si un processus est dans la file d'attente d'un sémaphore se alors son point de contrôle est devant une commande $\sharp(se)$

Lemme 2.8.5.2.6∿2

$\forall Pps \in \mathcal{P}ps,\ m \in \omega,\ p \in \Sigma[\![Pps]\!],\ \langle L_0,\dots,L_{m-1}\rangle \in \mathcal{L}^m,\ M \in \mathcal{M},\ \varphi \in (\mathcal{S}e \to m^{<\omega}),\ se \in \mathcal{S}e.$
$m \in \mathbb{m},\ j \in |p|.$
$[Pps \equiv Ps[\![Pre_0\|\dots\|Pre_{m-1}]\!];Ps' \wedge p_j = \ll L_0,\dots,L_{m-1}\rangle, M, \varphi\rangle \wedge \exists i \in |\varphi(se)|.\ \varphi(se)_i = m]$
$\Rightarrow [Pre_m \equiv \alpha L_m : \sharp(se); \beta]$

Démonstration

Par induction sur $j \in |p|$. Le lemme est vrai pour $j=0$ car $\varepsilon[\![Pps]\!](p_0)$ implique $\varphi(se)=0$. Si le lemme est vrai pour $p_j = \ll L_0,\dots,L_{m-1}\rangle, M, \varphi\rangle$ et $j+1 \in |p|$ alors il reste vrai pour $p_{j+1} = \ll L'_0,\dots,L'_{m-1}\rangle, M', \varphi'\rangle$.

- si $\forall i \in |\varphi(se)|.\ \varphi(se)_i \neq m$ et $\exists k \in |\varphi'(se)|.\ \varphi'(se)_k = m$ alors par définition de $t[\![Pps]\!]$, nous avons $\sharp_j = \underline{w}(se,m)$ et $Pre_m \equiv \alpha L_m : \sharp(se); \beta$
- si $\exists l \in |\varphi(se)|.\ \varphi(se)_l = m$ alors par hypothèse d'induction, nous avons $Pre_m \equiv \alpha L_m : \sharp(se); \beta$. Par définition de $t[\![Pps]\!]$, $\sharp_j$ n'est pas $\sharp_1$, $\sharp'$, $\underline{w}(se,m)$, $\underline{\sharp}(se,m)$ (car nous aurions $M(se)>0$ et d'après le lemme 2.8.5.2.6∿1, ceci impliquerait $\varphi(se)=\langle\rangle$), $\underline{v}(se,m)$ ou $\underline{v\sharp}(se,m,i')$ (car nous aurions $Pre_m \equiv \alpha' L_m : \underline{v}(se); \beta'$ contraire à l'hypothèse que L_m n'apparaît qu'une fois dans Pre_m). Si $\sharp_j$ est $i \in \mathbb{m}$, $\underline{w}(sm,i)$, $\underline{\sharp}(sm,i)$, $\underline{v}(sm,i)$, $\underline{v\sharp}(sm,i,i')$ avec $sm \in (\mathcal{S}e \vee Se)$, nous avons $i \neq m$ et $i' \neq m$ et donc $L'_m = L_m$ par définition de $t[\![Pps]\!]$. Appliquant l'hypothèse d'induction, nous en déduisons que $Pre_m = \alpha L'_m : \underline{\sharp}(se); \beta$. En effet $\sharp_j = \underline{v\sharp}(se,i,m)$, par définition de $t[\![Pps]\!]$ nous avons $\varphi(se)_i = m$ $\varphi'(se)_i$ et donc $\forall i \in |\varphi(se)|.\ \varphi'(se)_i \neq m$ car nous avons $\varphi'(se)_k = m$ ce qui est contraire à l'hypothèse.

Le lemme suivant exprime que le nombre d'exécutions d'une commande $\underline{\sharp}(se)$ conduisant à la mise en attente du processus qui l'exécute est égal au nombre d'exécutions d'une commande $\underline{v}(se)$ conduisant au réveil d'un processus en attente plus le nombre de processus en attente dans la file d'attente du sémaphore se :

Etant donnée une sémantique $\langle S, A, \Sigma\rangle$, nous définissons

$\sigma \in (2^A \to ((\Sigma \times \omega) \to \omega))$

$\sigma\alpha(p,j) = \sum_{k \in (j \cap |\sharp|)} (\sharp_k \in \alpha \to 1 | 0)$ (avec $\Sigma\,\phi = 0$)

de sorte que $\sigma\alpha(p,j)$ est le nombre de fois qu'une action appartenant à α est exécutée entre p_0 et p_j.

Lemme 2.8.5.2.6∿3

$\forall p \in \Sigma[\![Pps]\!],\ j \in |p|,\ L \in C[\![Pps]\!],\ M \in \mathcal{M},\ \varphi \in (\mathcal{S}e \to m^{<\omega}),\ se \in \mathcal{S}e.$
$[p_j = \langle L, M, \varphi\rangle] \Rightarrow [\sigma\{\underline{w}(se,i) : i \in \mathbb{m}\}(p,j) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in \mathbb{m}\}(p,j) + |\varphi(se)|]$

Démonstration

Par induction sur $j \in |p|$. Si $j=0$, nous avons $\varepsilon[\![Pps]\!](p_j)$ et donc $|\varphi(se)|=0$ de sorte que $\sigma\{\underline{w}(se,i) : i \in \mathbb{m}\}(p,j) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in \mathbb{m}\}(p,j) = |\varphi(se)| = 0$. Si par hypothèse d'induction la propriété est vraie pour $p_j = \langle L, M, \varphi\rangle$, $j+1 \in |p|$ et $p_{j+1} = \langle L', M', \varphi'\rangle$ alors si $\sharp_j$ est $\sharp_1$, $\sharp'$, $i \in \mathbb{m}$, $\underline{w}(sm,i)$, $\underline{\sharp}(sm,i)$, $\underline{v}(sm,i)$, $\underline{v\sharp}(sm,i,k)$ pour $sm \in (\mathcal{S}e \vee Se)$, $\underline{\sharp}(se,i)$ ou $\underline{v}(se,i)$, nous avons $\sigma\{\underline{w}(se,i) : i \in \mathbb{m}\}(p,j+1) = \sigma\{\underline{w}(se,i) : i \in \mathbb{m}\}(p,j)$, $\sigma\{\underline{v\sharp}(se,i,i') : i,i' \in \mathbb{m}\}(p,j+1) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in \mathbb{m}\}(p,j)$ et $\varphi' = \varphi$. Si $\sharp_j = \underline{w}(se,k)$ alors $\sigma\{\underline{w}(se,i) : i \in \mathbb{m}\}(p,j+1) = \sigma\{\underline{w}(se,i) : i \in \mathbb{m}\}(p,j) + 1$, $\sigma\{\underline{v\sharp}(se,i,i') : i,i' \in \mathbb{m}\}(p,j+1) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in \mathbb{m}\}(p,j)$ et $|\varphi'(se)| = |\varphi(se)| + 1$. Si $\sharp_j = \underline{v\sharp}(se,k,k')$ alors $\sigma\{\underline{w}(se,i) : i \in \mathbb{m}\}(p,j+1) = \sigma\{\underline{w}(se,i) : i \in \mathbb{m}\}(p,j)$, $\sigma\{\underline{v\sharp}(se,i,i') : i,i' \in \mathbb{m}\}(p,j+1) = \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in \mathbb{m}\}(p,j) + 1$ et $|\varphi'(se)| = |\varphi(se)| - 1$.

□

Le lemme suivant exprime que le nombre d'exécutions d'une commande $\underline{\sharp}(se)$ sans attente du processus qui l'exécute est égal au nombre d'exécutions de la commande $\underline{v}(se)$ alors qu'aucun processus n'était en attente dans la file de se plus la valeur initiale moins la valeur courante du sémaphore Se

Lemme 2.8.5.2.6∨4

$$\forall p \in \Sigma[\![Pps]\!],\ j \in |p|,\ L \in C[\![Pps]\!],\ M \in \mathcal{M},\ Q \in (\mathcal{S}e \to n^{<\omega}),\ se \in \mathcal{S}e.$$
$$[p_j = \langle L, M, Q\rangle]$$
$$\to [\sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j) + \underline{I}_{se\,m}(se) - M(se)]$$

Démonstration

Par induction sur $j \in |p|$. Pour $j=0$, nous avons $\sigma\{\underline{\sharp}(se,i) : i \in m\}(p,0) = \sigma\{\underline{v}(se,i) : i \in m\}(p,0) = 0$ et $\in [\![Pps]\!](p_0)$ entraîne que $M(se) = \underline{I}_{se\,m}(se)$. Si la propriété est vraie pour $p_j = \langle L, M, Q\rangle$, $j+1 \in |p|$ et $p_{j+1} = \langle L', M', Q'\rangle$ alors si $\sharp_j$ est $\underline{\sharp}$, $\underline{\sharp}'$, $i \in m$, $\underline{\omega}(sm,i)$, $\underline{\sharp}(sm,i)$, $\underline{v}(sm,i)$, $\underline{v\sharp}(sm,i,i')$ avec $sm \neq se$, $\underline{\omega}(se,i)$ ou $\underline{v\sharp}(se,i,i')$ alors $M' = M$ et $\sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j)$ et $\sigma\{\underline{v}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j)$. Si $\sharp_j$ est $\underline{\sharp}(se,i)$ alors $\sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j) + 1$, $\sigma\{\underline{v}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j)$ et $M'(se) = M(se) + 1$. Si $\sharp_j$ est $\underline{v}(se,i)$ alors $\sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j)$, $\sigma\{\underline{v}(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j) + 1$ et $M'(se) = M(se) + 1$ et dans les deux cas, nous déduisons de l'hypothèse d'induction que $\sigma\{p(se,i) : i \in m\}(p,j+1) = \sigma\{\underline{v}(se,i) : i \in m\}(p,j+1) + \underline{I}_{se\,m}(se) - M'(se)$.

□

Nous en déduisons le théorème d'Haberman[H2] qui s'énonce informellement comme suit :

[illegible] la commande $\underline{\sharp}(se)$ a été [illegible] est égale au minimum du nombre de fois où la commande $\underline{\omega}(se)$ a été exécutée et du nombre de fois où la commande $\underline{v}(se)$ a été

Théorème 2.8.5.2.6∨5

$$\forall p \in \Sigma[\![Pps]\!],\ j \in |p|\ se \in \mathcal{S}e.$$
$$\underline{I}_{se\,m}(se) \geq 0$$
$$\Rightarrow$$
$$\sigma\{\underline{\sharp}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) = n\{\sigma\{\underline{\omega}(se,i), \underline{\sharp}(se,i) : i \in m\}(p,j),$$
$$\sigma\{\underline{v}(se,i), \underline{v\sharp}(se,i) : i,i' \in m\}(p,j) + \underline{I}_{se\,m}(se)\}$$

Démonstration

Posons $p_j = \langle L, M, Q\rangle$. D'après le lemme 2.8.5.2.6∨1 deux cas sont à considérer :

- Si $|Q(se)| = 0$ alors $M(se) \geq 0$ et donc 2.8.5.2.6∨3 entraîne que $\sigma\{\underline{\sharp}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) = \sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j) + \sigma\{\underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) = \sigma\{\underline{\sharp}(se,i) : i \in m\}(p,j) + \sigma\{\underline{\omega}(se,i) : i \in m\}(p,j) = \sigma\{\underline{\sharp}(se,i), \underline{\omega}(se,i) : i \in m\}(p,j)$. D'autre part 2.8.5.2.6∨4 entraîne que $\sigma\{\underline{\sharp}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) \leq \sigma\{\underline{v}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) + \underline{I}_{se\,m}(se)$ et donc $\sigma\{\underline{\sharp}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j)$ est égal à l'infimum de ces deux quantités
- Si $|Q(se)| > 0$ alors $M(se) = 0$ et donc 2.8.5.2.6∨3 entraîne que $\sigma\{\underline{\sharp}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) < \sigma\{\underline{\sharp}(se,i), \underline{\omega}(se,i) : i \in m\}(p,j)$ tandisque 2.8.5.2.6∨4 entraîne que $\sigma\{\underline{\sharp}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) = \sigma\{\underline{v}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j) + \underline{I}_{se\,m}(se)$ et donc $\sigma\{\underline{\sharp}(se,i), \underline{v\sharp}(se,i,i') : i,i' \in m\}(p,j)$ est égal à l'infimum de ces deux quantités.

□

La première des propriétés d'équité des sémaphores est qu'aucun processus ne peut être bloqué devant une commande $\underline{v}(se)$. En termes d'actions et pour des traces infinies, nous avons que si une des actions $\underline{v}(se,m)$ ou $\underline{v\sharp}(se,m,i)$, $i \in (m \cup m)$, est exécutable alors une de ces actions sera fatalement exécutée.

Théorème 2.8.5.2.6 v6

$$\forall p \in \Sigma[\![Pps]\!],\ Se \in \mathcal{S}e,\ m \in m,\ i \in |p|.$$
$$\left[|p| = \omega \wedge \exists a \in S[\![Pps]\!].\left(t[\![Pps]\!]_{\underline{\nu}(se,m)}(p_i, a) \vee \exists m' \in m.\, t[\![Pps]\!]_{\underline{\psi}(se,m,m')}(p_i, a)\right)\right]$$
$$\rightarrow \left[\exists j \geq i.\left(p_j = \underline{\nu}(se,m) \vee \exists m' \in m.\, p_j = \underline{\psi}(se,m,m')\right)\right]$$

Démonstration

Si $\underline{\nu}(se,m)$ ou $\underline{\psi}(se,m,m')$ est exécutable en p_j alors par définition de $t[\![Pps]\!]$, nous avons $Pre_m \equiv \alpha L_m : \underline{\nu}(se); L'_m : \beta$. Si $\underline{\nu}(se,m)$ et $\underline{\psi}(se,m,m')$, $m' \in (m \vee m)$ ne sont jamais exécutées alors le contrôle de Pre_m reste en L_m pour tout p_j; $j \geq i$ (car par définition de $t[\![Pps]\!]$, seule l'exécution de l'une de ces actions peut le déplacer (en L'_m)). Posant $p_j = \langle\langle L_0, \ldots, L_n \rangle, M, Q\rangle$, nous observons que deux cas sont possibles :

- Si $Q(se) = \langle\rangle$ alors l'action $\underline{\nu}(se,m)$ est exécutable,
- Si $Q(se) = k\,Q'(se)$ alors d'après le lemme 2.8.5.2.6 v2, nous avons $Pre_k \equiv \alpha' L_k : \underline{p}(se); \beta'$ et donc d'après la syntaxe $\alpha' L_k : \underline{p}(se); L'_k : \beta''$ de sorte que l'action $\underline{\psi}(se,m,k)$ est exécutable.

Nous avons démontré que l'une des actions $\underline{\nu}(se,m)$ ou $\underline{\psi}(se,m,m')$, $m' \in (m \vee m)$ est toujours exécutable en p_j pour $j \geq i$. D'après l'hypothèse d'équité faible énoncée en 2.8.5.2.5 il n'est pas possible que les deux actions ne soient jamais exécutées au delà de p_i.

□

D'autre part, si la commande $\underline{p}(se)$ est exécutable infiniment souvent alors elle sera franchie. Cette propriété s'énonce plus précisément en termes d'actions et pour des traces infinies comme suit :

Si infiniment souvent une des actions $\underline{\nu}(se,m)$, $\underline{p}(se,m)$ ou $\underline{\psi}(se,i,m)$, $i \in (1,m)$, est exécutable alors une des actions $\underline{p}(se,m)$ ou $\underline{\psi}(se,i,m)$ sera [illegible] exécutée :

Théorème 2.8.5.2.6 v7

$$\forall p \in [\![Pps]\!].\ Se \in \mathcal{S}e,\ m \in m,\ i \in |p|.$$
$$\left[|p| = \omega\right]$$
$$\Rightarrow \left[\left[\forall j \geq i.\, \exists k \geq j,\, a \in S[\![Pps]\!].\left(t[\![Pps]\!]_{\underline{\nu}(se,m)}(p_k, a) \vee t[\![Pps]\!]_{\underline{p}(se,m)}(p_k, a) \vee \exists m' \in m.\, t[\![Pps]\!]_{\underline{\psi}(se,m,m)}(p_k, a)\right)\right]\right.$$
$$\left.\Rightarrow \left[\exists j \geq i.\left(p_j = \underline{p}(se,m) \vee \exists m' \in m.\, p_j = \underline{\psi}(se,m',m)\right)\right]\right]$$

Démonstration

Supposons $|p| = \omega$ et posons $p_\ell = \langle L^\ell, M^\ell, Q^\ell\rangle$ avec $L^\ell = \langle L^\ell_0, \ldots, L^\ell_{n-1}\rangle$ quand le contrôle est dans la commande parallèle de $Pps \equiv Ps[\![Pre_0 \| \ldots \| Pre_{n-1}]\!]; Ps'$. Supposons que infiniment souvent se actions $\underline{\nu}(se,m)$, $\underline{p}(se,m)$ ou $\underline{\psi}(se,m',m)$ avec $m' \in m$ sont exécutables au delà de p_i.

- Si $\exists m' \in m.\, \underline{\psi}(se,m',m)$ est exécutable (c'est-à-dire $\exists j \geq i, a' \in S[\![Pps]\!].\ t[\![Pps]\!]_{\underline{\psi}(se,m',m)}(p_j, a')$) alors il faut (d'après la définition de $t[\![Pps]\!]$) que le processus Pre_m soit en tête de la file d'attente du sémaphore se ($\varphi^i(se) = m\,\varphi'(se)$ où $a' = \langle L', M', Q'\rangle$). Si $\forall m'' \in n$, $\underline{\psi}(se,m'',m)$ n'était jamais exécuté ultérieurement et que seule cette action permet à Pre_m de sortir de la file de se, Pre_m restera toujours en tête de la file de $\varphi^k(se)$, $k \geq j$. D'autre part le théorème 2.8.5.2.6 v6 montre que $\underline{\psi}(se,m',m)$ étant exécutable en p_j, l'action $\underline{\nu}(se,m')$ ou $\underline{\psi}(se,m',m'')$ sera fatalement exécutée en p_k, $k \geq j$. Pour que $\underline{\nu}(se,m')$ puisse être exécutée il faudrait par définition de $t[\![Pps]\!]$ que la file de se soit vide ($\varphi^k(se) = \langle\rangle$) contraire au fait que Pre_m est en tête ($\varphi^k(se)_0 = m$). C'est donc $\underline{\psi}(se,m',m'')$ qui est exécuté et donc par définition de $t[\![Pps]\!]_{\underline{\psi}(se,m',m'')}$, m'' est en tête de la file de se ($\varphi^k(se)_0 = m''$) et donc $m'' = m$, ce qui montre que $\underline{\psi}(se,m',m)$ est fatalement exécuté.

- Si $\forall m' \in n.\, \underline{\psi}(se,m',m)$ n'est pas exécutable (c'est-à-dire $\forall j \geq i, a' \in S[\![Pps]\!].$ [illegible] alors infiniment souvent l'une des actions $\underline{\nu}(se,m)$ ou $\underline{p}(se,m)$ est exécutable au delà de p_i. Il existe un p_j, $j \geq i$ tel que l'une des actions $\underline{\nu}(se,m)$ ou $\underline{p}(se,m)$ est exécutable. En ce point, par définition

de $t[\![Pps]\!]_{\underline{w}(se,m)}$ et $t[\![Pps]\!]_{\underline{p}(se,m)}$ le processus $Pres_m$ n'est pas dans la file du sémaphore se ou cette file est vide, $\forall \ell \in |\varphi^i(se)|.\ \varphi^i(se)_\ell \neq m$. Deux cas sont alors possibles :

. Si $Pres_m$ entre ultérieurement dans la file d'attente du sémaphore se, c'est-à-dire $\exists k > j,\ \ell \in |\varphi^k(se)|.\ \varphi^k(se)_\ell = m$; tant que m reste dans la file d'attente, les actions $\underline{w}(se,m)$ et $\underline{p}(se,m)$ ne sont pas exécutables et comme infiniment souvent l'une d'elle doit l'être, il faut que $Pres_m$ sorte de la file d'attente de se après y être entré. Par définition de $t[\![Pps]\!]$, seule une action $\underline{wp}(se,m',m)$ peut faire sortir m de la file de se et donc une action de ce type est fatalement exécutée.

. Si $Pres_m$ n'entre jamais dans la file d'attente du sémaphore se, c'est-à-dire $\forall k > j,\ \ell \in |\varphi^k(se)|.\ \varphi^k(se)_\ell \neq m$. Par définition de $t[\![Pps]\!]$, il n'est pas possible que les actions $\underline{w}(se,m)$ et $\underline{wp}(se,m',m)$, n'en soient exécutées car elles auraient pour effet de faire entrer ou sortir le processus $Pres_m$ de la file d'attente de se. Comme $\underline{w}(se,m)$ ou $\underline{p}(se,m)$ est exécutable en p_j, nous avons $Pres_m \equiv \alpha L^i_m : \underline{p}(se); \beta$. Puisque le contrôle de $Pres_m$ se déplace de L^i_m au de delà de p_j, il faudrait qu'une action $\underline{wp}(se,m',m)$ ou $\underline{p}(se,m)$ soit exécutée. La première alternative n'est pas possible, la seconde termine la preuve. Supposons que l'action $\underline{p}(se,m)$ ne soit jamais exécutée au delà de p_j. Nous observons alors que $\forall k > j,\ Pres_m \equiv \alpha L^k_m : \underline{p}(se); \beta$ avec $L^k_m = L^i_m$. Par conséquent, $\forall k > j$, si $M^k(se) \leq 0$ alors $\underline{w}(se,m)$ est exécutable tandis que si $M^k(se) > 0$ alors $\underline{p}(se,m)$ est exécutable. Au delà de p_j, l'une des deux actions $\underline{w}(se,m)$ ou $\underline{p}(se,m)$ est toujours exécutable et d'après la définition du traces $\Sigma[\![Pps]\!]$ donnée en 2.8.5.2.5, l'une de ces actions sera fatalement exécutée. Comme il ne peut s'agir de $\underline{w}(se,m)$ (car $Pres_m$ n'entre jamais dans la file d'attente de se), il s'agit de $\underline{p}(se,m)$.

□

Les recherches actuelles sur les preuves de propriétés de fatalité de programmes parallèles avec sémaphores comme Lehman, Pnueli - Stavi [81] sont basées sur des hypothèses d'équité des sémaphores similaires à celle donnée dans les théorèmes 2.8.5.2.6∨6 et 2.8.5.2.6∨7. Ces méthodes ne peuvent pas être complètes pour la raison que certaines traces satisfaisant ces hypothèses d'équité ne sont pas des traces de $\Sigma[\![Pps]\!]$. C'est le cas en particulier parceque les propriétés d'équité énoncées dans les théorèmes 2.8.5.2.6∨6 et 2.8.5.2.6∨7 n'interdisent pas que l'attente pour passer un sémaphore ne soit pas bornée.

Au contraire, pour la sémantique de Dijkstra [68], nous avons la propriété suivante, que nous démontrons :

"Si un processus $Pres_m$ est en attente devant une commande $\underline{p}(se)$ en position r dans la file d'attente du sémaphore se, alors r+1 commandes $\underline{v}(se)$ et un nombre fini borné de commandes $\underline{p}(se)$ s'exécutent avant que l'attente ne prenne fin."

Théorème 2.8.5.2.6∨8

$$\forall Pps \in \mathcal{P}ps,\ n \in \omega,\ p \in \Sigma[\![Pps]\!],\ Q \in (|p| \to (\mathcal{L}_e \to m^{<\omega})),\ j,k \in |p|,\ se \in \mathcal{L}_e,\ r \in \omega.$$

$$\Big[Pps \equiv Ps[\![Pres_0 \| \ldots \| Pres_{m-1}]\!]; Ps' \wedge j \leq k \wedge \exists m \in n.\ \forall \ell \in |p|.\ \big((j \leq \ell \leq k) \Rightarrow (\exists < L^\ell_0, \ldots, L^\ell_{m-1} > \in \mathcal{L}^m,\ M^\ell \in (\mathcal{V} \to \mathcal{Z}).\ p_\ell \equiv \ll L^\ell_0, \ldots, L^\ell_{m-1} >, M^\ell, Q^\ell > \wedge Pres_m \equiv \alpha L^\ell_m : \underline{p}(se); L^k_m : \beta \wedge Q^\ell(se)_r = m \big) \Big]$$

$$\Rightarrow \Big[\sigma\{\underline{v}(se,i) : i \in n\}(p,k) - \sigma\{\underline{v}(se,i) : i \in n\}(p_{|j}) = 0$$

$$\wedge\ \sigma\{\underline{wp}(se,i,i') : i,i' \in n\}(p,k) - \sigma\{\underline{wp}(se,i,i') : i,i' \in n\}(p_{|j}) = r+1$$

$$\wedge\ \sigma\{\underline{p}(se,i) : i \in n\}(p,k) - \sigma\{\underline{v}(se,i) : i \in n\}(p_{|j}) = 0$$

$$\wedge\ \sigma\{\underline{w}(se,i) : i \in n\}(p,k) - \sigma\{\underline{w}(se,i) : i \in n\}(p_{|j}) \leq n - |\varphi^j(se)| + r \Big]$$

Démonstration

Nous construisons $R \in (|Pp| \to \omega)$ tel que $R_j = r$ et $\forall \ell \in |p|.\ j \le \ell \le k$ nous aurons $\varphi^\ell(Se)_{R_\ell} = m$, $\sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,\ell) - \sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,j) = R_j - R_\ell$, $\sigma\{\underline{\psi}(se,i) : i \in m\}(p,\ell) - \sigma\{\underline{\psi}(se,i) : i \in m\}(p,j) = 0$ et $\sigma\{\underline{\underline{\psi}}(se,i) : i \in m\}(p,\ell) - \sigma\{\underline{\underline{\psi}}(se,i) : i \in m\}(p,j) = 0$

Par hypothèse, nous avons $\varphi^j(Se)_r = m$ de sorte que les propriétés ci-dessus sont vraies pour $j = \ell$ en posant $R_j = r$. Ayant construit R_ℓ pour $\ell < k$, nous définissons $R_{\ell+1}$ comme suit : si $\underline{\underline{\psi}}_\ell = \underline{\underline{\psi}}(se,i,i')$ alors par définition de $t[\![Pps]\!]_{\underline{\underline{\psi}}(se,i,i')}$ nous avons $\varphi^\ell(Se) = i' \varphi^{\ell+1}(Se)$ avec $Pres_{i'} \equiv \alpha' L^\ell_{i'} : \underline{\underline{\psi}}(se) ; L^{\ell+1}_{i'} : \beta'$. D'après l'hypothèse que les étiquettes ne figurent qu'une seule fois dans le programme Pps, nous avons $L^\ell_{i'} \neq L^{\ell+1}_{i'}$ et donc $m \neq i'$ car $L^\ell_m = L^{\ell+1}_m$. Posons $R_{\ell+1} = R_\ell - 1$ et donc $\varphi^{\ell+1}(Se)_{R_{\ell+1}} = \varphi^\ell(Se)_{R_{\ell+1}+1} = \varphi^\ell(Se)_{R_\ell} = m$. De plus $\sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,\ell+1) = \sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,\ell) + 1$ et donc $\sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,\ell+1) - \sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,j) = R_j - R_{\ell+1} = R_j - R_{\ell+1}$. Nous ne pouvons pas avoir $\underline{\underline{\psi}}_\ell = \underline{\psi}(se,i)$ (car il faudrait $\varphi^\ell(Se) = \langle\rangle$) et donc $\sigma\{\underline{\psi}(se,i) : i \in m\}(p,\ell+1) = \sigma\{\underline{\psi}(se,i) : i \in m\}(p,\ell)$. De même, nous ne pouvons pas avoir $\underline{\underline{\psi}}_\ell = \underline{\underline{\psi}}(se,i)$ (car il faudrait $M^\ell(Se) > 0$ et donc d'après le lemme 2.8.5.2.6 n°1, nous aurions $|\varphi(Se)| = 0$ contraire à $\varphi^\ell(Se)_{R_\ell} = m$) et donc $\sigma\{\underline{\underline{\psi}}(se,i) : i \in m\}(p,\ell+1) = \sigma\{\underline{\underline{\psi}}(se,i) : i \in m\}(p,\ell)$. Si $\underline{\underline{\psi}}_\ell$ est $\underline{\omega}(se,i)$, $\underline{\omega}(sm,i)$, $\underline{\underline{\psi}}(sm,i)$, $\underline{\underline{\psi}}(sm,i)$, $\underline{\underline{\psi}}(sm,i,i')$ avec $sm \in (\mathcal{S}e \cup Se)$ nous avons $\varphi^{\ell+1}(Se) = \varphi^\ell(Se)$ par définition de $t[\![Pps]\!]$ et la propriété reste vraie en posant $R_{\ell+1} = R_\ell$.

Observons que pour $\ell = k-1$, nous avons $\underline{\underline{\psi}}_{k-1} = \underline{\underline{\psi}}(se,i,m)$. En effet par définition de $t[\![Pps]\!]$ et le fait que $Pres_m \equiv \alpha L^{k-1}_m : \underline{\underline{\psi}}(se) ; L^k_m : \beta$ nous ne pouvons avoir que $\underline{\underline{\psi}}_{k-1} \in \{\underline{\underline{\psi}}(se,m), \underline{\underline{\psi}}(se,i,m)\}$. Mais $\underline{\underline{\psi}}_{k-1} = \underline{\underline{\psi}}(se,m)$ est impossible car il faudrait $M^{k-1}(se) > 0$ et donc d'après 2.8.5.2.6 n°1 $\varphi^{k-1}(Se) = \langle\rangle$ en contradiction avec $\varphi^{k-1}(Se)_{R_{k-1}} = m$. Par définition de $t[\![Pps]\!]_{\underline{\underline{\psi}}(se,i,m)}$ et le lemme [illegible] $\sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,j) = \sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,k-1) + 1 - \sigma\{\underline{\underline{\psi}}(se,i,i') : i,i' \in m\}(p,\ell)$

Observons qu'entre p_j et p_{R-1} (et donc p_R) les processus qui ont pu exécuter la commande $\underline{\underline{\psi}}(Se)$ sont ceux qui n'étaient pas dans la file d'attente $\varphi^j(Se)$ en p_j ainsi que ceux qui étaient devant $Pres_m$ dans la file d'attente $\varphi^j(Se)$ et qui sont sortis de cette file avant $Pres_m$. Nous avons vu que l'exécution d'une telle commande ne peut correspondre qu'à l'action $\underline{\omega}(se,i)$ qui a pour effet de ranger le processus $Pres_i$ après $Pres_m$ dans la file d'attente de Se. Comme $Pres_m$ ne sort de la file qu'en p_R le processus ne peut rentrer dans la file qu'au plus une fois, nous en déduisons $\sigma\{\underline{\omega}(se,i) : i \in m\}(p,R) - \sigma\{\underline{\omega}(se,i) : i \in m\}(p,j) = m - |\varphi^j(Se)| + r$.

□

2.8.5.3 Sémantique libérale

Pour démontrer certaines propriétés des programmes parallèles avec sémaphores, nous pourrons quelquefois utiliser une sémantique libérale qui ne prend pas en compte la file d'attente et correspond à une attente active.

$$Sl[\![Pps]\!] = C[\![Pps]\!] \times \mathcal{M}$$

$$El[\![Pps]\!] = [\exists L \in \mathcal{L}, M \in \mathcal{M}.\ (\Delta = \langle L, M\rangle \wedge Pps \equiv L : \alpha \wedge \forall Se \in \mathcal{S}e.\ M(Se) = \underline{\underline{\mathrm{sem}}}(se))]$$

$$tl[\![Pps]\!]_{\underline{\omega}(se,i)}(\langle\langle L_0,\dots,L_{m-1}\rangle, M\rangle, \langle\langle L'_0,\dots,L'_{m-1}\rangle, M'\rangle) =$$
$$[Pps \equiv Ps[\![Pres_0 \| \dots \| Pres_{m-1} \|]\!] Ps' \wedge Pres_i \equiv \alpha L_i : \underline{\underline{\psi}}(se) ; \beta \wedge M(Se) \le 0 \wedge \forall k \in m.\ L'_k = L_k \wedge M' = M]$$

$$tl[\![Pps]\!]_{\underline{\underline{\psi}}(se,i)}(\langle\langle L_0,\dots,L_{m-1}\rangle, M\rangle, \langle\langle L'_0,\dots,L'_{m-1}\rangle, M'\rangle) =$$
$$[Pps \equiv Ps[\![Pres_0 \| \dots \| Pres_{m-1} \|]\!] Ps' \wedge Pres_i \equiv \alpha L_i : \underline{\underline{\psi}}(se) ; L'_i : \beta \wedge M(Se) > 0 \wedge \forall k \in (m+1).\ L'_k = L_k \wedge M'(Se) = M(Se) - 1 \wedge \forall V \in (\mathcal{V} \cup Se).\ M'(V) = M(V)]$$

$$tl[\![Pps]\!]_{\underline{v}(se,i)}(\langle\langle L_0,\dots,L_{m-1}\rangle,M\rangle,\langle\langle L'_0,\dots,L'_{m-1}\rangle,M'\rangle) =$$
$$[Pps \equiv Ps[\![Pra_0\|\dots\|Pra_{m-1}]\!];Ps' \wedge Pra_i \equiv \alpha L_i:\underline{v}(se);L'_i:\beta \wedge$$
$$\forall k\in(m\setminus i).L'_k=L_k \wedge M'(se)=M(se)+1 \wedge \forall v\in(\mathcal{V}\setminus se).M'(v)=M(v)]$$

$$tl[\![Pps]\!]_{\underline{vf}(se,i,j)}(\langle\langle L_0,\dots,L_{m-1}\rangle,M\rangle,\langle\langle L'_0,\dots,L'_{m-1}\rangle,M'\rangle) =$$
$$[Pps \equiv Ps[\![Pra_0\|\dots\|Pra_{m-1}]\!];Ps' \wedge Pra_i \equiv \alpha L_i:\underline{v}(se);L'_i:\beta \wedge$$
$$Pra_j \equiv \alpha' L_j:\underline{f}(se);L'_j:\beta' \wedge \forall k\in m\setminus\{i,j\}.L'_k=L_k \wedge M'=M]$$

Cette sémantique étant utilisée pour raisonner sur des ensembles d'états accessibles, nous remarquons que l'action d'attente $\underline{w}(se,i)$ peut être supprimée puisqu'elle ne change pas l'état courant. Nous observons également que si l'exécution de l'action $\underline{vf}(se,i,j)$ conduit de l'état s à l'état s' alors il est également possible que les actions $\underline{v}(se,i)$ puis $\underline{f}(se,j)$ soient exécutées, ce qui conduirait également de s à s'. Pour raisonner sur l'ensemble des états accessibles par la sémantique libérale, les actions $\underline{w}(se,i)$ et $\underline{vf}(se,i,j)$ peuvent donc être supprimées :

$$Al[\![Pps]\!] = \{\underline{f},\underline{f}'\} \cup m \cup \{\underline{f}(se,i),\underline{v}(se,i) : se\in\mathcal{Se} \wedge i\in m\}$$

Aucune propriété d'équité n'étant à prendre en compte dans cette sémantique libérale, nous définissons :

$$\Sigma l[\![Pps]\!] = \Sigma\langle Sl[\![Pps]\!], Al[\![Pps]\!], tl[\![Pps]\!], el[\![Pps]\!]\rangle$$

La propriété évidente de cette sémantique est qu'à une fonction d'un état près, l'ensemble des états accessibles d'après la sémantique (exacte) est inclus dans l'ensemble des états accessibles d'après la sémantique libérale.

Première 2.8.5.3 ∨1

Si $\underline{fl} \in (S[\![Pps]\!] \to Sl[\![Pps]\!])$ est défini par $\underline{fl}(\langle L,M,\Phi\rangle) = \langle L,M\rangle$

alors $\{s\in Sl[\![Pps]\!] : \exists p\in v\langle fl\rangle(\Sigma[\![Pps]\!]), i\in|p| . p_i = s\}$
$\subseteq \{s\in Sl[\![Pps]\!] : \exists p\in \Sigma l[\![Pps]\!], i\in|p| . p_i = s\}$

2.8.5.4 Exemple

Le programme parallèle suivant réalise une section critique :

```
sem Se init 1;
0:
   [[10:   while true do
   11:
   12:         f(Se);
   13:         v(Se);
   14:     od;
   ||
   20:   while true do
   21:
   22:         f(Se);
   23:         v(Se);
   24:     od;
   ]];
1:
```

La sémantique exacte est bien plus complexe que la sémantique libérale [illegible] sur le plan mathématique. La sémantique [illegible] sémantique mais aussi la sémantique [illegible]

2.9 REFERENCES

ABRAHAMSON K. [80], "Expressiveness and decidability of logics of procems", Ph.D. Thesis, Univ. of Washington, Seattle, USA, (1980).

COUSOT P. [78], "Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes", Thèse d'Etat, USMG Grenoble, (1978).

COUSOT P. [79], "Analysis of the behavior of dynamic discrete systems", Rapport de Recherche n°161, IMAG, USMG Grenoble, (Jan. 1979).

COUSOT P. [81], "Semantic foundations of program analysis", dans "Program flow analysis, theory and applications", S.S. Muchnick & N.J Jones (Eds.), Prentice-Hall, (1981), 303-342.

COUSOT P., COUSOT R. [79], "A constructive characterization of the lattices of all retractions, preclosure, quasi-closure and closure operators on a complete lattice", Portugaliae Mathematica, Vol.38, Fasc. 1-2, (1979), 185-198.

DIJKSTRA E.W.D. [68], "Cooperating sequential processes", dans "Programming Languages", F. Genuys (Ed.), Academic Press, N.Y., (1968), 43-112.

EMERSON E.A. [81], "Alternative semantics for temporal logics", Research Report TR-182, Dept. of C. Sci., U. of Texas at Austin, (Oct. 1981), 16 p.

HAGERMANN A.N. [72], "Synchronization of communicating processes", CACM 15, 3 (1972), 171-176.

HOARE C.A.R. [78], "Communicating sequential processes", CACM 21, 8 (1978), 666-677.

KELLER R.M. [76], "Formal verification of parallel programs", CACM 19, 7 (1976), 371-384.

LAMPORT L. [80], "'Sometime' is sometimes 'not never'", 7th Annual ACM Symp. on Principles of Programming Languages, (1980), 174-185.

LEHMANN D., PNUELI A., STAVI J. [81], "Impartiality, justice and fairness: the ethics of concurrent termination", Proc. 8th Coll. on Automata, Languages and Programming, Lect. Notes in Comp. Sci. 115, Springer Verlag, (1981), 264-277.

MILNE R., STRACHEY C. [76], "A theory of programming language semantics", Chapman & Hall (Londres) & Wiley (New York), (1976).

PRATT V.R. [79], "Process logic", Proc. 6th ACM symp. on Principles of Programming Languages, (1979), 93-100.

3. PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

3. PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

3. PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

3.1 SPECIFICATION DE PROGRAMMES

Une preuve de programme consiste à démontrer une relation entre une sémantique (définissant "ce que fait l'exécution du programme") et une spécification (définissant "ce que devrait faire l'exécution du programme").

Pour qu'une étude des méthodes de preuve ne dépende pas des techniques choisies pour spécifier les programmes il faut donner une définition abstraite et générale de la spécification de programmes. Nous proposons de définir la spécification d'un programme Pr comme étant une sémantique $\langle S, A, \Sigma \rangle$. De cette façon une preuve de programme consiste à démontrer qu'une relation est vraie entre deux sémantiques : la spécification $\langle S, A, \Sigma \rangle$ et la sémantique opérationnelle $\langle S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!] \rangle$.

Exemple 3.1-1

Etant donnés des ensembles S d'états et A d'actions et une assertion $\phi \in (S \to \{tt, ff\})$ sur les états, l'ensemble des traces pour lesquelles ϕ est tout le temps vraie en cours d'exécution est $\Sigma = \{p \in \Sigma \langle S, A \rangle : \forall i \in |p| . \phi(i)\}$.

(i) La preuve de l'invariance de ϕ pour un programme Pr consiste à démontrer que $\langle S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!] \rangle \subseteq \langle S, A, \Sigma \rangle$ c'est-à-dire essentiellement $\forall p \in \Sigma[\![Pr]\!], i \in |p| . \phi(p_i)$.

(ii) La preuve de préservation de ϕ pour un programme Pr consiste à démontrer que $S[\![Pr]\!] \subseteq S$, $A[\![Pr]\!] \subseteq A$ et $\Sigma[\![Pr]\!] \cap \Sigma \neq \emptyset$ c'est-à-dire essentiellement $\exists p \in \Sigma[\![Pr]\!] . \forall i \in |p| . \phi(p_i)$.

Etant donnés des ensembles S d'états et A d'actions et une assertion $\psi \in (S \to \{tt, ff\})$ sur les états, l'ensemble des traces pour lesquelles ψ est inévitablement vraie en cours d'exécution est $\Sigma = \{p \in \Sigma\langle S,A\rangle : \exists i \in |p| . \psi(p_i)\}$.

(3) La preuve de fatalité de ψ pour un programme Pr consiste à démontrer que $\langle S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!]\rangle \subseteq \langle S,A,\Sigma\rangle$ c'est-à-dire eventuellement $\forall p \in \Sigma[\![Pr]\!] . \exists i \in |p| . \psi(p_i)$.

(4) La preuve de possibilité de ψ pour un programme Pr consiste à démontrer que $S[\![Pr]\!] \subseteq S$, $A[\![Pr]\!] \subseteq A$ et $\Sigma[\![Pr]\!] \cap \Sigma \neq \emptyset$ c'est-à-dire eventuellement $\exists p \in \Sigma[\![Pr]\!], i \in |p| . \psi(p_i)$.

□

Les spécifications de programmes les plus souvent utilisées concernent les propriétés d'invariance et de fatalité dont nous donnons maintenant des exemples.

3.2 INVARIANCE

Nous donnons une définition de l'invariance conditionnelle qui généralise une notion introduite par Lamport[80]. Nous obtenons comme cas particulier la notion classique d'invariance dont la correction partielle est un cas particulier. Nous distinguons l'invariance relationnelle qui permet d'exprimer une relation entre un état initial et un état courant sur une trace et l'invariance assertionnelle qui permet d'exprimer une assertion sur l'état courant d'une trace de la sémantique du programme. Nous donnons les définitions et quelques exemples

3.2.1 INVARIANCE CONDITIONNELLE

Soient $\langle S,A,\Sigma\rangle$ une sémantique opérationnelle et $\phi, \psi \in (S \times S \to \{tt, ff\})$ des relations entre états. ψ est invariante sous condition ϕ pour $\langle S,A,\Sigma\rangle$ si et seulement si

$$\forall p \in \Sigma, i \in |p| . [\forall j \in i . \phi(p_0, p_j)] \to \psi(p_0, p_i)$$

Exemple

Considérons un programme parallèle asynchrone $Ppa \equiv Ps[\![Pra_0 \| Pra_1]\!]; Ps'$. L'assertion $\underline{Terminé}_i(s) = [\exists L_0, L_1 \in \mathcal{L}, M \in \mathcal{M} . s = \langle\langle L_0, L_1\rangle, M\rangle \wedge Pra_i \equiv \alpha L_i :]$ exprime que l'exécution du processus Pra_i, $i = 0,1$ s'est terminée correctement. L'invariance de $\psi(s,s') = \neg \underline{Terminé}_1(s')$ sous condition $\phi(s,s') = \neg \underline{Terminé}_0(s')$ pour Ppa exprime que tant que l'exécution du processus Pra_0 n'est pas terminée correctement, l'exécution du processus Pra_1 ne peut pas se terminer.

□

Nous pouvons imaginer de très nombreuses variantes de cette définition comme celle-ci :

$\Psi \in (S \times S \to \{tt, ff\})$ est *invariante sous condition* $\Phi \in (S \times S \to \{tt, ff\})$ à partir de $\varepsilon \in (S \to \{tt, ff\})$ si et seulement si

$$\forall p \in \Sigma,\ j \in |\mathbb{N}|,\ i \in j.\ [\varepsilon(p_i) \wedge \forall k \in (j \dots i).\ \Phi(p_k, p_{k+1})] \Rightarrow \Psi(p_i, p_{j+1})$$

Exemple

Un certain nombre de propriétés du schéma producteur-consommateur suivant peuvent s'exprimer sous la forme ci-dessus :

```
0: [[ 10:
        while true do
     11:
        produire(PX);
     12:
        c! PX ;
     13:
        od;
     14:
   ||
     20:
        while true do
     21:
        c? CX;
     22:
        consommer(CX);
     23:
        od;
     24:
   ]];
   1:
```

Un message est produit dans l'état $\langle\langle C1, C2\rangle, M\rangle$ si $C1 = 11$ et l'état successeur est de la forme $\langle\langle 12, C2'\rangle, M'\rangle$:

$$\pi(\langle\langle C1, C2\rangle, M\rangle, \langle\langle C1', C2'\rangle, M'\rangle) = [C1 = 11 \wedge C1' = 12]$$

Un message est consommé dans l'état $\langle\langle C1, C2\rangle, M\rangle$ si $C2 = 22$ et l'état successeur est de la forme $\langle\langle C1', 23\rangle, M'\rangle$:

$$\gamma(\langle\langle C1, C2\rangle, M\rangle, \langle\langle C1', C2'\rangle, M'\rangle) = [C2 = 22 \wedge C2' = 23]$$

En choisissant $\varepsilon(s) = tt$, $\Phi = \pi$ et $\Psi = \gamma$ dans la formule ci-dessus, nous exprimons qu'à partir du moment où l'exécution du programme est commencée, il n'est pas possible de consommer tant qu'il n'y a pas eu production.

En choisissant $\varepsilon(\langle\langle C1, C2\rangle, M\rangle) = [C2 = 23]$, $\Phi = \pi$ et $\Psi = \gamma$ dans la formule ci-dessus, nous exprimons qu'après une consommation il n'est pas possible de consommer à nouveau sans qu'il y ait eu production entre temps.

□

3.2.2 INVARIANCE RELATIONNELLE

Le cas particulier où $\Phi(s, s') = tt$ correspond à l'invariance relationnelle.

$\Psi \in (S \times S \to \{tt, ff\})$ est *invariante* pour $\langle S, A, \Sigma\rangle$ si et seulement si

$$\forall p \in \Sigma,\ i \in |\mathbb{N}|.\ \Psi(p_0, p_i)$$

Exemple

La *correction partielle* est une propriété d'invariance relationnelle. Etant données une spécification $\Phi \in (\mathcal{M} \to \{tt, ff\})$ de l'état mémoire d'entrée et une spécification de sortie $\Psi \in (\mathcal{M} \times \mathcal{M} \to \{tt, ff\})$ liant l'état mémoire final à l'état mémoire initial, la correction partielle d'un programme $Pr \in \mathcal{P}r$ pour Φ, Ψ exprime que si l'exécution commence dans un état M satisfaisant Φ et atteint un état de sortie M' alors Ψ lie M et M'. Autrement dit

$$\Psi(s, s') = \big([\exists L, L', M, M'.\ s = \langle L, M\rangle \wedge \Phi(M) \wedge s' = \langle L', M'\rangle \wedge Pr \equiv \alpha L' :] \Rightarrow \Psi(M, M')\big)$$

c'est-à-dire qu'il est toujours vrai en cours d'exécution que si l'état initial satisfait Φ et l'état courant est un état final alors Ψ lie l'état final à l'état initial.

□

3.2.3 INVARIANCE ASSERTIONNELLE

Le cas particulier de l'invariance conditionnelle où $\phi(s,s') = tt$ et $\psi(s,s') = \psi(s')$ correspond à l'invariance assertionnelle.

$\psi \in (S \to \{tt, ff\})$ est invariante pour $\langle S, A, \Sigma \rangle$ si et seulement si $\forall p \in \Sigma, i \in |p|. \psi(p_i)$

Exemple

L'exclusion mutuelle est une propriété d'invariance assertionnelle. Par exemple, nous exprimons que les deux processus du programme 2.8.5.4 ne peuvent jamais être simultanément en section critique par l'invariance de

$$\psi(s) = [\exists L_0, L_1, M.\ s = \langle\langle L_0, L_1\rangle, M\rangle \wedge \neg(L_0 = 12 \wedge L_1 = 22)]$$

□

La non-terminaison, l'absence d'erreurs à l'exécution, l'absence d'interblocages globaux permanents sont d'autres exemples de propriétés de programmes qui peuvent s'exprimer par l'invariance assertionnelle.

3.3 FATALITE

Nous donnons la définition de la fatalité sous invariance, Gabbay-Pnueli-Shelah-Stavi [80], avec les cas particuliers de fatalité relationnelle et fatalité assertionnelle.

3.3.1 FATALITE SOUS INVARIANCE

Soient $\langle S, A, \Sigma \rangle$ une sémantique opérationnelle et $\phi, \psi \in (S \times S \to \{tt, ff\})$ des relations entre états.

$\psi \in (S \times S \to \{tt, ff\})$ est fatale sous invariance de ϕ pour $\langle S, A, \Sigma \rangle$ si et seulement si

$$\forall p \in \Sigma.\ \exists i \in |p|.\ [\forall j \in i.\ \phi(p_0, p_j) \wedge \psi(p_0, p_i)]$$

Exemple

Nous pouvons exprimer que la valeur d'une variable entière d'un programme reste strictement positive ou négative avant d'atteindre la valeur zéro par la fatalité de

$$\psi(s,s') = [\exists c', M'.\ s' = \langle c', M'\rangle \wedge M'(x) = 0]$$

sous invariance de

$$\phi(s,s') = [\exists c, c', M, M'.\ s = \langle c, M\rangle \wedge s' = \langle c', M'\rangle \wedge M(x) \times M'(x) \geq 0]$$

□

3.3.2 FATALITE RELATIONNELLE

La fatalité relationnelle correspond au cas particulier de la fatalité sous invariance avec $\phi(s,s') = tt$.

$\psi \in (S \times S \to \{tt, ff\})$ est fatale pour $\langle S, A, \Sigma \rangle$ si et seulement si

$$\forall p \in \Sigma.\ \exists i \in |p|.\ \psi(p_0, p_i)$$

Exemple

La correction totale est une propriété de fatalité relationnelle. Etant données une spécification d'entrée $\Phi \in (\mathcal{M} \to \{tt, ff\})$ décrivant l'état mémoire initial et une spécification de sortie $\Psi \in (\mathcal{M} \times \mathcal{M} \to \{tt, ff\})$ liant l'état mémoire final à l'état mémoire initial, la correction totale d'un programme $Pr \in \mathcal{P}r$ s'exprime par la fatalité de

$$\psi(A, A') = ([\exists L, M.\ A = \langle L, M \rangle \wedge \Phi(M)] \Rightarrow [\exists L, M, L', M'.\ A = \langle L, M \rangle \wedge A' = \langle L', M' \rangle \wedge Pr \equiv \alpha L' : \wedge \Psi(M, M')])$$

pour la sémantique $\langle s[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!] \rangle$.

□

3.3.3 FATALITE ASSERTIONNELLE

La fatalité assertionnelle correspond au cas particulier de la fatalité sous invariance avec $\phi(A, A') = tt$ et $\psi(A, A') = \Psi(A')$.

$\psi \in (S \to \{tt, ff\})$ est fatale pour $\langle s, A, \Sigma \rangle$ si et seulement si

$$\forall p \in \Sigma.\ \exists i \in |p|.\ \psi(p_i)$$

Exemple

L'absence de famine est un exemple de fatalité assertionnelle. Par exemple, nous exprimons que le premier processus du programme 2.8.7.4 rentre fatalement en section critique s'il en fait la demande par la fatalité de

$$\psi(A) = [\exists L_0, L_1, M.\ (A = \langle\langle L_0, L_1 \rangle, M \rangle \wedge L_0 = 12)]$$

pour la sémantique

$$\langle S, A, \{p \in \underset{\sim}{Suff}(\Sigma) : \exists L_0, L_1, M.\ (A = \langle\langle L_0, L_1 \rangle, M \rangle \wedge L_0 = 11)\} \rangle$$

□

La terminaison, la garantie de réponse à un signal peuvent également s'...

4. PREUVES D'INVARIANCE

4. PREUVES D'INVARIANCE

4.1 RELATIONS ENTRE SEMANTIQUES CONSERVANT L'INVARIANCE

4.1.1 CONSERVATION DE PROPRIETES D'INVARIANCE POUR DES SEMANTIQUES CONCORDANTES A DES RELATIONS ENTRE ETATS PRES

4.1.2 CONSERVATION DE PROPRIETES D'INVARIANCE APRES REDUCTION DES ETATS INOBSERVABLES

4.1.3 CONSERVATION DE PROPRIETES D'INVARIANCE PAR RETRACTION DE LA SEMANTIQUE PAR TRANSITIONS

4.2 PRINCIPES D'INDUCTION

4.2.1 PRINCIPES D'INDUCTION POUR LES SEMANTIQUES CLOSES

4.2.1.1 Principe d'induction de base pour l'invariance

4.2.1.2 Transformations de principes d'induction

4.2.1.2.1 Transformation par distinction/confusion des états initiaux ou finaux

4.2.1.2.2 Transformation par déduction/prédiction

4.2.1.2.3 Transformation par inversion

4.2.1.2.4 Transformation contrapositive

4.2.1.2.5 Transformation relation/assertion

4.2.1.3 Principes d'induction dérivés par transformations

4.2.1.4 Equivalence forte des principes d'induction dérivés

4. PREUVES D'INVARIANCE

Floyd [67] et Naur [66] sont souvent cités comme étant à l'origine des preuves de correction partielle des programmes déterministes séquentiels (bien que l'idée remonte certainement aux origines de la programmation (Turing, Von Neumann). Hoare [68] introduisit la présentation des preuves par induction sur la syntaxe des programmes. La méthode de Morris-Wegbreit [77] (dite "subgoal induction") a montré beaucoup plus tard qu'il n'y a pas qu'une seule manière de faire des preuves d'invariance. La généralisation au cas des programmes parallèles conduisit à une profusion de méthodes qu'il est bien difficile de comparer ne serait-ce que parceque les langages de programmation sont différents (Ashcroft [75], Ashcroft-Manna [70], Hoare [75], Howard [76], Keller [76], Lamport [77], Mazurkiewicz [77], Newton [75], Owicki-Gries [76a], [76b], etc.).

Le but de notre travail est de présenter un modèle abstrait pour étudier les méthodes de preuve d'invariance. Il s'agit d'en formaliser l'essence à l'aide de principes d'induction, d'en étudier la correction et la complétude relativement à une sémantique, de les comparer notamment du point de vue de l'équivalence forte et de proposer une méthode de construction systématique d'une méthode de preuve d'invariance à partir d'une définition de la sémantique opérationnelle.

4.1 RELATIONS ENTRE SEMANTIQUES CONSERVANT L'INVARIANCE

Pour démontrer une propriété d'un programme relativement à une sémantique, on cherche souvent à se ramener à une sémantique plus simple conservant la propriété à démontrer.

La relation entre ces deux sémantiques est souvent exprimée indirectement, par exemple elle est induite par une transformation du programme :

- Un exemple très courant est celui de la compilation. Pour éviter d'avoir à prendre en compte la compilation dans une preuve d'invariance, on ne raisonne jamais sur la sémantique du code objet mais toujours sur une sémantique du code source. Cette démarche est implicitement basée sur une hypothèse de correction du compilateur que nous pouvons formuler en disant qu'après réduction des états inobservables, les sémantiques source et objet sont concordantes à une relation entre états et actions près.

- Un autre exemple est fourni par l'utilisation que font ouvrich Gries [76a] de variables auxiliaires dans les preuves de programmes : une preuve de correction d'un programme P se fait en raisonnant sur un programme transformé P' qui utilise un ensemble VA de variables dites auxiliaires qui n'apparaissent que dans des commandes d'affectation $X := E$ telles que $X \in VA$, et tel que P s'obtient à partir de P' en enlevant toutes les commandes d'affectation à ces variables auxiliaires. En définissant les états inobservables comme ceux dont l'état de contrôle désigne une commande d'affectation à une variable auxiliaire, on observe qu'après réduction des états inobservables, les sémantiques de P' et P sont concordantes à une fonction des états près

4.1.1 CONSERVATION DE PROPRIETES D'INVARIANCE POUR DES SEMANTIQUES CONCORDANTES A DES RELATIONS ENTRE ETATS PRES

Théorème 4.1.1 n°1

Si $\simeq \langle rs, ra\rangle (\langle S, A, \Sigma\rangle, \langle S', A', \Sigma'\rangle)$ alors

$[\psi$ est invariante pour $\langle S, A, \Sigma\rangle] \Rightarrow \not\Leftarrow [rs^{-1} \circ \psi \circ rs$ est invariante pour $\langle S', A', \Sigma'\rangle]$

Démonstration

($\Rightarrow$) Si ψ est invariante pour $\langle S, A, \Sigma\rangle$, $p' \in \Sigma'$ et $i \in |p'|$ alors il existe $p \in \Sigma$ tel que $|p| = |p'|$, $rs(p_0, p'_0)$ et $rs(p_i, p'_i)$. Par conséquent $\psi(p_0, p_i)$ entraine $rs^{-1} \circ \psi \circ rs(p'_0, p'_i)$.

($\not\Leftarrow$) Choisir $S = \{0,1\}$, $A = \phi$, $\Sigma = \{0,1\}$, $S' = \{0'\}$, $\Sigma' = \{0'\}$, $rs(0,0')$, $rs(1,0')$, $\psi(0,0)$, $\neg\psi(1,1)$. Nous avons $rs^{-1} \circ \psi \circ rs(0',0')$ et donc $rs^{-1} \circ \psi \circ rs$ est invariante pour Σ' mais ψ n'est pas invariante pour Σ.

□

La réciproque est vraie si nous ajoutons une condition supplémentaire :

Théorème 4.1.1 n°2

Si

$\simeq \langle rs, ra\rangle (\langle S, A, \Sigma\rangle, \langle S', A', \Sigma'\rangle)$ (1)

$\wedge$

$(rs^{-1} \circ \psi \circ rs(s'_1, s'_2) \wedge rs^{-1}(A'_1, A_1) \wedge rs(A_2, A'_2)) \Rightarrow \psi(s_1, s_2)$ (2)

alors

$[\psi$ est invariante pour $\langle S, A, \Sigma\rangle] \leftrightarrow [rs^{-1} \circ \psi \circ rs$ est invariante pour $\langle S', A', \Sigma'\rangle]$

Démonstration

($\Rightarrow$) La même que pour 4.1.1v1. ($\Leftarrow$) Si $ra^{-1} \circ \psi \circ ra$ est invariante pour $\langle s', A', \Sigma' \rangle$, $p \in \Sigma$ et $i \in |p|$ alors il existe $p' \in \Sigma'$ tel que $ra(p_0, p'_0)$ et $ra(p_i, p'_i)$ - d'après (1). Par conséquent, $ra^{-1} \circ \psi \circ ra\ (p'_0, p'_i)$ entraine d'après (2), $\psi(p_0, p_i)$.

□

Observons que les théorèmes 4.1.1v1 et 4.1.1v2 ne sont pas vrais pour l'invariance conditionnelle. Un contre-exemple (pour 4.1.1v1) est donné par $S = \{0,1,2\}, A = \{a\}$, $\Sigma = \{a \xrightarrow{a} 1\}$, $S' = \{0', 1'\}$, $A' = \{a'\}$, $\Sigma' = \{0' \xrightarrow{a'} 1'\}$, $\neg\phi(0,0)$, $\psi(0,0)$, $\neg\psi(0,1)$ (de sorte que ψ est invariant sous condition ϕ pour Σ), $ra(0,0')$, $ra(1,1')$, $ra(a,a')$ (de sorte que $\simeq \langle ra, ra \rangle (\Sigma, \Sigma')$) et $\phi(2,2)$, $ra(2,0')$ et $\neg\psi(2,1)$ (de sorte que $ra^{-1} \circ \psi \circ ra\ (0',0')$ est vrai tandis que $ra^{-1} \circ \psi \circ ra\ (0',1')$ est faux). Toutefois sous des conditions plus restrictives, nous obtenons les théorèmes 4.1.1v3 et 4.1.1v4 suivants (dont les théorèmes 4.1.1v1 et 4.1.1v2 sont des cas particuliers respectifs) :

Théorème 4.1.1v3

Si
- $\simeq \langle ra, ra \rangle (\langle S, A, \Sigma \rangle, \langle S', A', \Sigma' \rangle)$ (1)
- $\wedge\ (\phi'(A'_1, A'_2) \wedge ra^{-1}(A'_1, A_1) \wedge ra(A_2, A'_2)) \Rightarrow \phi(A_1, A_2)$ (2)
- $\wedge\ (\psi(A_1, A_2) \wedge ra(A_1, A'_1) \wedge ra^{-1}(A'_2, A_2)) \Rightarrow \psi'(A'_1, A'_2)$ (3)

alors

[ψ est invariante sous ϕ pour $\langle S, A, \Sigma \rangle$]

$\Rightleftharpoons$

[ψ' est invariante sous ϕ' pour $\langle S, A, \Sigma \rangle$]

Démonstration

($\Rightarrow$) Si $p' \in \Sigma'$, $i \in |p'|$ et $\forall j \in i.\ \phi'(p'_0, p'_j)$ alors d'après (1), $\exists p \in \Sigma$ tel que $|p| = |p'|$ et $\forall j \in i.\ ra(p_j, p'_j)$. Par conséquent d'après (2) nous avons $\forall j \in i.\ \phi(p_0, p_j)$ ce qui implique $\psi(p_0, p_i)$ et donc $\psi'(p'_0, p'_i)$ d'après (3).

($\Leftarrow$) Même choix que dans la démonstration de 4.1.1v1 avec $\phi'(a, a') = tt$.

Théorème 4.1.1v4

Si
- $\simeq \langle ra, ra \rangle (\langle S, A, \Sigma \rangle, \langle S', A', \Sigma' \rangle)$ (1)
- $\wedge\ (\phi'(A'_1, A'_2) \wedge ra^{-1}(A'_1, A_1) \wedge ra(A_2, A'_2)) \Rightarrow \phi(A_1, A_2)$ (2)
- $\wedge\ (\psi(A_1, A_2) \wedge ra(A_1, A'_1) \wedge ra^{-1}(A'_2, A_2)) \Rightarrow \psi'(A'_1, A'_2)$ (3)
- $\wedge\ (\phi(A_1, A_2) \wedge ra(A_1, A'_1) \wedge ra^{-1}(A'_2, A_2)) \Rightarrow \phi'(A'_1, A'_2)$ (4)
- $\wedge\ (\psi'(A'_1, A'_2) \wedge ra^{-1}(A'_1, A_1) \wedge ra(A_2, A'_2)) \Rightarrow \psi(A_1, A_2)$ (5)

alors

[ψ est invariante sous ϕ pour $\langle S, A, \Sigma \rangle$]

$\Leftrightarrow$

[ψ' est invariante sous ϕ' pour $\langle S', A', \Sigma' \rangle$]

Démonstration

($\Rightarrow$) La même que pour 4.1.1v3. ($\Leftarrow$) Si $p \in \Sigma$, $i \in |p|$ et $\forall j \in i.\ \phi(p_0, p_j)$ alors d'après (4), $\exists p' \in \Sigma'$ tel que $|p'| = |p|$ et $\forall j \in i.\ ra(p_j, p'_j)$. Par conséquent d'après (4) nous avons $\forall j \in i.\ \phi'(p'_0, p'_j)$ ce qui implique $\psi'(p'_0, p'_i)$ et donc $\psi(p_0, p_i)$ d'après (5).

□

Dans le cas particulier d'une concordance entre sémantiques à une fonction entre états près, nous obtenons le corollaire suivant :

Corollaire 4.1.1v5

Si
- $\langle S', A', \Sigma' \rangle = \simeq \langle f_A \rangle (\langle S, A, \Sigma \rangle)$
- $\wedge\ \phi(A_1, A_2) = \phi'(f_A(A_1), f_A(A_2))$
- $\wedge\ \psi(A_1, A_2) = \psi'(f_A(A_1), f_A(A_2))$

alors

[ψ est invariante sous condition ϕ pour $\langle S, A, \Sigma \rangle$]

$\Leftrightarrow$

[ψ' est invariante sous condition ϕ' pour $\langle S', A', \Sigma' \rangle$]

4.1.2 CONSERVATION DE PROPRIETES D'INVARIANCE APRES REDUCTION DES ETATS INOBSERVABLES

La conservation d'une propriété d'invariance après réduction des états inobservables est décrite (dans un cas simplifié mais dont la généralisation est aisée) par :

Théorème 4.1.2-1

Si $\langle S', A', \Sigma' \rangle = \underline{Redei}\ \langle S' \rangle (\langle S, A, \Sigma \rangle) \ \wedge\ \forall p \in \Sigma . p_0 \in S'$

alors [ψ' est invariante sous condition ϕ' pour $\langle S', A', \Sigma' \rangle$]

$\Leftrightarrow$ [$\psi(a,a') = [a' \in S' \Rightarrow \psi'(a,a')]$ est invariante sous condition $\phi(a,a') = [a' \in S' \Rightarrow \phi'(a,a')]$ pour $\langle S, A, \Sigma \rangle$]

Démonstration

($\Rightarrow$) Soit p une trace de $\langle S, A, \Sigma \rangle$, $i \in |p|$ tel que $\forall j \in i . \phi(p_0, p_j)$. Si $p_i \notin S'$ alors de manière évidente nous avons $\psi(p_0, p_i)$. Si $p_i \in S'$, il faut montrer que $\psi'(p_0, p_i)$ est vrai. Il suffit que $\phi'(p_0, p_{r(k)})$ soit vrai pour tout $k > 0$ tel que $r(k) < i$ (où $r(k)$ a été défini en 2.5.4.1). Si $k = 0$ alors $p_0 \in S'$ implique $r(k) = 0$. Si $i > 0$ alors $p_0 \in S'$ et $\phi(p_0, p_0)$ impliquent $\phi'(p_0, p_{r(0)})$. Si $k > 0$ et $r(k) < i$ alors $p_{r(k)} \in S'$ et donc à nouveau $\phi(p_0, p_{r(k)})$ implique $\phi'(p_0, p_{r(k)})$.

($\Leftarrow$) Soit p' une trace de $\langle S', A', \Sigma' \rangle$, $i' \in |p'|$ tel que $\forall j \in i' . \phi'(p'_0, p'_j)$. Ceci entraîne que pour tout $k > 0$ tel que $r(k) < r(i')$ nous avons $\phi(p_0, p_{r(k)})$. Soit maintenant $j \in r(i')$ tel que $p_j \notin S'$. Ceci entraîne de manière évidente $\phi(p_0, p_j)$. Finalement nous avons $\forall j \in r(i') . \phi(p_0, p_j)$, d'où nous déduisons $\psi(p_0, p_{r(i')})$ et donc $\psi'(p'_0, p'_{i'})$.

□

4.1.3 CONSERVATION DE PROPRIETES D'INVARIANCE PAR RETRACTION DE LA SEMANTIQUE PAR TRANSITIONS

De manière générale, on fait des preuves d'invariance par induction sur la longueur des calculs. On cherche donc à utiliser un système de transition. Cette démarche qui consiste à remplacer une preuve d'invariance relative à une sémantique par une preuve relative à la rétraction de cette sémantique par transitions est justifiée par les résultats suivants :

Théorème 4.1.3-1

[ψ est invariante sous condition ϕ pour $\langle S, A, \Sigma \rangle$ $\wedge$ $\langle S', A', \Sigma' \rangle \subseteq \langle S, A, \Sigma \rangle$]
$\Rightarrow$ [ψ est invariante sous condition ϕ pour $\langle S', A', \Sigma' \rangle$]

Théorème 4.1.3-2

[ψ est invariante sous condition ϕ pour $\langle S, A, \Sigma \rangle$]
$\Leftrightarrow$ [ψ est invariante sous condition ϕ pour $Pref^{<\omega}(\langle S, A, \Sigma \rangle)$]

Démonstration

($\Rightarrow$) Supposons $p \in \Sigma^{<\omega}\langle S, A \rangle$, $q \in \Sigma$, $p \mapsto q$, $\forall \tau \in \Sigma, k \in |\tau| . [\forall j \in k . \phi(\tau_0, \tau_j)] \Rightarrow \psi(\tau_0, \tau_k)$ et $i \in |p|$. Alors $\forall k \in |p| . p_k = q_k$ et donc $[\forall j \in i . \phi(p_0, p_j)] \Rightarrow [\forall j \in i . \phi(q_0, q_j)] \Rightarrow \psi(q_0, q_i) \Rightarrow \psi(p_0, p_i)$.

($\Leftarrow$) Si $p \in \Sigma$, $i \in |p|$ et $\forall j \in i . \phi(p_0, p_j)$ alors $p^{<i}$ est un préfixe fini de p et donc $\psi(p_0, p_i)$ est vrai.

□

Nous en déduisons que pour faire une preuve d'invariance de ψ sous condition ϕ pour une sémantique $\langle S,A,\Sigma\rangle$, il est toujours correct de faire la preuve relativement à $\underline{Rtran}(\langle S,A,\Sigma\rangle)$ c'est-à-dire en raisonnant sur le système de transition qu'elle engendre. Cette démarche n'est pas toujours complète. Plus précisement :

Corollaire 4.1.3 v 3

$[\psi$ est invariante sous condition ϕ pour $\langle S,A,\Sigma\rangle]$

(1) $\Leftrightarrow [\psi$ est invariante sous condition ϕ pour $\underline{Pref}(\langle S,A,\Sigma\rangle)]$

(2) $\Leftarrow\!\!\!\!\Rightarrow [\psi$ est invariante sous condition ϕ pour $\underline{Suff}(\langle S,A,\Sigma\rangle)]$

(3) $\Leftrightarrow [\psi$ est invariante sous condition ϕ pour $\underline{Redeca}(\langle S,A,\Sigma\rangle)]$

(4) $\Leftarrow\!\!\!\!\Rightarrow [\psi$ est invariante sous condition ϕ pour $\underline{Efus}(\langle S,A,\Sigma\rangle)]$

(5) $\Leftrightarrow [\psi$ est invariante sous condition ϕ pour $\underline{Flim}(\langle S,A,\Sigma\rangle)]$

(6) $\Rightarrow\!\!\!\!\Leftarrow [\psi$ est invariante sous condition ϕ pour $\underline{Retps}(\langle S,A,\Sigma\rangle)]$

(7) $\Leftarrow\!\!\!\!\Rightarrow [\psi$ est invariante sous condition ϕ pour $\underline{Rtran}(\langle S,A,\Sigma\rangle)]$

Si $\underline{card}(\alpha)<\omega$ alors

(8) $\Leftrightarrow [\psi$ est invariante sous condition ϕ pour $\underline{Wfair}\langle\alpha\rangle(\langle S,A,\Sigma\rangle)]$

(9) $\Leftrightarrow [\psi$ est invariante sous condition ϕ pour $\underline{Sfair}\langle\alpha\rangle(\langle S,A,\Sigma\rangle)]$

Démonstration

(1) ($\Leftarrow$) $\underline{Pref}$ est extensive et 4.1.3 v 1. ($\Rightarrow$) 2.6.1 v 2 et 4.1.3 v 2.

(2) ($\Leftarrow$) $\underline{Suff}$ est extensive et 4.1.3 v 1. ($\not\Rightarrow$) $\psi(s,s') = [s'=1]$ est invariante sous condition $\phi(s,s') = [s' \in \{0,2\}]$ pour la trace $0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} 3$ mais pas pour son suffixe $2 \xrightarrow{a} 3$.

(3) Trivial.

(4) ($\Leftarrow$) $\underline{Efus}$ est extensive et 4.1.3 v 1. ($\not\Rightarrow$) Si $\psi(0,0)$, $\psi(0,1)$, $\psi(1,1)$, $\psi(1,2)$ et $\phi(s,s')$ sont vrais alors ψ est invariant sous condition ϕ pour les traces $0 \xrightarrow{a} 1$ et $1 \xrightarrow{a} 2$ mais pas pour la fusion $0 \xrightarrow{a} 1 \xrightarrow{a} 2$.

(5) 4.1.3 v 2 et 2.6.7 v 3-(1).

(6) ($\Rightarrow$) $\underline{Retps}$ est réductive et 4.1.3 v 1. ($\not\Leftarrow$) $\psi(s,s') = ff$ est invariante sous condition $\phi(s,s') = tt$ pour $\langle\{0\},\{a\},\phi\rangle = \underline{Retps}(\langle\{0\},\{a\},\Sigma\rangle)$ avec $\Sigma = \{\sigma, 0 \xrightarrow{a} 0, \ldots, 0 \xrightarrow{a} 0 \xrightarrow{a} 0 \ldots 0 \xrightarrow{a} 0, \ldots\}$ mais pas pour $\langle\{0\},\{a\},\Sigma\rangle$.

(7) ($\Leftarrow$) ψ est invariant sous condition ϕ pour $\underline{Rtran}(\langle S,A,\Sigma\rangle)$ si et seulement si d'après 4.1.3 v 3.1 ψ est invariant sous condition ϕ pour $\underline{Pref}^{\omega} \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ ce qui entraine d'après 2.6.8 v 1 et 4.1.3 v 1 que ψ est invariant sous condition ϕ pour $\langle S,A,\Sigma\rangle$. ($\not\Rightarrow$) Même contre-exemple que pour (4).

(8), (9) 4.1.3 v 2 et 2.6.4 v 1.

□

Observons que la démarche qui consiste à remplacer une preuve d'invariance relative à une sémantique $\langle S,A,\Sigma\rangle$ par une preuve relative à $\underline{Rtran}(\langle S,A,\Sigma\rangle)$ est toujours correcte mais pas toujours complète (cf. 4.1.3 v 3).

Toutefois la complétude est obtenue si la sémantique $\langle S,A,\Sigma\rangle$ est fermée par fusions :

Théorème 4.1.3 v 4

$\rightarrow$ $[\psi$ est invariante sous condition ϕ pour $\langle S,A,\Sigma\rangle \wedge \underline{Efus}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle]$
$[\psi$ est invariante sous condition ϕ pour $\underline{Rtran}(\langle S,A,\Sigma\rangle)]$

Démonstration

ψ est invariante sous condition ϕ pour $\langle S,A,\Sigma\rangle$ si et seulement si ψ est invariante sous condition ϕ pour $\underline{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)$ d'après 4.1.3 v 2. Ceci entraine d'après 2.6.8 v 7 et 4.1.3 v 1 que ψ est invariante sous condition ϕ pour $\underline{Pref}^{<\omega} \circ \underline{Rtran}(\langle S,A,\Sigma\rangle)$ qui d'après 4.1.3 v 2 implique que ψ est invariante sous condition ϕ pour $\underline{Rtran}(\langle S,A,\Sigma\rangle)$.

□

Par conséquent, d'après 4.1.3∨3.8, 4.1.3∨3.9 et 4.1.3∨4, les preuves d'invariance pour le langage que nous avons considéré en 2.8 peuvent toujours se faire en raisonnant sur un système de transition.

Dans le cas d'une sémantique non fermée par fusions, il est tout de même possible de se ramener à un raisonnement sur un système de transition, par exemple en spécifiant cette sémantique par concordance avec une sémantique close (cf. 2.7.2.2).

4.2 PRINCIPES D'INDUCTION

Un principe d'induction est l'essence d'une méthode de preuve.

4.2.1 PRINCIPES D'INDUCTION POUR LES SEMANTIQUES CLOSES

Nous commençons par considérer le cas des sémantiques engendrées par un système de transition.

4.2.1.1 Principe d'induction de base pour l'invariance

Le principe d'induction de base est l'essence de la méthode de Floyd-Naur. Toutes les autres méthodes de preuve de propriétés d'invariance en dérivent. Ce principe est déduit de l'

Exemple 4.2.1.1-1

Pour démontrer la correction partielle du programme suivant qui calcule le quotient et le reste de deux entiers x et y :

```
1:
2:  q := 0;
3:  while x ≥ y do
4:      q := q+1;
5:      x := x-y;
6:  od;
```

il faut établir la relation $\underline{x} = \bar{q} \times \underline{y} + \bar{x} \wedge \bar{x} < \underline{y}$ entre les valeurs initiales $\underline{x}, \underline{y}, \underline{q}$ et finales $\bar{x}, \bar{y}, \bar{q}$ des variables x, y et q.

Comme la méthode de Floyd-Naur n'utilise que des assertions P(x,y,q) sur les valeurs des variables X, Y et Q, il faut introduire une variable auxiliaire X1 à laquelle est affectée la valeur initiale de X en début de programme et qui n'est plus modifiée par la suite :

X1:=X; Q:=0; <u>while</u> X≥Y <u>do</u> Q:=Q+1; X:=X-Y; <u>od</u>

de sorte qu'à la terminaison nous pourrons prouver que $[x_1 = qxy + x \wedge x<y]$

Cette transformation peut être évitée en utilisant la méthode de Manna[71] qui est tout à fait similaire à la méthode de Floyd-Naur mais qui consiste à utiliser des relations entre valeurs initiales et courantes des variables plutôt que des assertions sur les valeurs courantes. La méthode consiste à associer un invariant local P_i à chaque point i, i=1,...,6 du programme. L'invariant local P_i associé au point i du programme est une relation entre les valeurs initiales $\underline{x}, \underline{y}$ (nous omettons q qui est inutile) des variables X, Y et les valeurs courantes x, y, q de ces variables X, Y, Q qui est vrai quand le contrôle atteint ce point i :

$P_1(\underline{x},\underline{y},x,y,q) = [x=\underline{x} \wedge y=\underline{y}]$

$P_2(\underline{x},\underline{y},x,y,q) = [x=\underline{x} \wedge y=\underline{y} \wedge q=0]$

$P_3(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=qxy+x]$

$P_4(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=(q-1)x\underline{y}+x]$

$P_5(\underline{x},\underline{y},x,y,q) = [y=\underline{y} \wedge \underline{x}=qx\underline{y}+x]$

$P_6(\underline{x},\underline{y},x,y,q) = [\underline{x}=qx\underline{y}+x \wedge x<\underline{y} \wedge y=\underline{y}]$

Les conditions de vérification sont similaires à celles obtenues par la méthode de Floyd-Naur, excepté pour la condition d'entrée :

$P_1(\underline{x},\underline{y},x,y,q) \Leftarrow [x=\underline{x} \wedge y=\underline{y}]$

$P_2(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists q'.\ P_1(\underline{x},\underline{y},x,y,q') \wedge q=0]$

$P_3(\underline{x},\underline{y},x,y,q) \Leftarrow [(P_2(\underline{x},\underline{y},x,y,q) \vee P_5(\underline{x},\underline{y},x,y,q)) \wedge x \geq y]$

$P_4(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists q'.\ P_3(\underline{x},\underline{y},x,y,q') \wedge q=q'+1]$

$P_5(\underline{x},\underline{y},x,y,q) \Leftarrow [\exists x'.\ P_4(\underline{x},\underline{y},x',y,q) \wedge x=x'-y]$

$P_6(\underline{x},\underline{y},x,y,q) \Leftarrow [(P_6(\underline{x},\underline{y},x,y,q) \vee P_5(\underline{x},\underline{y},x,y,q)) \wedge x<y]$

$[\underline{x}=qx\underline{y}+x \wedge x<\underline{y} \wedge y=\underline{y}] \Leftarrow P_6(\underline{x},\underline{y},x,y,q)$

Ces conditions de vérification expriment que

- $P_1(\underline{x},\underline{y},x,y,q)$ est vrai quand l'exécution commence avec des valeurs $\underline{x},\underline{y},q$ de X, Y, Q
- Si l'exécution commencée avec les valeurs $\underline{x},\underline{y},q$ de X, Y, Q atteint le point i du programme qui est immédiatement suivi par le point j du programme, alors l'hypothèse que $P_i(\underline{x},\underline{y},x',y',q')$ est vrai en i pour les valeurs courantes x', y', q' des variables X, Y, Q implique que $P_j(\underline{x},\underline{y},x,y,q)$ est vrai quand le contrôle est en j avec les valeurs x, y, q des variables X, Y, Q.

Par induction sur la longueur des calculs, on en déduit que si l'exécution commence avec $X=\underline{x}$ et $Y=\underline{y}$ et atteint le point i avec X=x, Y=y et Q=q alors $P_i(\underline{x},\underline{y},x,y,q)$ est vrai. En particulier, si i=6 la dernière condition de vérification permet de conclure que le programme est partiellement correct.

Pour dégager l'essence de cette preuve nous considérions la sémantique du programme (comme elle a été définie en 2.8.1.2) :

Les états <L,M> de ce programme consistent en un état de contrôle L (i.e. un point du programme) et un état mémoire (i.e. une fonction M qui définit les valeurs M(X), M(Y), M(Q) des variables X, Y, Q) :

$\mathcal{L} = \{1,2,3,4,5,6\}$

$\mathcal{V} = \{x, y, q\}$

$\mathcal{M} = (\mathcal{V} \to \mathcal{D})$

$S = \mathcal{L} \times \mathcal{M}$

Les états initiaux $\underline{s}$ du programme correspondent au point 1 du programme avec des valeurs arbitraires des variables :

$\varepsilon(\underline{s}) = [\exists M \in (\mathcal{V} \to \mathcal{D}).\ \underline{s} = \langle 1, M \rangle]$

La relation de transition est définie par cas en écrivant $\langle L, M \rangle \xrightarrow{t} \langle L', M' \rangle$ quand $t(\langle L, M \rangle, \langle L', M' \rangle)$ est vraie (et en omettant l'action unique, cf. 2.8.1.3.2). En plus (x, y, q) dénote la fonction M lorsque $M(X) = x$, $M(Y) = y$ et $M(Q) = q$ et les valeurs des variables X, Y, Q du programme sont implicitement universellement quantifiées sur $\mathbb{Z}$:

$\langle 1, (x,y,q) \rangle \xrightarrow{t} \langle 2, (x,y,0) \rangle$

$\langle 2, (x,y,q) \rangle \xrightarrow{t} \langle 3, (x,y,q) \rangle$ si et seulement si $x \geq y$

$\langle 2, (x,y,q) \rangle \xrightarrow{t} \langle 6, (x,y,q) \rangle$ si et seulement si $x < y$

$\langle 3, (x,y,q) \rangle \xrightarrow{t} \langle 4, (x,y,q+1) \rangle$

$\langle 4, (x,y,q) \rangle \xrightarrow{t} \langle 5, (x-y,y,q) \rangle$

$\langle 5, (x,y,q) \rangle \xrightarrow{t} \langle 3, (x,y,q) \rangle$ si et seulement si $x \geq y$

$\langle 5, (x,y,q) \rangle \xrightarrow{t} \langle 6, (x,y,q) \rangle$ si et seulement si $x < y$

Définissons

$\bar{\psi}(\underline{x}, \underline{y}, \underline{q}, \bar{x}, \bar{y}, \bar{q}) = [\underline{x} = \bar{q} \times \underline{y} + \bar{x} \wedge \bar{x} < \underline{y} \wedge \bar{y} = \underline{y}]$

$\psi(\underline{s}, \bar{s}) = [\exists \underline{x}, \underline{y}, \underline{q}, \bar{x}, \bar{y}, \bar{q} \in \mathbb{Z}.\ \underline{s} = \langle 1, (\underline{x}, \underline{y}, \underline{q}) \rangle \wedge \bar{s} = \langle 6, (\bar{x}, \bar{y}, \bar{q}) \rangle \wedge \bar{\psi}(\underline{x}, \underline{y}, \underline{q}, \bar{x}, \bar{y}, \bar{q})]$

alors lorsque nous disons que le programme est partiellement correct cela signifie :

$\forall p \in \Sigma\langle S, A, t, \varepsilon \rangle,\ i \in |p|.\ \psi(p_0, p_i)$

Nous pouvons maintenant dériver le principe d'induction de l'exemple par abstractions successives :

Notre première abstraction consiste à noter que les invariants locaux P_i associés aux points i, $i = 1, \ldots, 6$ du programme peuvent être compris comme une relation I sur des états. Nous avons

$P_1(\underline{x}, \underline{y}, x, y, q) = [x = \underline{x} \wedge y = \underline{y}]$

$P_2(\underline{x}, \underline{y}, x, y, q) = [x = \underline{x} \wedge y = \underline{y} \wedge q = 0]$

$P_3(\underline{x}, \underline{y}, x, y, q) = [y = \underline{y} \wedge \underline{x} = q \times \underline{y} + x]$

$P_4(\underline{x}, \underline{y}, x, y, q) = [y = \underline{y} \wedge \underline{x} = (q-1) \times \underline{y} + x]$

$P_5(\underline{x}, \underline{y}, x, y, q) = [y = \underline{y} \wedge \underline{x} = q \times \underline{y} + x]$

$P_6(\underline{x}, \underline{y}, x, y, q) = [\underline{x} = q \times \underline{y} + x \wedge x < \underline{y} \wedge y = \underline{y}]$

de sorte que

$I \in (S \times S \to \{tt, ff\})$

$I(\underline{s}, s) = [\exists j \in \mathcal{L}, \underline{x}, \underline{y}, \underline{q}, x, y, q \in \mathbb{Z}.\ \underline{s} = \langle 1, (\underline{x}, \underline{y}, \underline{q}) \rangle \wedge s = \langle j, (x, y, q) \rangle \wedge P_j(\underline{x}, \underline{y}, x, y, q)]$

Notre seconde abstraction consiste à comprendre les conditions de vérification sur les P_i, $i = 1, \ldots, 6$ en termes de conditions de vérification équivalentes faisant intervenir I et nos définitions abstraites du programme et de sa correction partielle (c'est à dire en termes de ε, t, ε, δ et ψ) :

- La première condition de vérification était

$[x = \underline{x} \wedge y = \underline{y}] \Rightarrow P_1(\underline{x}, \underline{y}, x, y, q)$

c'est à dire $P_1(\underline{x}, \underline{y}, x, y, q)$ qui est équivalent à $\forall \underline{s},\ \varepsilon(\underline{s}) \Rightarrow I(\underline{s}, \underline{s})$ puisque $[\exists \underline{x}, \underline{y}, \underline{q} \in \mathbb{Z}.\ \underline{s} = \langle 1, (\underline{x}, \underline{y}, \underline{q}) \rangle] \Rightarrow I(\underline{s}, \underline{s})$ est équivalent à $I(\langle 1, (\underline{x}, \underline{y}, \underline{q}) \rangle, \langle 1, (\underline{x}, \underline{y}, \underline{q}) \rangle)$ soit $[\exists j \in \mathcal{L}.\ j = 1 \wedge P_j(\underline{x}, \underline{y}, \underline{x}, \underline{y}, \underline{q})] = P_1(\underline{x}, \underline{y}, \underline{x}, \underline{y}, \underline{q})$.

- Les conditions de vérification 2 à 6

$[\exists q'.\ P_1(\underline{x}, \underline{y}, x, y, q') \wedge q = 0] \Rightarrow P_2(\underline{x}, \underline{y}, x, y, q)$

$[(P_2(\underline{x}, \underline{y}, x, y, q) \vee P_5(\underline{x}, \underline{y}, x, y, q)) \wedge x \geq y] \Rightarrow P_3(\underline{x}, \underline{y}, x, y, q)$

$[\exists q'.\ P_3(\underline{x}, \underline{y}, x, y, q') \wedge q = q' + 1] \Rightarrow P_4(\underline{x}, \underline{y}, x, y, q)$

$[\exists x'.\ P_4(\underline{x}, \underline{y}, x', y, q) \wedge x = x' - y] \Rightarrow P_5(\underline{x}, \underline{y}, x, y, q)$

$[(P_2(\underline{x}, \underline{y}, x, y, q) \vee P_5(\underline{x}, \underline{y}, x, y, q)) \wedge x < y] \Rightarrow P_6(\underline{x}, \underline{y}, x, y, q)$

sont équivalentes à :

$[\exists x',y',q'.\ P_1(\underline{x},\underline{y},x',y',q') \wedge x=x' \wedge y=y' \wedge q=0] \rightarrow P_t(\underline{x},\underline{y},x,y,q)$

$[\exists x',y',q'.\ P_2(\underline{x},\underline{y},x',y',q') \wedge x'\geq y' \wedge x=x' \wedge y=y' \wedge q=q'] \rightarrow P_3(\underline{x},\underline{y},x,y,q)$

$[\exists x',y',q'.\ P_3(\underline{x},\underline{y},x',y',q') \wedge x'\geq y' \wedge x=x' \wedge y=y' \wedge q=q'] \Rightarrow P_3(\underline{x},\underline{y},x,y,q)$

$[\exists x',y',q'.\ P_3(\underline{x},\underline{y},x',y',q') \wedge x=x' \wedge y=y' \wedge q=q'+1] \Rightarrow P_4(\underline{x},\underline{y},x,y,q)$

$[\exists x',y',q'.\ P_4(\underline{x},\underline{y},x',y',q') \wedge x=x'-y' \wedge y=y' \wedge q=q'] \rightarrow P_5(\underline{x},\underline{y},x,y,q)$

$[\exists x',y',q'.\ P_5(\underline{x},\underline{y},x',y',q') \wedge x'<y' \wedge x=x' \wedge y=y' \wedge q=q'] \rightarrow P_6(\underline{x},\underline{y},x,y,q)$

$[\exists x',y',q'.\ P_5(\underline{x},\underline{y},x',y',q') \wedge x'<y' \wedge x=x' \wedge y=y' \wedge q=q'] \rightarrow P_6(\underline{x},\underline{y},x,y,q)$

Utilisant I, t, $\underline{M}=(\underline{x},\underline{y},q)$ $M'=(x',y',q')$ et $M=(x,y,q)$, elles peuvent s'écrire :

$[I(\langle 1,\underline{M}\rangle,\langle 1,M'\rangle) \wedge t(\langle 1,M'\rangle,\langle 2,M\rangle)] \Rightarrow I(\langle 1,\underline{M}\rangle,\langle 2,M\rangle)$

$[I(\langle 1,\underline{M}\rangle,\langle 2,M'\rangle) \wedge t(\langle 2,M'\rangle,\langle 3,M\rangle)] \Rightarrow I(\langle 1,\underline{M}\rangle,\langle 3,M\rangle)$

$[I(\langle 1,\underline{M}\rangle,\langle 5,M'\rangle) \wedge t(\langle 5,M'\rangle,\langle 3,M\rangle)] \Rightarrow I(\langle 1,\underline{M}\rangle,\langle 3,M\rangle)$

$[I(\langle 1,\underline{M}\rangle,\langle 3,M'\rangle) \wedge t(\langle 3,M'\rangle,\langle 4,M\rangle)] \Rightarrow I(\langle 1,\underline{M}\rangle,\langle 4,M\rangle)$

$[I(\langle 1,\underline{M}\rangle,\langle 4,M'\rangle) \wedge t(\langle 4,M'\rangle,\langle 5,M\rangle)] \Rightarrow I(\langle 1,\underline{M}\rangle,\langle 5,M\rangle)$

$[I(\langle 1,\underline{M}\rangle,\langle 2,M'\rangle) \wedge t(\langle 2,M'\rangle,\langle 6,M\rangle)] \rightarrow I(\langle 1,\underline{M}\rangle,\langle 6,M\rangle)$

$[I(\langle 1,\underline{M}\rangle,\langle 5,M'\rangle) \wedge t(\langle 5,M'\rangle,\langle 6,M\rangle)] \Rightarrow I(\langle 1,\underline{M}\rangle,\langle 6,M\rangle)$

c'est-à-dire

$\forall L',L \in \mathfrak{B}.\ [I(\langle 1,\underline{M}\rangle,\langle L',M'\rangle) \wedge t(\langle L',M'\rangle,\langle L,M\rangle)] \rightarrow I(\langle 1,\underline{M}\rangle,\langle L,M\rangle)$

qui est équivalent à :

$\forall \underline{a},a',a.\ [\varepsilon(\underline{a}) \wedge I(\underline{a},a') \wedge t(a',a)] \Rightarrow I(\underline{a},a)$

- La dernière condition de vérification est

$P_6(\underline{x},\underline{y},\bar{x},\bar{y},\bar{q}) \Rightarrow [\underline{x}=\bar{q}\times\underline{y}+\bar{x} \wedge \bar{x}<\underline{y} \wedge \bar{y}=\underline{y}]$

c'est-à-dire

$I(\langle 1,\underline{M}\rangle,\langle 6,\bar{M}\rangle) \Rightarrow \Psi(\langle 1,\underline{M}\rangle,\langle 6,\bar{M}\rangle)$

soit

$\forall \underline{a},\bar{a}.\ [\varepsilon(\underline{a}) \wedge I(\underline{a},\bar{a})] \Rightarrow \psi(\underline{a},\bar{a})$

Ainsi, nous avons montré que cette méthode de preuve consiste essentiellement à découvrir un invariant I et à prouver que

$[\ \forall \underline{a} \in S.\ \varepsilon(\underline{a}) \Rightarrow I(\underline{a},\underline{a})$

$\wedge\ (\forall \underline{a},a',a \in S.\ [\varepsilon(\underline{a}) \wedge I(\underline{a},a') \wedge t(a',a)] \Rightarrow I(\underline{a},a))$

$\wedge\ (\forall \underline{a},\bar{a} \in S.\ [\varepsilon(\underline{a}) \wedge I(\underline{a},\bar{a})] \Rightarrow \psi(\underline{a},\bar{a}))\ ]$

pour en déduire

$\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle,\ i \in |p|.\ \psi(p_0,p_i)$

□

Le principe d'induction de base est

Théorème 4.2.1.1 ~1

$[\exists I \in (S^2 \rightarrow \{\mathit{ff},\mathit{tt}\}).\ \forall \underline{a},a,\bar{a} \in S,\ a \in A$

$(\mathcal{I}.\varepsilon)$	$\varepsilon(\underline{a}) \Rightarrow I(\underline{a},\underline{a})$	
	$\wedge$	
$(\mathcal{I}.i)$	$[\exists a' \in S.\ \varepsilon(\underline{a}) \wedge I(\underline{a},a') \wedge t_a(a',a)] \rightarrow I(\underline{a},a)$	$(\mathcal{I})$
	$\wedge$	
$(\mathcal{I}.s)$	$[\varepsilon(\underline{a}) \wedge I(\underline{a},\bar{a})] \Rightarrow \psi(\underline{a},\bar{a})\ ]$	
	$\Leftrightarrow$	
	$[\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle,\ i \in |p|.\ \psi(p_0,p_i)]$	

Démonstration

($\Rightarrow$) Pour la preuve de correction, soit $p \in \Sigma\langle S,A,t,\varepsilon\rangle$. Démontrons par récurrence sur $i \in |p|$ que $I(p_0,p_i)$. Pour $i=0$, nous avons $\varepsilon(p_0)$ et donc $I(p_0,p_0)$ d'après $(\mathcal{I}.\varepsilon)$. Si $I(p_0,p_i)$ et $i+1 \in |p|$ alors $t_{p_i}(p_i,p_{i+1})$ et $(\mathcal{I}.i)$ impliquent $I(p_0,p_{i+1})$. Nous déduisons de $(\mathcal{I}.\varepsilon)$ que $\forall i \in |p|.\ \psi(p_0,p_i)$.

($\Leftarrow$) La preuve de complétude sémantique est également très simple en choisissant $I(\underline{a},a) = [\exists p \in \Sigma\langle S,A,t,\varepsilon\rangle,\ i \in |p|.\ (p_n=\underline{a} \wedge p_i=a)]$. $(\mathcal{I}.\varepsilon)$ et $(\mathcal{I}.i)$ dérivent

alors de la définition de $\Sigma\langle S,A,t,\varepsilon\rangle$. $(\mathcal{Y}.s)$ dérive de l'hypothèse $\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|. \psi(p_0,p_i)$.

□

Remarque 4.2.1.1-2 (Invariance conditionnelle)

Par souci de simplicité, nous donnons uniquement les principes d'induction pour l'invariance car la généralisation à l'invariance conditionnelle est triviale. Par exemple le principe d'induction $(\mathcal{Y})$ se généralise immédiatement en :

$$
\begin{array}{l}
[\exists I \in (S \times S \to \{tt, ff\}).\ \forall \underline{s}, s, \bar{s} \in S,\ a \in A. \\
\qquad \varepsilon(\underline{s}) \Rightarrow I(\underline{s},\underline{s}) \\
\qquad \wedge\ [\exists s' \in S.\ \varepsilon(\underline{s}) \wedge I(\underline{s},s') \wedge \phi(\underline{s},s') \wedge t_a(s',s)] \Rightarrow I(\underline{s},s) \\
\qquad \wedge\ [\varepsilon(\underline{s}) \wedge I(\underline{s},\bar{s})] \Rightarrow \psi(\underline{s},\bar{s})\ ] \\
\leftrightarrow \\
[\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|.\ [\forall j \in i.\ \phi(p_0,p_j)] \Rightarrow \psi(p_0,p_i)\ ]
\end{array}
$$

La preuve de correction consiste essentiellement à démontrer que pour tout $p \in \Sigma\langle S,A,t,\varepsilon\rangle$ nous avons par récurrence sur $i \in |p|$, $[\forall j \in i.\ \phi(p_0,p_j)] \Rightarrow I(p_0,p_i)$
Pour la preuve de complétude nous choisissons $I(\underline{s},s) =$
$[\exists p \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|.\ (p_0 = \underline{s} \wedge \forall j \in i.\ \phi(p_0,p_j) \wedge p_i = s)]$.

□

4.2.1.2 Transformations de principes d'induction

Par transformation du principe d'induction de base $(\mathcal{Y})$, nous obtenons un certain nombre de principes d'induction dérivés qui permettent de rendre compte des méthodes de preuve d'invariance existentes mais également d'en découvrir de nouvelles.

4.2.1.2.1 Transformation par distinction/confusion des états initiaux ou finaux

Observons que la définition de l'invariance

$\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|. \psi(p_0,p_i)$

est équivalente à

$\forall p \in \Sigma\langle S,A,t,tt\rangle, i \in |p|. (\varepsilon(p_0) \Rightarrow \psi(p_0,p_i))$

en posant $tt(a) = tt$.

Par conséquent, nous obtenons un principe d'induction $(\mathcal{J})$ équivalent à $(\mathcal{Y})$ en substituant tt à ε et $\psi'(s,s') = (\varepsilon(s) \Rightarrow \psi(s,s'))$ à $\psi(s,s')$ dans $(\mathcal{Y})$:

$$
\begin{array}{ll}
 & [\exists I \in (S^2 \to \{tt, ff\}).\ \forall \underline{s}, s, \bar{s} \in S,\ a \in A. \\
(\mathcal{J}.\varepsilon) & \qquad I(\underline{s},\underline{s}) \\
(\mathcal{J}.\iota) & \qquad \wedge\ [\exists s' \in S.\ I(\underline{s},s') \wedge t_a(s',s)] \Rightarrow I(\underline{s},s) \\
(\mathcal{J}.s) & \qquad \wedge\ I(\underline{s},\bar{s}) \Rightarrow \psi(\underline{s},\bar{s})\ ] \qquad (\mathcal{J}) \\
 & \qquad \Longleftrightarrow \\
 & [\forall p \in \Sigma\langle S,A,t,tt\rangle, i \in |p|.\ \psi(p_0,p_i)]
\end{array}
$$

Par symétrie avec les états initiaux, on peut être amené à distinguer des états finaux, en écrivant la définition de l'invariance sous la forme

$\forall p \in \Sigma\langle S, A, t, \varepsilon\rangle, i \in |p|. \ (\sigma(p_i) \Rightarrow \psi(p_0, p_i))$

Si nous substituons $\psi'(a,a') = [\sigma(a') \Rightarrow \psi(a,a')]$ à $\psi(a,a')$ dans $(-\mathcal{Y}-)$, nous obtenons $(-\mathcal{Y}-)$:

$$
\begin{array}{l}
[\,\exists I \in (S \times S \to \{\mathit{tt}, \mathit{ff}\}).\ \forall \underline{a}, a, \bar{a} \in S,\ a \in A. \\
\quad (-\mathcal{Y}-.\varepsilon) \qquad \varepsilon(\underline{a}) \Rightarrow I(\underline{a},\underline{a}) \\
\quad (-\mathcal{Y}-.i) \qquad \wedge\ [\exists a' \in S.\ \varepsilon(\underline{a}) \wedge I(\underline{a},a') \wedge t_a(a',a)] \Rightarrow I(\underline{a},a) \\
\quad (-\mathcal{Y}-.s) \qquad \wedge\ [\varepsilon(\underline{a}) \wedge I(\underline{a},\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \psi(\underline{a},\bar{a})\ ] \\
\quad \Longleftrightarrow \\
[\forall p \in \Sigma\langle S, A, t, \mathit{tt}\rangle, i \in |p|.\ (\,[\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_0, p_i))\,]
\end{array}
\qquad (-\mathcal{Y}-)
$$

4.2.1.2.2 Transformation par déduction/prédiction

Pour l'affectation

```
4:
    x := x - y ;
5:
```

la condition de vérification due à Floyd est "déductive" :

$P_5(\underline{z}, \underline{y}, x, y, q) \Leftarrow [\exists x'.\ P_4(\underline{z}, \underline{y}, x', y, q) \wedge x = x' - y\,]$

La plus forte post-condition déduite de la précondition doit entraîner la post-condition. La condition de vérification due à Hoare est "prédictive" :

$P_4(\underline{z}, \underline{y}, x, y, q) \Rightarrow P_5(\underline{z}, \underline{y}, x - y, y, q)$

La précondition doit entraîner la plus faible précondition prédite à partir de la post-condition. Ces deux conditions de vérification sont équivalentes.

Cette remarque se généralise comme suit :

La condition de vérification $(-\mathcal{Y}-.i)$

$[\forall \underline{a}, a \in S,\ a \in A.\ [\exists a' \in S.\ \varepsilon(\underline{a}) \wedge I(\underline{a},a') \wedge t_a(a',a)] \Rightarrow I(\underline{a},a)]$

est équivalente à

$[\forall \underline{a}, a' \in S,\ a \in A.\ [\varepsilon(\underline{a}) \wedge I(\underline{a},a')] \Rightarrow [t_a(a',a) \Rightarrow I(\underline{a},a)]]$

$\Leftrightarrow\ [\forall \underline{a}, a' \in S,\ a \in A.\ [\varepsilon(\underline{a}) \wedge I(\underline{a},a')] \Rightarrow [\forall a \in S.\ t_a(a',a) \Rightarrow I(\underline{a},a)]]$

$\Leftrightarrow\ [\forall \underline{a}, a \in S,\ a \in A.\ [\varepsilon(\underline{a}) \wedge I(\underline{a},a)] \Rightarrow \neg[\exists a' \in S.\ t_a(a,a') \wedge \neg I(\underline{a},a')]]$

Nous allons à partir de $(-\mathcal{Y}-)$, nous dériverons le principe d'induction équivalent $(-\tilde{\mathcal{Y}}-)$:

$$
\begin{array}{l}
[\,\exists I \in (S \times S \to \{\mathit{tt}, \mathit{ff}\}).\ \forall \underline{a}, a, \bar{a} \in S,\ a \in A. \\
\quad (-\tilde{\mathcal{Y}}-.\varepsilon) \qquad \varepsilon(\underline{a}) \Rightarrow I(\underline{a},\underline{a}) \\
\quad (-\tilde{\mathcal{Y}}-.i) \qquad \wedge\ [\varepsilon(\underline{a}) \wedge I(\underline{a},a)] \Rightarrow \neg[\exists a' \in S.\ t_a(a,a') \wedge \neg I(\underline{a},a')] \\
\quad (-\tilde{\mathcal{Y}}-.s) \qquad \wedge\ [\varepsilon(\underline{a}) \wedge I(\underline{a},\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \psi(\underline{a},\bar{a})\,] \\
\quad \Longleftrightarrow \\
[\forall p \in \Sigma\langle S, A, t, \mathit{tt}\rangle, i \in |p|.\ [\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_0, p_i)\,]
\end{array}
\qquad (-\tilde{\mathcal{Y}}-)
$$

4.2.1.2.3 Transformation par inversion

Définissons l'inverse p^{-1} d'une trace finie

$p = a_0 \xrightarrow{a_0} a_1 \cdots a_{m-2} \xrightarrow{a_{m-2}} a_{m-1} = \langle m, a, a\rangle$

comme étant :

$p^{-1} = a_{m-1} \xrightarrow{a_{m-2}} a_{m-2} \cdots a_1 \xrightarrow{a_0} a_0 = \langle m, a', a'\rangle$

où $\forall i \in m.\ a'_i = a_{m-i-1}$ et $\forall i \in (m-1).\ a'_i = a_{m-i-2}$

et l'inverse d'un ensemble Σ de traces comme étant l'ensemble Σ^{-1} des inverses des traces de Σ :

$\Sigma^{-1} = \{p^{-1} : p \in \Sigma\}$

Démontrer pour tout p qu'on a la propriété d'invariance

$$\forall i \in |p|.[[\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_0,p_i)]$$

est équivalent, d'après le théorème 4.1.3 v 2, à la preuve que pour tous les préfixes finis $q \in \Sigma^{<\omega}$ des traces de Σ, nous avons :

$$[\varepsilon(q_0) \wedge \sigma(q_{|q|-1})] \Rightarrow \psi(q_0, q_{|q|-1})$$

en posant $\langle S, A, \Sigma^{<\omega}\rangle = \underline{\text{Pref}}^{<\omega}(\langle S,A,\Sigma\rangle)$, ou bien encore en raisonnant sur les traces inverses :

$$\forall q \in (\Sigma^{<\omega})^{-1}.[[\sigma(q_0) \wedge \varepsilon(q_{|q|-1})] \Rightarrow \psi^{-1}(q_0, q_{|q|-1})]$$

Dans le cas particulier où tout état p_i d'une trace p de Σ satisfaisant ε est origine du suffixe de la trace de p commençant à p_i :

$$\forall p \in \Sigma, i \in |p|.\ [\varepsilon(p_i) \Rightarrow (p^{\geq i} \in \Sigma)]$$

la propriété ci-dessus est équivalente à la propriété d'invariance suivante qui porte sur les inverses de préfixes finis de Σ :

$$\forall p \in (\Sigma^{<\omega})^{-1},\ i \in |p|.\ [[\sigma(p_0) \wedge \varepsilon(p_i)] \Rightarrow \psi^{-1}(p_0, p_i)]$$

Dans ce cas, toute preuve d'invariance portant sur $S, A, \Sigma, \varepsilon, \sigma, \psi$ peut se faire en raisonnant respectivement sur S, A, $(\Sigma^{<\omega})^{-1}$, $\sigma, \varepsilon, \psi^{-1}$.

Dans le cas particulier où l'ensemble de traces est engendré par un système de transition $\langle S, A, t, \varepsilon\rangle$, nous obtenons :

$$[\forall \underline{a}, \bar{a} \in S.\ [\sigma(\underline{a}) \wedge (t^*(\underline{a},\bar{a}))^{-1} \wedge \varepsilon(\bar{a})] \Rightarrow \psi^{-1}(\underline{a},\bar{a})]$$

ou peut s'écrire puisque $(t^*(\underline{a},\bar{a}))^{-1} = t^{-1*}(\underline{a},\bar{a})$,

$$[\forall \underline{a},\bar{a} \in S.\ [\sigma(\underline{a}) \wedge t^{-1*}(\underline{a},\bar{a}) \wedge \varepsilon(\bar{a})] \Rightarrow \psi^{-1}(\underline{a},\bar{a})]$$

ou peut se démontrer en utilisant le principe d'induction $(\mathcal{Y}_-)$ où $\varepsilon, t, \sigma, \psi$ sont respectivement choisis comme $\sigma, t^{-1}, \varepsilon, \psi^{-1}$ d'où les conditions de vérification suivantes :

$$\begin{aligned}
&[\exists I \in (S\times S \to \{tt, ff\}).\ \forall \underline{a}, a, \bar{a} \in S,\ a \in A.\\
&\quad \sigma(\underline{a}) \Rightarrow I(\underline{a},\underline{a})\\
&\quad \wedge\ [\exists a' \in S.\ \sigma(\underline{a}) \wedge I(\underline{a},a') \wedge t_a^{-1}(a',a)] \Rightarrow I(\underline{a},a)\\
&\quad \wedge\ [\sigma(\underline{a}) \wedge I(\underline{a},\bar{a}) \wedge \varepsilon(\bar{a})] \Rightarrow \psi^{-1}(\underline{a},\bar{a})]
\end{aligned}$$

Soit J l'inverse I^{-1} de I. Ces conditions de vérification sont équivalentes à :

$$\begin{aligned}
&[\exists J \in (S\times S \to \{tt, ff\}).\ \forall \underline{a}, a, \bar{a} \in S,\ a \in A.\\
&\quad \sigma(\underline{a}) \Rightarrow J(\underline{a},\underline{a})\\
&\quad \wedge\ [\exists a' \in S.\ t_a^{-1}(a',a) \wedge J(a',\underline{a}) \wedge \sigma(\underline{a})] \Rightarrow J(a,\underline{a})\\
&\quad \wedge\ [\varepsilon(\bar{a}) \wedge J(\bar{a},\underline{a}) \wedge \sigma(\underline{a})] \Rightarrow \psi^{-1}(\underline{a},\bar{a})]
\end{aligned}$$

Renommant les variables muettes $\underline{a}, \bar{a}$ respectivement en $\bar{a}, \underline{a}$, nous obtenons

$$\begin{aligned}
&[\exists J \in (S\times S \to \{tt, ff\}).\ \forall \underline{a}, a, \bar{a} \in S, a \in A.\\
&\quad \sigma(\bar{a}) \Rightarrow J(\bar{a},\bar{a})\\
&\quad \wedge\ [\exists a' \in S.\ t_a^{-1}(a',a) \wedge J(a',\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow J(a,\bar{a})\\
&\quad \wedge\ [\varepsilon(\underline{a}) \wedge J(\underline{a},\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \psi^{-1}(\bar{a},\underline{a})]
\end{aligned}$$

Utilisant la définition des relations inverses, nous venons de démontrer que le principe d'induction $(\mathcal{Y}_-^{-1})$ est correct et sémantiquement complet :

$$\begin{aligned}
&[\exists J \in (S\times S \to \{tt, ff\}).\ \forall \underline{a}, a, \bar{a} \in S, a \in A.\\
&(\mathcal{Y}_-^{-1}.\sigma) \quad \sigma(\bar{a}) \Rightarrow J(\bar{a},\bar{a})\\
&(\mathcal{Y}_-^{-1}.t) \quad \wedge\ [\exists a' \in S.\ t_a(a,a') \wedge J(a',\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow J(a,\bar{a})\\
&(\mathcal{Y}_-^{-1}.\varepsilon) \quad \wedge\ [\varepsilon(\underline{a}) \wedge J(\underline{a},\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \psi(\underline{a},\bar{a})]\\
&\qquad \Leftrightarrow\\
&[\forall p \in \Sigma^{<}\langle S, A, t, tt\rangle, i \in |p|.\ [\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_0,p_i)]
\end{aligned} \qquad (\mathcal{Y}_-^{-1})$$

Ce principe d'induction est à la base de la méthode de Morris-Wegbreit [77] dite "subgoal induction".

Plus généralement, la transformation notée -1 consiste à remplacer la preuve :

$$P(s,A,t,\varepsilon,\sigma,\psi) \Leftrightarrow \exists I.\, Cv[\![s,A,t,\varepsilon,\sigma,\psi]\!](I)$$

par une preuve :

$$P'(s,A,t^{-1},\sigma,\varepsilon,\psi^{-1}) \Leftrightarrow \exists J.\, Cv'[\![s,A,t^{-1},\sigma,\varepsilon,\psi^{-1}]\!](J)$$

(où $J = I^{-1}$) quand :

$$P(s,A,t,\varepsilon,\sigma,\psi) \Leftrightarrow P'(s,A,t^{-1},\sigma,\varepsilon,\psi^{-1})$$

4.2.1.2.4 Transformation contrapositive

Utilisant la propriété que $\neg\neg J = J$, nous pouvons récrire les conditions de vérification $(\overline{\neg J^{-1}})$ comme suit :

$$\begin{aligned}
&[\exists J \in (S\times S \to \{tt,ff\}).\ \forall \underline{a},A,\bar{a} \in S, a \in A.\\
&\quad [\varepsilon(\underline{a}) \wedge \neg\neg J(\underline{a},\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \psi(\underline{a},\bar{a})\\
&\quad \wedge\ [\exists A' \in S.\ t_a(A,A') \wedge \neg\neg J(A',\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \neg\neg J(A,\bar{a})\\
&\quad \wedge\ \sigma(\bar{a}) \Rightarrow \neg\neg J(\bar{a},\bar{a})\ ]
\end{aligned}$$

Posons $\bar{J} = \neg J$ et utilisons le fait que $P \Rightarrow Q$ si et seulement si $\neg Q \Rightarrow \neg P$ dans la condition ci-dessus. Nous obtenons la condition équivalente suivante :

$$\begin{aligned}
&[\exists \bar{J} \in (S\times S \to \{tt,ff\}).\ \forall \underline{a},A',\bar{a} \in S, a \in A.\\
&\quad [\varepsilon(\underline{a}) \wedge \neg\psi(\underline{a},\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \bar{J}(\underline{a},\bar{a})\\
&\quad \wedge\ [\exists A \in S.\ \bar{J}(A,\bar{a}) \wedge t_a(A,A') \wedge \sigma(\bar{a})] \Rightarrow \bar{J}(A',\bar{a})\\
&\quad \wedge\ \sigma(\bar{a}) \Rightarrow \neg\bar{J}(\bar{a},\bar{a})]
\end{aligned}$$

à partir de laquelle nous concluons que le principe d'induction $(\overline{\neg J^{-1}})$ est équivalent à $(\neg J^{-1})$ et est donc correct et sémantiquement complet.

$$\begin{aligned}
&[\exists \bar{J} \in (S\times S \to \{tt,ff\}).\ \forall \underline{a},A,\bar{a} \in S, a \in A.\\
&(\overline{\neg J^{-1}}.\varepsilon) \quad [\varepsilon(\underline{a}) \wedge \neg\psi(\underline{a},\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \bar{J}(\underline{a},\bar{a})\\
&(\overline{\neg J^{-1}}.t) \quad \wedge\ [\exists A' \in S.\ \bar{J}(A',\bar{a}) \wedge t_a(A',A) \wedge \sigma(\bar{a})] \Rightarrow \bar{J}(A,\bar{a})\\
&(\overline{\neg J^{-1}}.\sigma) \quad \wedge\ \sigma(\bar{a}) \Rightarrow \neg\bar{J}(\bar{a},\bar{a})]\\
&\Leftrightarrow\\
&[\forall p \in \Sigma^{<}(S,A,t,tt),\ i \in |p|.\ [\varepsilon(p_0) \wedge \sigma(p_i) \Rightarrow \psi(p_0,p_i)]
\end{aligned} \qquad (\overline{\neg J^{-1}})$$

Nous obtenons ainsi une nouvelle méthode de preuve par l'absurde.

Exemple 4.2.1.2.4-1

En utilisant cette méthode pour démontrer la correction partielle du programme 4.2.1.2-1, nous procédons comme suit :

Cette méthode de preuve étant contrapositive, les invariants locaux décrivent ce qui n'arrivera pas pendant l'exécution du programme :

$$\begin{aligned}
P_1(x,y,q,\bar{x},\bar{q}) &= [(x = \bar{q}\times y + \bar{x}) \Rightarrow (\bar{x} \geq y)]\\
P_2(x,y,q,\bar{x},\bar{q}) &= [(x = (\bar{q}-q)\times y + \bar{x}) \Rightarrow (\bar{x} \geq y)]\\
P_3(x,y,q,\bar{x},\bar{q}) &= [(x = (\bar{q}-q)\times y + \bar{x}) \Rightarrow (\bar{x} \geq y)]\\
P_4(x,y,q,\bar{x},\bar{q}) &= [(x = (\bar{q}-q+1)\times y + \bar{x}) \Rightarrow (\bar{x} \geq y)]\\
P_5(x,y,q,\bar{x},\bar{q}) &= [(x = (\bar{q}-q)\times y + \bar{x}) \Rightarrow (\bar{x} \geq y)]\\
P_6(x,y,q,\bar{x},\bar{q}) &= [((x = (\bar{q}-q)\times y + \bar{x}) \Rightarrow (\bar{x} \geq y)) \wedge (x < y)]
\end{aligned}$$

Soient x,y,q et $\bar{x},\bar{y},\bar{q}$ les valeurs initiales et finales des variables x,y,q du programme. Si le programme n'était pas partiellement correct, alors on aurait $\neg[x = \bar{q}\times y + \bar{x} \wedge \bar{x} < y]$ et donc $P_1(x,y,q,\bar{x},\bar{q})$ serait vrai d'après la première condition de vérification :

$$P_1(x,y,q,\bar{x},\bar{y}) \Leftarrow \neg[x = \bar{q}\times y + \bar{x} \wedge \bar{x} < y]$$

Puis par induction sur le nombre n de pas de calcul durant l'exécution du programme, l'hypothèse que $P_1(x,y,q,\bar{x},\bar{q})$ est vrai et les conditions

de vérification impliquent que les $P_i(x,y,q,\bar{x},\bar{q})$, $i = 1,\dots,6$ sont vrais :

$$P_2(x,y,q,\bar{x},\bar{q}) \Leftarrow [\exists q'.\ P_1(x,y,q',\bar{x},\bar{q}) \wedge q = 0]$$

$$P_3(x,y,q,\bar{x},\bar{q}) \Leftarrow [(P_2(x,y,q,\bar{x},\bar{q}) \vee P_5(x,y,q,\bar{x},\bar{q})) \wedge (x \leq y)]$$

$$P_4(x,y,q,\bar{x},\bar{q}) \Leftarrow [\exists q'.\ P_3(x,y,q',\bar{x},\bar{q}) \wedge q = q'+1]$$

$$P_5(x,y,q,\bar{x},\bar{q}) \Leftarrow [\exists x'.\ P_4(x',y,q,\bar{x},\bar{q}) \wedge x = x'-y]$$

$$P_6(x,y,q,\bar{x},\bar{q}) \Leftarrow [(P_2(x,y,q,\bar{x},\bar{q}) \vee P_5(x,y,q,\bar{x},\bar{q})) \wedge (x > y)]$$

si nous supposons que l'exécution du programme se termine, alors la dernière condition de vérification :

$$\neg P_6(x,y,q,\bar{x},\bar{q}) \Leftarrow [x = \bar{x} \wedge q = \bar{q}]$$

implique que $P_6(x,y,q,\bar{x},\bar{q})$ n'est pas vrai, donc contradiction. Nous avons donc montré par l'absurde que le programme est partiellement correct.

□

4.2.1.2.5 Transformation relation/assertion

Dans le cas d'une assertion invariante

$$\forall p \in \Sigma\langle S, A, t, \Psi\rangle,\ i \in |p|.\ [\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_i)$$

l'invariant Σ utilisé dans le principe d'induction $(\text{-}\mathcal{I})$ peut également être unaire. Nous obtenons le principe d'induction :

$$[\exists i \in (S \to \{\Psi, \not\Psi\}).\ \forall \underline{a}, a, \bar{a} \in S,\ \alpha \in A.$$

$$\begin{array}{ll} (\text{-}i\text{-}.\varepsilon) & \varepsilon(\underline{a}) \Rightarrow i(\underline{a}) \\ (\text{-}i\text{-}.i) & \wedge\ [\exists a' \in S.\ i(a') \wedge t_\alpha(a', a)] \Rightarrow i(a) \\ (\text{-}i\text{-}.\sigma) & \wedge\ [i(\bar{a}) \wedge \sigma(\bar{a})] \Rightarrow \psi(\bar{a}) \end{array} \qquad (\text{-}i\text{-})$$

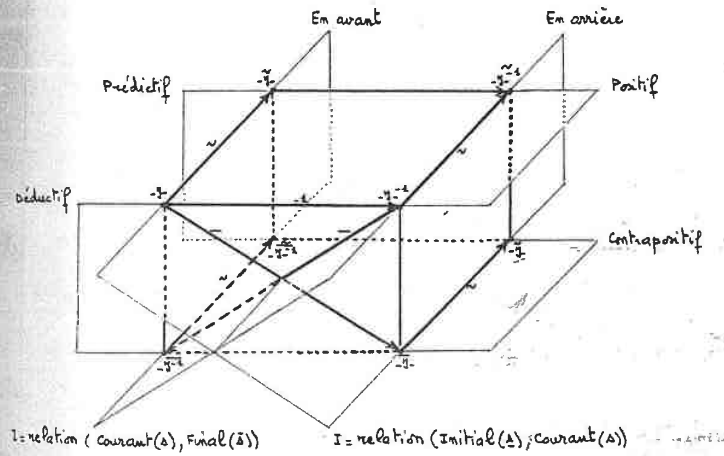
$$\Longleftrightarrow$$

$$[\forall p \in \Sigma\langle S, A, t, \Psi\rangle,\ i \in |p|.\ [\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_i)]$$

4.2.1.3 Principes d'induction dérivés par transformations

En partant du principe d'induction de base $(\text{-}\mathcal{I})$ qui est correct et sémantiquement complet, nous obtenons des principes d'induction dérivés en utilisant les trois transformations $\sim$, $^{-1}$ et $^{-}$ qui conservent la correction et la complétude sémantique, ce qui évite d'avoir à refaire les démonstrations pour chaque principe d'induction dérivé.

Nous pouvons représenter ces dérivations comme suit :



La liste des principes d'induction dérivés est la suivante :

$$[\forall p \in \Sigma\langle S, A, t, \mathbb{t}\rangle, i \in |p|.\ [\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_0, p_i)]$$

$\Longleftrightarrow$

$[\exists I \in (S \times S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$	$[\exists J \in (S \times S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$
$\quad \varepsilon(\underline{a}) \Rightarrow I(\underline{a}, \underline{a})$	$\quad \sigma(\bar{A}) \Rightarrow J(\bar{A}, \bar{A})$
$\quad \wedge\ [\varepsilon(\underline{a}) \wedge I(\underline{a}, A) \Rightarrow \neg[\exists A' \in S.\ t_a(A, A') \wedge \neg I(\underline{a}, A')]$	$\quad \wedge\ [J(A, \bar{A}) \wedge \sigma(\bar{A})] \Rightarrow \neg[\exists A' \in S.\ \neg J(A', \bar{A}) \wedge t_a(A', A)]$
$\quad \wedge\ [\varepsilon(\underline{a}) \wedge I(\underline{a}, \bar{A}) \wedge \sigma(\bar{A}) \Rightarrow \psi(\underline{a}, \bar{A})]$	$\quad \wedge\ [\varepsilon(\underline{a}) \wedge J(\underline{a}, \bar{A}) \wedge \sigma(\bar{A}) \Rightarrow \psi(\underline{a}, \bar{A})]$

$\Longleftrightarrow$

$[\exists I \in (S \times S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$	$[\exists J \in (S \times S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$
$\quad \varepsilon(\underline{a}) \Rightarrow I(\underline{a}, \underline{a})$	$\quad \sigma(\bar{A}) \Rightarrow J(\bar{A}, \bar{A})$
$\quad \wedge\ [\exists A' \in S.\ \varepsilon(\underline{a}) \wedge I(\underline{a}, A') \wedge t_a(A', A)] \Rightarrow I(\underline{a}, A)$	$\quad \wedge\ [\exists A' \in S.\ t_a(A, A') \wedge J(A', \bar{A}) \wedge \sigma(\bar{A})] \Rightarrow J(A, \bar{A})$
$\quad \wedge\ [\varepsilon(\underline{a}) \wedge I(\underline{a}, \bar{A}) \wedge \sigma(\bar{A}) \Rightarrow \psi(\underline{a}, \bar{A})]$	$\quad \wedge\ [\varepsilon(\underline{a}) \wedge J(\underline{a}, \bar{A}) \wedge \sigma(\bar{A}) \Rightarrow \psi(\underline{a}, \bar{A})]$

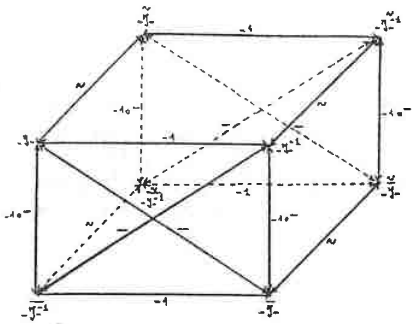
$\Longleftrightarrow$

$[\exists \bar{J} \in (S \times S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$	$[\exists \bar{I} \in (S \times S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$
$\quad [\varepsilon(\underline{a}) \wedge \neg\psi(\underline{a}, \bar{A}) \wedge \sigma(\bar{A})] \Rightarrow \bar{J}(\underline{a}, \bar{A})$	$\quad [\varepsilon(\underline{a}) \wedge \neg\psi(\underline{a}, \bar{A}) \wedge \sigma(\bar{A})] \Rightarrow \bar{I}(\underline{a}, \bar{A})$
$\quad \wedge\ [\bar{J}(A, \bar{A}) \wedge \sigma(\bar{A})] \Rightarrow \neg[\exists A' \in S.\ t_a(A, A') \wedge \neg\bar{J}(A', \bar{A})]$	$\quad \wedge\ [\varepsilon(\underline{a}) \wedge \bar{I}(\underline{a}, A)] \Rightarrow \neg[\exists A' \in S.\ \neg\bar{I}(\underline{a}, A') \wedge t_a(A', A)]$
$\quad \wedge\ \sigma(\bar{A}) \Rightarrow \neg\bar{J}(\bar{A}, \bar{A})]$	$\quad \wedge\ \varepsilon(\underline{a}) \Rightarrow \neg\bar{I}(\underline{a}, \underline{a})]$

$\Longleftrightarrow$

$[\exists \bar{J} \in (S \times S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$	$[\exists \bar{I} \in (S \times S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$
$\quad [\varepsilon(\underline{a}) \wedge \neg\psi(\underline{a}, \bar{A}) \wedge \sigma(\bar{A})] \Rightarrow \bar{J}(\underline{a}, \bar{A})$	$\quad [\varepsilon(\underline{a}) \wedge \neg\psi(\underline{a}, \bar{A}) \wedge \sigma(\bar{A})] \Rightarrow \bar{I}(\underline{a}, \bar{A})$
$\quad \wedge\ [\exists A' \in S.\ \bar{J}(A', \bar{A}) \wedge t_a(A', A) \wedge \sigma(\bar{A})] \Rightarrow \bar{J}(A, \bar{A})$	$\quad \wedge\ [\varepsilon(\underline{a}) \wedge \bar{I}(\underline{a}, A') \wedge t_a(A', A)] \Rightarrow \bar{I}(\underline{a}, A)$
$\quad \wedge\ \sigma(\bar{A}) \Rightarrow \neg\bar{J}(\bar{A}, \bar{A})]$	$\quad \wedge\ \varepsilon(\underline{a}) \Rightarrow \neg\bar{I}(\underline{a}, \underline{a})]$

Des applications supplémentaires des transformations $\sim$, $^{-1}$ et $-$ aux principes d'induction ci-dessus ne donnent pas de nouveaux principes d'induction. Plus généralement, le diagramme suivant commute :

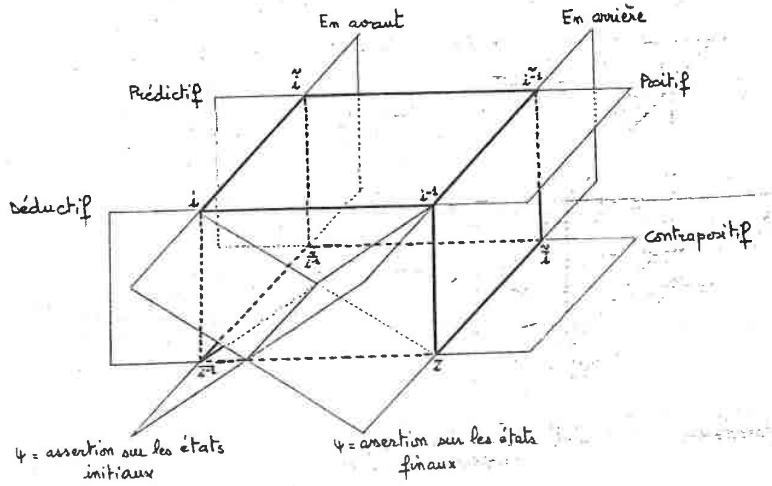


La transformation relation/assertion conduit aux principes d'induction suivants:

$\forall p \in \Sigma\langle S, A, t, \mathbb{t}\rangle, i \in |p|.\ [\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_0)$	$\forall p \in \Sigma\langle S, A, t, \mathbb{t}\rangle, i \in |p|.\ [\varepsilon(p_0) \wedge \sigma(p_i)] \Rightarrow \psi(p_0)$
$\Longleftrightarrow$	$\Longleftrightarrow$
$[\exists i \in (S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$	$[\exists i \in (S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$
$\quad \varepsilon(\underline{a}) \Rightarrow i(\underline{a})$	$\quad \sigma(\bar{A}) \Rightarrow i(\bar{A})$
$\quad \wedge\ [\exists A' \in S.\ i(A') \wedge t_a(A', A)] \Rightarrow i(A)$	$\quad \wedge\ [\exists A' \in S.\ t_a(A, A') \wedge i(A')] \Rightarrow i(A)$
$\quad \wedge\ [i(\bar{A}) \wedge \sigma(\bar{A})] \Rightarrow \psi(\bar{A})]$	$\quad \wedge\ [\varepsilon(\underline{a}) \wedge i(\underline{a})] \Rightarrow \psi(\underline{a})]$
$\Longleftrightarrow$	$\Longleftrightarrow$
$[\exists i \in (S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$	$[\exists i \in (S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$
$\quad \varepsilon(\underline{a}) \Rightarrow i(\underline{a})$	$\quad \sigma(\bar{A}) \Rightarrow i(\bar{A})$
$\quad \wedge\ i(A) \Rightarrow \neg[\exists A' \in S.\ t_a(A, A') \wedge \neg i(A')]$	$\quad \wedge\ i(A) \Rightarrow \neg[\exists A' \in S.\ \neg i(A') \wedge t_a(A', A)]$
$\quad \wedge\ [i(\bar{A}) \wedge \sigma(\bar{A})] \Rightarrow \psi(\bar{A})]$	$\quad \wedge\ [\varepsilon(\underline{a}) \wedge i(\underline{a})] \Rightarrow \psi(\underline{a})]$
$\Longleftrightarrow$	$\Longleftrightarrow$
$[\exists i \in (S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$	$[\exists i \in (S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$
$\quad [\neg\psi(\bar{A}) \wedge \sigma(\bar{A})] \Rightarrow i(\bar{A})$	$\quad [\varepsilon(\underline{a}) \wedge \neg\psi(\underline{a})] \Rightarrow i(\underline{a})$
$\quad \wedge\ [\exists A' \in S.\ t_a(A, A') \wedge i(A')] \Rightarrow i(A)$	$\quad \wedge\ [\exists A' \in S.\ i(A') \wedge t_a(A', A)] \Rightarrow i(A)$
$\quad \wedge\ \varepsilon(\underline{a}) \Rightarrow \neg i(\underline{a})]$	$\quad \wedge\ \sigma(\bar{A}) \Rightarrow \neg i(\bar{A})]$
$\Longleftrightarrow$	$\Longleftrightarrow$
$[\exists i \in (S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$	$[\exists i \in (S \to \{\mathbb{t}, \mathbb{ff}\}).\ \forall \underline{a}, A, \bar{A} \in S,\ a \in A.$
$\quad [\neg\psi(\bar{A}) \wedge \sigma(\bar{A})] \Rightarrow i(\bar{A})$	$\quad [\varepsilon(\underline{a}) \wedge \neg\psi(\underline{a})] \Rightarrow i(\underline{a})$
$\quad \wedge\ i(A) \Rightarrow \neg[\exists A' \in S.\ \neg i(A') \wedge t_a(A', A)]$	$\quad \wedge\ i(A) \Rightarrow \neg[\exists A' \in S.\ t_a(A, A') \wedge \neg i(A')]$
$\quad \wedge\ \varepsilon(\underline{a}) \Rightarrow \neg i(\underline{a})]$	$\quad \wedge\ \sigma(\bar{A}) \Rightarrow \neg i(\bar{A})]$

Notons cependant que tous les principes d'induction relationnels sont équivalents tandis que pour les principes d'induction assertionnels, qui servent à démontrer différentes propositions, seuls ceux d'une même colonne sont équivalents.

Nous pouvons classer ces principes d'induction assertionnels comme suit :



En appliquant la transformation par distinction/confusion des états initiaux, des états finaux et des états initiaux et finaux, nous obtenons 64 autres principes d'induction.

Nous verrons dans la suite que chaque principe d'induction peut donner lieu à un grand nombre de méthodes de preuves.

4.2.1.4 Equivalence forte des principes d'induction dérivés

Tous les principes d'induction dans une même catégorie (c'est-à-dire permettant de démontrer une propriété d'invariance d'un certain type) sont corrects et sémantiquement complets. Par conséquent, s'il existe une preuve par une méthode M alors il existe également une preuve par toute autre méthode M' de la même catégorie.

En ce qui concerne les méthodes de Floyd [67] et Morris-Wegbreit [77], Dijkstra [72] a montré que toute preuve de correction partielle par "subgoal induction" peut se réécrire en une preuve par la méthode de Floyd (et conclut que la méthode de Morris-Wegbreit est inutile). Morris et Wegbreit avaient démontré la réciproque. Ce dernier résultat est généralisé dans Cousot-R [81] aux principes d'induction (-$\mathcal{Y}$-) et (-$\mathcal{Y}^{-1}$-) comme suit :

Théorème 4.2.1.4-1

Les conditions .$\mathcal{S}$, .i et .ε de (-$\mathcal{Y}^{-1}$-) et .ε, .i et .$\mathcal{S}$ de (-$\mathcal{Y}$-) sont équivalentes en définissant :

$I(\underline{a},\bar{a}) = [\forall \bar{z} \in S.\ (J(\underline{a},\bar{z}) \wedge \mathcal{S}(\bar{z})) \Rightarrow \psi(\underline{a},\bar{z})]$

$J(\underline{a},\bar{a}) = [\forall \underline{z} \in S.\ (\varepsilon(\underline{z}) \wedge I(\underline{z},\bar{a})) \Rightarrow \psi(\underline{z},\bar{a})]$

et la remarque de Dijkstra se généralise de la même manière :

Théorème 4.2.1.4-2

Les conditions .$\mathcal{S}$, .i et .ε de (-$\mathcal{Y}^{-1}$-) et .ε, .i et .$\mathcal{S}$ de (-$\mathcal{Y}$-) sont équivalentes en définissant :

$I(\underline{a},\bar{a}) = [\varepsilon(\underline{a}) \wedge \forall \bar{z} \in S.\ (J(\underline{a},\bar{z}) \wedge \mathcal{S}(\bar{z})) \Rightarrow J(\underline{a},\bar{z})]$

$J(\underline{a},\bar{a}) = [\mathcal{S}(\bar{a}) \wedge \forall \underline{z} \in S.\ (\varepsilon(\underline{z}) \wedge I(\underline{z},\bar{a})) \Rightarrow I(\underline{z},\bar{a})]$

Démonstration

- Si J satisfait les conditions .s, .i et .e de $(\mathcal{Y}^{-1})$ alors nous avons :

$(\mathcal{Y}^{-1}.e)$ $\quad \varepsilon(\underline{a}) \Rightarrow [\varepsilon(\underline{a}) \wedge \forall \bar{a} \in S.((J(\underline{a},\bar{a}) \wedge s(\bar{a})) \Rightarrow J(\underline{a},\bar{a}))] \Rightarrow I(\underline{a},\underline{a})$

$(\mathcal{Y}^{-1}.i)$ $\quad [I(\underline{a},\underline{a}') \wedge t_a(a',a)] \Rightarrow [\varepsilon(\underline{a}) \wedge \forall \bar{a} \in S.((J(a',\bar{a}) \wedge s(\bar{a})) \Rightarrow J(\underline{a},\bar{a})) \wedge t(a',a)] \Rightarrow$

$[\varepsilon(\underline{a}) \wedge \forall \bar{a} \in S.((J(a',\bar{a}) \wedge s(\bar{a})) \Rightarrow J(\underline{a},\bar{a})) \wedge \forall \bar{a} \in S.((J(a,\bar{a}) \wedge s(\bar{a})) \Rightarrow J(a',\bar{a}))]$ (car

$t_a(a',a) \Rightarrow [(J(a',\bar{a}) \wedge s(\bar{a})) \Rightarrow J(a',\bar{a})]$ d'après $(\mathcal{Y}^{-1}.i)$) $\Rightarrow$

$[\varepsilon(\underline{a}) \wedge \forall \bar{a} \in S.((J(a,\bar{a}) \wedge s(\bar{a})) \Rightarrow J(\underline{a},\bar{a}))] \Rightarrow I(\underline{a},a)$

$(\mathcal{Y}^{-1}.s)$ $\quad [I(\underline{a},\bar{a}) \wedge s(\bar{a})] \Rightarrow [\varepsilon(\underline{a}) \wedge \forall \bar{a} \in S.((J(\bar{a},\bar{a}) \wedge s(\bar{a})) \Rightarrow J(\underline{a},\bar{a}) \wedge J(\bar{a},\bar{a}) \wedge s(\bar{a})]$

(d'après la définition de I et $(\mathcal{Y}^{-1}.s)$) $\Rightarrow [\varepsilon(\underline{a}) \wedge J(\underline{a},\bar{a}) \wedge s(\bar{a}) \Rightarrow$

$\psi(\underline{a},\bar{a})$ (d'après $(\mathcal{Y}^{-1}.e)$)).

- Pour la réciproque, nous utilisons la transformation -1.

□

- Ces résultats se généralisent aisément à tous les principes d'induction dérivés de $(\mathcal{Y})$ par les transformations $\sim$, $\bar{\ }$ et $^{-1}$:

Ceci signifie que si une preuve a été faite en utilisant une méthode M (c'est-à-dire que nous avons trouvé un invariant I satisfaisant les conditions de vérification CV) et M' est une autre méthode qui nécessite la découverte d'un invariant I' satisfaisant CV', alors nous pouvons définir formellement I' en fonction de I de sorte que CV(I) implique CV'(I') :

- Si $M' = \tilde{M}$ alors $I' = I$
- Si $M' = \bar{M}$ alors $I' = \neg I$
- Si $M' = M^{-1}$ et M est une méthode positive où I relie un état initial à un état courant alors $I'(a,\bar{a}) = [\forall \underline{a} \in S.\ (\varepsilon(\underline{a}) \wedge I(\underline{a},a)) \Rightarrow \psi(\underline{a},\bar{a})]$
- Si $M' = M^{-1}$ et M est une méthode contrapositive où I relie un état initial à un état courant alors $I'(a,\bar{a}) = [\exists \underline{a} \in S.\ \varepsilon(\underline{a}) \wedge I(\underline{a},a) \wedge \neg \psi(\underline{a},\bar{a})]$
- Si $M' = M^{-1}$ et M est une méthode positive où I relie un état courant

- Si $M' = M^{-1}$ et M est une méthode contrapositive où I relie un état courant à un état final alors $I'(\underline{a},a) = [\exists \bar{a} \in S.\ I(a,\bar{a}) \wedge s(\bar{a}) \wedge \neg \psi(\underline{a},\bar{a})]$

- De même quand ψ est une assertion, alors une preuve par un principe d'induction $(\mathcal{M})$ peut également se faire par le principe d'induction $(\mathcal{M})$ correspondant en choisissant

$I(\underline{a},a) = [\varepsilon(\underline{a}) \wedge i(a)]$ (ou bien $I(a,\bar{a}) = [i(a) \wedge s(\bar{a})]$)

La technique des variables auxiliaires (qui permet de récrire une preuve d'invariance d'une relation par une méthode relationnelle en une preuve assertionnelle pour un programme transformé où les valeurs initiales des variables sont mémorisées dans des variables auxiliaires) se généralise comme suit :

Si $[\forall p \in \Sigma^{<S,A,t,\varepsilon>}, i \in |p|.\ s(p_i) \Rightarrow \psi(p_0,p_i)]$ a été démontré par la méthode $(\mathcal{M})$ utilisant l'invariant relationnel I, alors nous pouvons faire la même démonstration par la méthode $(\mathcal{m})$ en posant :

- si I relie un état initial à un état courant,
  $S' = S \times S$, $A' = A$, $\varepsilon'(\langle \underline{a},a \rangle) = [\varepsilon(\underline{a}) \wedge \underline{a} = a]$, $t'_a(\langle \underline{a},a \rangle, \langle \underline{a}',a' \rangle) = [\underline{a} = \underline{a}' \wedge t_a(a,a')]$, $s'(\langle \underline{a},a \rangle) = s(a)$ avec l'invariant unaire $i(\langle \underline{a},a \rangle) = I(\underline{a},a)$.
- si I relie un état courant à un état final,
  $S' = S \times S$, $A' = A$, $s'(\langle a,\bar{a} \rangle) = [a = \bar{a} \wedge s(\bar{a})]$, $t'_a(\langle a,\bar{a} \rangle, \langle a',\bar{a}' \rangle) = [t_a(a,a') \wedge \bar{a} = \bar{a}']$, $\varepsilon(\langle a,\bar{a} \rangle) = \varepsilon(a)$ avec l'invariant unaire $i(\langle a,\bar{a} \rangle) = I(a,\bar{a})$.

4.2.2 PRINCIPES D'INDUCTION POUR UNE SEMANTIQUE NON CLOSE FERMEE PAR FUSIONS

Dans le cas d'une sémantique $\langle S,A,\Sigma\rangle$ fermée par fusions (c'est-à-dire $\underset{\sim}{Efus}(\langle S,A,\Sigma\rangle) = \langle S,A,\Sigma\rangle$), l'invariance de ψ sous condition ϕ pour $\langle S,A,\Sigma\rangle$ est équivalente, d'après les théorèmes 4.1.3∨3.7 et 4.1.3∨4 à l'invariance de ψ sous condition ϕ pour $\underset{\sim}{Rtran}(\langle S,A,\Sigma\rangle)$. Ceci montre que dans ce cas il est toujours correct et sémantiquement complet de faire les preuves d'invariance en utilisant l'un quelconque des principes d'induction du paragraphe 4.2.1 pour le système de transition engendré par cette sémantique $\langle S,A,\Sigma\rangle$. C'est le cas en particulier (et d'après 4.1.3∨3.8 et 4.1.3∨3.9) pour le langage que nous avons considéré en 2.8.

4.2.3 PRINCIPES D'INDUCTION POUR UNE SEMANTIQUE (NON CLOSE ET) NON FERMEE PAR FUSIONS

4.2.3.1 Principes d'induction pour une sémantique non fermée par fusions définie par une condition sur les préfixes des traces engendrées par un système de transition

Dans le cas d'une sémantique $\langle S,A,\Sigma\rangle$ non fermée par fusions, définie par une condition sur les préfixes des traces engendrées par un système de transition $\langle S,A,t,\varepsilon\rangle$:

$$\langle S,A,\Sigma\rangle = \langle S,A,\{p\in\Sigma' : \langle S,A,\Sigma'\rangle = \underset{\sim}{Pref}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle) \wedge Cp(p)\}\rangle$$

ψ est invariante sous condition ϕ pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ si et seulement si ψ est invariante sous condition ϕ pour $\underset{\sim}{Pref}(\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)$ (cf. 4.1.3∨3.1

Il est donc toujours correct de faire une preuve d'invariance relative à $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ (pour laquelle nous pourrons toujours utiliser l'un quelconque des principes d'induction du paragraphes 4.2.1) pour conclure relativement à $\langle S,A,\Sigma\rangle$. Cette approche n'est pas toujours complète (comme en témoigne 4.1.3∨1).

Pour être complet nous proposons d'utiliser le principe d'induction suivant (ou ses transformés comme en 4.2.1.2) qui utilise des variables auxiliaires (dans la preuve mais pas dans le programme) permettant de cumuler des histoires. Ce principe d'induction est directement inspiré d'un principe d'induction similaire proposé dans Cousot-Cousot [82] pour des preuves de fatalité. Il peut se motiver comme suit :

Pour démontrer que :

ψ est invariante pour $\langle S,A,\Sigma\rangle$

il faut et il suffit en général de démontrer l'invariance d'une propriété plus forte :

$$\exists J\in(S\times S\to\{tt,ff\}).\ [J \text{ est invariante pour } \langle S,A,\Sigma\rangle \wedge J\Rightarrow\psi]$$

d'après le théorème 4.1.3∨2, il suffit de démontrer l'invariance pour les préfixes finis :

$$\exists J\in(S\times S\to\{tt,ff\}).\ [J \text{ est invariante pour } \underset{\sim}{Pref}^{<\omega}\langle S,A,\Sigma\rangle \wedge J\Rightarrow\psi]$$

En écrivant $\forall p\in\underset{\sim}{Pref}^{<\omega}\langle S,A,\Sigma\rangle.\ Q(p)$ pour abréger $\forall p.[(\exists\Sigma'.[\langle S,A,\Sigma'\rangle = \underset{\sim}{Pref}^{<\omega}\langle S,A,\Sigma\rangle \wedge p\in\Sigma'])\Rightarrow Q(p)]$, ceci s'équivaut d'après 3.2.2 à :

$$\exists J\in(S\times S\to\{tt,ff\}).\ [\forall p\in\underset{\sim}{Pref}^{<\omega}\langle S,A,\Sigma\rangle, i\in|p|.\ J(p_0,p_i) \wedge J\Rightarrow\psi]$$

En procédant par induction sur i, nous obtenons :

$$\exists J\in(S\times S\to\{tt,ff\}).\ \forall p\in\underset{\sim}{Pref}^{<\omega}\langle S,A,\Sigma\rangle.$$
$$J(p_0,p_0)$$
$$\wedge\ \forall i\in(|p|\vee 0).\ J(p_0,p_{i-1})\Rightarrow J(p_0,p_i)$$
$$\wedge\ J\Rightarrow\psi$$

Il suffit de faire la preuve pour $i=|p|$ car si $p\in\underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)$ et $i\in|p|$ alors $p\downarrow i\in\underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)$:

$$\exists J \in (S\times S \to \{tt, ff\}).\ \forall p \in \underset{\sim}{Pref}^{<\omega}(\langle S, A, \Sigma\rangle).$$

$$J(p_0, p_0)$$
$$\wedge\ [|p|>1 \wedge J(p_0, p_{|p|-1})] \Rightarrow J(p_0, p_{|p|})$$
$$\wedge\ J \Rightarrow \Psi$$

La condition $(p \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle) \wedge |p|>1)$ équivaut à $(\exists p' \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle), a\in A, s\in S.\ p = p' \xrightarrow{a} \langle s\rangle \wedge p \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle))$, ce qui donne :

$$\exists J \in (S\times S \to \{tt, ff\}).$$
$$\forall p \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle).\ J(p_0,p_0)$$
$$\wedge\ \forall p' \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle),\ a\in A,\ s\in S.$$
$$[J(p'_0, p'_{|p'|-1}) \wedge p' \xrightarrow{a} \langle s\rangle \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)] \Rightarrow J(p'_0, s)$$
$$\wedge\ J \Rightarrow \Psi$$

En introduisant le système de transition $\langle S, A, t, \varepsilon\rangle$ engendré par $\langle S,A,\Sigma\rangle$, ceci peut s'écrire :

$$\exists J \in (S\times S \to \{tt, ff\}).$$
$$\forall \underline{s} \in S.\ \varepsilon(\underline{s}) \Rightarrow J(\underline{s}, \underline{s})$$
$$\wedge\ \forall p \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle),\ a\in A,\ s\in S.$$
$$[J(p_0, p_{|p|}) \wedge t_a(p_{|p|}, s) \wedge p \xrightarrow{a} \langle s\rangle \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)] \Rightarrow J(p_0, s)$$
$$\wedge\ J \Rightarrow \Psi$$

ou encore

$$\exists J \in (S\times S \to \{tt, ff\}).$$
$$\forall \underline{s}, \underline{s}', s, \bar{s} \in S,\ p', \bar{p} \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle),\ a\in A.$$
$$\varepsilon(\underline{s}) \Rightarrow [\exists p \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle).\ [p_0 = \underline{s} \wedge J(\underline{s},\underline{s})]]$$
$$\wedge\ [[p'_0 = \underline{s} \wedge J(\underline{s}, \underline{s}')] \wedge t_a(\underline{s}', s) \wedge [s' = p'_{|p'|} \wedge p' \xrightarrow{a} \langle s\rangle \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)]]$$
$$\Rightarrow [p'_0 = \underline{s} \wedge J(\underline{s}, s)]$$
$$\wedge\ [\bar{p}_0 = \underline{s} \wedge J(\underline{s}, \bar{s})] \Rightarrow \Psi(\underline{s}, \bar{s})$$

En posant

$H = \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)$

$F(\underline{s}, p) = [p_0 = \underline{s}]$

$R(s', p', a, s) = [s' = p'_{|p'|} \wedge p' \xrightarrow{a} \langle s\rangle \in \underset{\sim}{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)]$

$N(s', p', a, s, p) = [s' = p'_{|p'|} \wedge p = p' \xrightarrow{a} \langle s\rangle]$

$I(\underline{s}, s, p) = [p_0 = \underline{s} \wedge J(\underline{s}, s)]$

nous obtenons

$$\exists I \in (S\times S\times H \to \{tt, ff\}).\ \forall \underline{s}, s, s', \bar{s} \in S,\ p', \bar{p} \in H,\ a \in A.$$
$$\varepsilon(\underline{s}) \Rightarrow [\exists p \in H.\ F(\underline{s}, p) \wedge I(\underline{s}, \underline{s}, p)]$$
$$\wedge\ [I(\underline{s}, s', p') \wedge t_a(s', s) \wedge R(s', p', a, s)] \Rightarrow [\exists p \in H.\ N(s', p', a, s, p) \wedge I(\underline{s}, s, p)]$$
$$\wedge\ I(\underline{s}, \bar{s}, \bar{p}) \Rightarrow \Psi(\underline{s}, \bar{s})$$

En pratique, nous évitons de raisonner sur les (préfixes des) traces de Σ en utilisant des "histoires" que nous interprétons comme des fonctions des préfixes des traces cumulant l'essentiel de l'information contenue dans ces préfixes. En comprenant H comme un ensemble d'histoires, $F(\underline{s}, h)$ comme l'assertion que l'histoire h commence en l'état $\underline{s}$, $R(s', h', a, s)$ comme l'assertion qu'au terme de l'histoire h' l'état s' a un successeur s par l'action a et $N(s', h', a, s, h)$ comme l'assertion que l'information, qu'au terme de l'histoire h' l'état s' a un successeur s par l'action a, se cumule dans l'histoire h, nous obtenons le principe d'induction suivant :

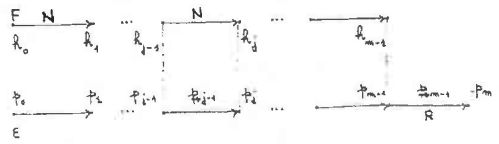
Théorème 4.2.3.1 ∨1

$$[[\exists H,\ F\in(S\times H\to\{tt,ff\}),\ R\in(S\times H\times A\times S\to\{tt,ff\}),$$
$$N\in(S\times H\times A\times S\times H\to\{tt,ff\}).$$
$$\text{Hüt}\langle S,A,\Sigma\rangle(H,F,N,R)$$
$$\wedge\ (\exists I\in(S\times S\times H\to\{tt,ff\}).\ \forall \underline{s},\underline{s}',s,\bar{s}\in S,\ \underline{h}',\bar{h}\in H,\ a\in A.$$
$$\varepsilon(\underline{s})\Rightarrow[\exists \underline{h}\in H.\ F(\underline{s},\underline{h})\wedge I(\underline{s},\underline{s},\underline{h})]$$
$$\wedge\ [I(\underline{s},\underline{s}',\underline{h}')\wedge t_a(\underline{s}',s)\wedge R(\underline{s}',\underline{h}',a,s)]$$
$$\Rightarrow[\exists h\in H.\ N(\underline{s}',\underline{h}',a,s,h)\wedge I(\underline{s},s,h)]$$
$$\wedge\ I(\underline{s},\bar{s},\bar{h})\Rightarrow\Psi(\underline{s},\bar{s}))]\qquad(\text{-}\uparrow\Psi)$$
$$\Longleftrightarrow$$
$$[\forall p\in\Sigma,\ i\in|p|\cdot\ \Psi(p_0,p_i)]$$

où

$$\text{Hüt}\langle S,A,\Sigma\rangle(H,F,N,R)=$$
$$[\forall p\in\Sigma,\ m\in\omega.\ (1\leqslant m<|p|)\Rightarrow$$
$$(\forall h\in(m\to H).\ [F(p_0,h_0)\wedge\forall j\in(m\vee 0).\ N(p_{j-1},h_{j-1},p_{j-1},p_j,h_j)]$$
$$\Rightarrow R(p_{m-1},h_{m-1},p_{m-1},p_m))]$$

une formule qui se comprend mieux à l'aide du schéma suivant :



Démonstration

(⇒) Soit $p\in\Sigma$. Nous démontrons par induction sur $m\in(|p|\vee 0)$ qu'il existe $h\in(m\to H)$ tel que $\forall j\in m.\ I(p_0,p_j,h_j)\wedge F(p_0,h_0)\wedge\forall j\in(m\vee 0).N(p_{j-1},h_{j-1},p_{j-1},p_j,h_j)$. En effet pour $m=1$, nous avons $\varepsilon(p_0)\Rightarrow\exists h_0\in H.\ F(p_0,h_0)\wedge I(p_0,p_0,h_0)$. Si par hypothèse d'induction le lemme est vrai pour $m-1\geqslant 1$ et $m\in|p|$ alors nous avons $I(p_0,p_{m-2},h_{m-2})$ et d'après la condition Hüt, $R(p_{m-2},h_{m-2},p_{m-2},p_{m-1})$ ce qui avec $t_{p_{m-2}}(p_{m-2},p_{m-1})$ implique que $\exists h_{m-1}\in H.\ N(p_{m-2},h_{m-2},p_{m-2},p_{m-1},h_{m-1})\wedge I(p_0,p_{m-1},h_{m-1})$. Nous en déduisons que $\forall j\in|p|.\ I(p_0,p_j,h_j)$ et donc $\Psi(p_0,p_j)$.

(⇐) Si $\forall p\in\Sigma,\ i\in|p|.\ \Psi(p_0,p_i)$ alors nous pouvons choisir :

$H=\Sigma^*\times\omega$, $F(a,\langle T,m\rangle)=[T=\{p\in\Sigma : p_0=a\}\neq\phi\wedge m=0]$, $N(a',\langle T',m'\rangle,a,s,\langle T,m\rangle)=$
$[m=m'+1\wedge T=\{p\in T' : m\in|p|\wedge p_{m'}=a'\wedge p_m=s\}\neq\phi]$, $R(a',\langle T',m'\rangle,a,s)=$
$[\exists p\in T'.\ (m'+1)\in|p|\wedge p_{m'}=a'\wedge p_{m'+1}=a\wedge p_{m'+1}=s]$ et $I(\underline{s},s,\langle T,m\rangle)=$
$[T=\{p\in\Sigma : m\in|p|\wedge p_0=\underline{s}\wedge p_m=s\}\neq\phi]$.

□

Observons que (-↑Ψ) est une généralisation de (-Ψ) que nous retrouvons en choisissant $F=tt$, $R=tt$, $N=tt$ et $H=1$.

4.2.3.2 Principes d'induction pour une sémantique non fermée par fusions définie par concordance avec une sémantique close

Il est toujours possible de définir une sémantique non close $\langle S,A,\Sigma\rangle$ par concordance avec une sémantique close (engendrée par un système de transition $\langle S^\#,A^\#,t^\#,\varepsilon^\#\rangle$) à une fonction $f_s^\#\in(S^\#\to S)$ entre états près (cf. 2.7.2.2 ∨1) :

$$\langle S,A,\Sigma\rangle = \sim\langle f_s^\#\rangle(\langle S^\#,A^\#,\Sigma\langle S^\#,A^\#,t^\#,\varepsilon^\#\rangle\rangle)$$

Par conséquent, pour démontrer que $\psi \in (S \times S \to \{tt, ff\})$ est invariante pour $\langle S, A, \Sigma \rangle$ il est toujours correct et possible (d'après 4.1.3.v3 et 4.1.3.v4) d'utiliser l'un quelconque des principes d'induction du paragraphe 4.2.1 pour $\langle S^\#, A^\#, t^\#, \varepsilon^\# \rangle$ et $\psi^\#$ définie par $\psi^\#(a_0, a_1) = \psi(f_a^\#(a_0), f_a^\#(a_1))$.

Par exemple, en utilisant le principe d'induction $(-\mathcal{Y})$, nous obtenons :

$$
\begin{array}{l}
[\ \exists S^\#, A^\#, t^\# \in (A^\# \to (S^\# \times S^\# \to \{tt, ff\})),\ \varepsilon^\# \in (S^\# \to \{tt, ff\}),\ e^\# \in (S^\# \to S).\\
\quad \exists I \in (S^\# \times S^\# \to \{tt, ff\}).\ \forall \underline{a}, a, \underline{a}' \in S^\#,\ a \in A^\#.\\
\qquad \langle S, A, \Sigma \rangle = \mathcal{U}\langle f_a^\# \rangle(\langle S^\#, A^\#, \Sigma\langle S^\#, A^\#, t^\#, \varepsilon^\# \rangle \rangle) \\
\qquad \wedge\ \varepsilon^\#(\underline{a}) \Rightarrow I(\underline{a}, \underline{a}) \qquad\qquad (-\mathcal{Y}^\#)\\
\qquad \wedge\ [\exists a' \in S^\#.\ I(\underline{a}, a') \wedge t_a^\#(a', a)] \Rightarrow I(\underline{a}, a)\\
\qquad \wedge\ I(\underline{a}, \underline{a}) \Rightarrow \psi(f_a^\#(\underline{a}), f_a^\#(\underline{a}))\ ]\\
\Longleftrightarrow\\
[\forall p \in \Sigma,\ i \in |p|.\ \psi(p_0, p_i)\ ]
\end{array}
$$

4.2.3.3 Equivalence forte des deux principes d'induction $(-\uparrow\mathcal{Y})$ et $(-\mathcal{Y}^\#)$

Théorème 4.2.3.3.v1

Toute preuve d'invariance de ψ pour une sémantique quelconque $\langle S, A, \Sigma \rangle$ utilisant le principe d'induction $(-\mathcal{Y}^\#)$ peut se recrire en une preuve utilisant $(-\uparrow\mathcal{Y})$, et réciproquement.

Démonstration

- Ayant trouvé $S^\#, A^\#, t^\#, \varepsilon^\#, f_a^\#$ et $I^\#$ satisfaisant les conditions données en $(-\mathcal{Y}^\#)$, nous pouvons toujours recrire cette preuve d'invariance en une preuve utilisant le principe d'induction $(-\uparrow\mathcal{Y})$ en posant :

$H = S^\# \times S^\#$

$F(\underline{a}, \langle \underline{a}^\#, a^\# \rangle) = [\varepsilon^\#(\underline{a}^\#) \wedge a^\# = \underline{a}^\# \wedge \underline{a} = f_a^\#(\underline{a}^\#)]$

$R(a', \langle \underline{a}^{\#\prime}, a^{\#\prime} \rangle, a, a) = [a' = f_a^\#(a^{\#\prime}) \wedge \exists a^\# \in S^\#.\ (t_a^\#(a^{\#\prime}, a^\#) \wedge a = f_a^\#(a^\#))]$

$N(a', \langle \underline{a}^{\#\prime}, a^{\#\prime} \rangle, a, a, \langle \underline{a}^\#, a^\# \rangle) = [a' = f_a^\#(a^{\#\prime}) \wedge t_a^\#(a^{\#\prime}, a^\#) \wedge a = f_a^\#(a^\#)]$

$I(\underline{a}, a, \langle \underline{a}^\#, a^\# \rangle) = [I^\#(\underline{a}^\#, a^\#) \wedge \underline{a} = f_a^\#(\underline{a}^\#) \wedge a = f_a^\#(a^\#)]$

En effet, nous avons bien :

- $\mathrm{Hist}\langle S, A, \Sigma \rangle(H, F, N, R)$ car si nous avons $p \in \Sigma$, $m \in \omega$, $\forall i \in m.\ \langle \underline{a}_i^\#, a_i^\# \rangle \in H$ tels que $1 < m < |p|$, $F(p_0, \langle \underline{a}_0^\#, a_0^\# \rangle)$ et $\forall j \in (m \vee 0).\ N(p_{j-1}, \langle \underline{a}_{j-1}^\#, a_{j-1}^\# \rangle, p_{j-1}, p_j, \langle \underline{a}_j^\#, a_j^\# \rangle)$ alors la séquence $a_0^\# \xrightarrow{p} \cdots \xrightarrow{p_{m-1}} a_{m-1}^\#$ est préfixe d'une trace $p^\# \in \Sigma\langle S^\#, A^\#, t^\#, \varepsilon^\# \rangle$ telle que $p = f_a^\#(p^\#)$. Par conséquent comme $m < |p| = |p^\#|$, nous avons $p_{m-1} = f_a^\#(p_{m-1}^\#) = f_a^\#(a_{m-1}^\#)$ et $t_{a_{m-1}^\#}^\#(p_{m-1}^\#, p_m^\#)$ donc $t_{a_{m-1}^\#}^\#(a_{m-1}^\#, p_m^\#)$ et $p_m = f_a^\#(p_m^\#)$. soit $R(p_{m-1}, \langle \underline{a}_{m-1}^\#, a_{m-1}^\# \rangle, p_{m-1}, p_m)$.

- Si $\varepsilon(\underline{a})$ est vrai, alors $\exists \underline{a}^\# \in S^\#.(\varepsilon^\#(\underline{a}^\#) \wedge \underline{a} = f_a^\#(\underline{a}^\#))$ donc $F(\underline{a}, \langle \underline{a}^\#, \underline{a}^\# \rangle)$ et $I^\#(\underline{a}^\#, \underline{a}^\#)$ donc $F(\underline{a}, \langle \underline{a}^\#, \underline{a}^\# \rangle) \wedge I(\underline{a}, a, \langle \underline{a}^\#, \underline{a}^\# \rangle)$.

- Si $I(\underline{a}, a', \langle \underline{a}^\#, a^{\#\prime} \rangle) \wedge t_a(a', a) \wedge R(a', \langle \underline{a}^\#, a^{\#\prime} \rangle, a, a)$ est vrai, alors $\underline{a} = f_a^\#(\underline{a}^\#) \wedge I^\#(\underline{a}^\#, a^{\#\prime}) \wedge t_a(a^{\#\prime}, a^\#) \wedge a = f_a^\#(a^\#)$ impliquent $N(a', \langle \underline{a}^\#, a^{\#\prime} \rangle, a, a, \langle \underline{a}^\#, a^\# \rangle)$ et $I^\#(\underline{a}^\#, a^\#)$ donc $I(\underline{a}, a, \langle \underline{a}^\#, a^\# \rangle)$.

- Si $I(\underline{a}, a, \langle \underline{a}^\#, a^\# \rangle)$ est vrai alors nous avons $I^\#(\underline{a}^\#, a^\#)$ qui implique $\psi(f_a^\#(\underline{a}^\#), f_a^\#(a^\#)) = \psi(\underline{a}, a)$.

- Réciproquement, ayant trouvé E, H, F, R, N et I satisfaisant les conditions de $(-\uparrow\mathcal{Y})$, nous pouvons toujours recrire cette preuve d'invariance en une preuve utilisant le principe d'induction $(-\mathcal{Y}^\#)$ comme suit :

Etant donnée $p \in \Sigma$, nous construisons $p^\# \in \Sigma\langle S\times H, A\rangle$ comme suit. Comme $\varepsilon(p_0) \Rightarrow \exists h_0 \in H.\ F(p_0,h_0) \wedge I(p_0,p_0,h_0)$, nous choisissons $p_0^\# = \langle p_0, h_0\rangle$. Nous avons aussi $R(p_0,h_0,p_0,p_1)$ et $I(p_0,p_0,h_0) \wedge t_{p_0}(p_0,p_1)$ qui impliquent $\exists h_1 \in H.\ N(p_0,h_0,p_0,p_1,h_1) \wedge I(p_0,p_1,h_1)$. Nous posons alors $p_0^\# = p_0$ et $p_1^\# = \langle p_1, h_1\rangle$. Ayant construit $p_j^\#$ pour $j = 0,\dots,m-2$ et $p_j^\# = \langle p_j, h_j\rangle$ pour $j = 0,\dots,m-1$ avec $1 < m < |p|$ tel que $F(p_0,h_0) \wedge \forall j \in (m\vee 0).\ N(p_{j-1},h_{j-1},p_{j-1},p_j,h_j) \wedge \forall j \in m.\ I(p_0,p_j,h_j)$ nous avons $t_{p_{m-1}}(p_{m-1},p_m)$ et $R(p_{m-1},h_{m-1},p_{m-1},p_m)$ et donc $\exists h_m \in H.\ N(p_{m-1},h_{m-1},p_{m-1},p_m,h_m) \wedge I(p_0,p_m,h_m)$. Nous posons alors $p_{m-1}^\# = p_{m-1}$ et $p_m^\# = \langle p_m, h_m\rangle$.

Posons maintenant :

$\Sigma^\# = \{p^\# :\ p \in \Sigma\}$

$S^\# = S\times 2^{\Sigma^\#}\times\omega$

$A^\# = A$

$\varepsilon^\#(\langle A,T,m\rangle) = [T = \{p^\# \in \Sigma^\# :\ p_0^\#(0) = A\} \neq \phi \wedge m = 0]$

$t_a^\#(\langle A',T',m'\rangle, \langle A,T,m\rangle) = [m = m'+1 \wedge T = \{p^\# \in T' :\ m \in |p^\#| \wedge p_m^\#(0) = A' \wedge p_{m'}^\# = a \wedge p_m^\#(0) = A\} \neq \phi]$

$I^\#(\langle \underline{A},\underline{T},\underline{m}\rangle, \langle A,T,m\rangle) = [T = \{p^\# \in \Sigma^\# :\ m \in |p^\#| \wedge p_0^\#(0) = \underline{A} \wedge p_m^\#(0) = A\} \neq \phi]$

$f_A^\#(\langle A,T,m\rangle) = A$

alors, nous avons bien :

- $\langle S,A,\Sigma\rangle = \approx\langle f_A^\#\rangle(\langle S^\#, A^\#, \Sigma\langle S^\#,A^\#,t^\#,\varepsilon^\#\rangle\rangle)$ par construction.
- $\varepsilon^\#(\langle \underline{A},\underline{T},\underline{m}\rangle) = [T = \{p^\# \in \Sigma^\# :\ p_0^\#(0) = \underline{A}\} \neq \phi \wedge \underline{m} = 0] \Rightarrow I^\#(\langle \underline{A},\underline{T},\underline{m}\rangle, \langle \underline{A},\underline{T},\underline{m}\rangle)$.
- $\exists \langle A',T',m'\rangle \in S^\#.\ (I^\#(\langle \underline{A},\underline{T},\underline{m}\rangle, \langle A',T',m'\rangle) \wedge t_a^\#(\langle A',T',m'\rangle, \langle A,T,m\rangle)) \Leftrightarrow$
$\exists \langle A',T',m'\rangle \in S^\#.\ [T' = \{p^\# \in \Sigma^\# :\ m' \in |p^\#| \wedge p_0^\#(0) = \underline{A} \wedge p_{m'}^\#(0) = A'\} \neq \phi \wedge m = m'+1 \wedge$
$T = \{p^\# \in T' :\ m \in |p^\#| \wedge p_{m'}^\#(0) = A' \wedge p_{m'}^\# = a \wedge p_m^\#(0) = A\} \neq \phi] \Rightarrow [T = \{p^\# \in \Sigma :\ m \in |p^\#| \wedge$
$p_0^\#(0) = \underline{A} \wedge p_m^\#(0) = A\} \neq \phi] = I^\#(\langle \underline{A},\underline{T},\underline{m}\rangle, \langle A,T,m\rangle)$.
- $I^\#(\langle \underline{A},\underline{T},\underline{m}\rangle, \langle \bar{A},\bar{T},\bar{m}\rangle) = [T = \{p^\# \in \Sigma^\# :\ \bar{m} \in |p^\#| \wedge p_0^\#(0) = \underline{A} \wedge p_{\bar{m}}^\#(0) = \bar{A}\} \neq \phi] \Rightarrow$
$\{p \in \Sigma :\ \bar{m} \in |p| \wedge p_0 = \underline{A} \wedge p_{\bar{m}} = \bar{A}\} \neq \phi \Rightarrow \psi(\underline{A},\bar{A}) \Rightarrow \psi(f_{\underline{A}}^\#(\langle \underline{A},\underline{T},\underline{m}\rangle), f_{\bar{A}}^\#(\langle \bar{A},\bar{T},\bar{m}\rangle))$.

□

4.3 CONSTRUCTION D'UNE METHODE DE PREUVE D'INVARIANCE A PARTIR D'UNE SEMANTIQUE OPERATIONNELLE ET D'UN PRINCIPE D'INDUCTION PAR DECOMPOSITION DE L'INVARIANT GLOBAL EN INVARIANTS LOCAUX

4.3.1 FORMALISATION DE LA CONSTRUCTION

Nous montrons comment construire formellement une méthode de preuve pour un langage de programmation étant donnés une sémantique opérationnelle, un principe d'induction, un langage d'assertions et sa sémantique.

4.3.1.1 Définition de la sémantique opérationnelle

Pour construire formellement une méthode de preuve de propriétés d'invariance pour un langage de programmation $\mathcal{P}r$ il faut commencer par en définir la sémantique opérationnelle,

$Pr \in \mathcal{P}r \longrightarrow \langle S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!]\rangle$

ce qui donne $\varepsilon[\![Pr]\!]$ et $t[\![Pr]\!]$ (cf. 2.3), ou bien

$Pr \in \mathcal{P}r \longrightarrow \langle S[\![Pr]\!], A[\![Pr]\!], t[\![Pr]\!], \varepsilon[\![Pr]\!]\rangle$

ce qui donne $\Sigma[\![Pr]\!]$ (cf. 2.4).

4.3.1.2 Définition de la propriété invariante à démontrer

Il faut ensuite définir la propriété d'invariance d'intérêt en définissant un ensemble $\mathcal{P}$ de propriétés et une application

$$\langle Pr \in \mathcal{P}r,\ P \in \mathcal{P} \rangle \longrightarrow \psi[\![Pr,P]\!] \in (S[\![Pr]\!] \times S[\![Pr]\!] \to \{tt,ff\})$$

de sorte que la spécification "Pr a la propriété P" signifie :

$$\forall p \in \Sigma[\![Pr]\!],\ i \in |p|.\ \psi[\![Pr,P]\!](p_0,p_i)$$

4.3.1.3 Choix d'un principe d'induction

Il faut ensuite choisir un principe d'induction qui, de manière générale a la forme :

$$[\exists I \in A_S[\![S]\!].\ Cv[\![S,A,\Sigma,t,\varepsilon,\psi]\!](I)]$$

En remplaçant $S, A, \Sigma, t, \varepsilon$ et ψ par leurs définitions dans ce principe d'induction, nous obtenons les conditions de vérification d'une propriété P pour un programme Pr quelconque. Il s'agit de déterminer l'application :

$$\langle Pr,P \rangle \longrightarrow \langle \tilde{A_S}[\![Pr,P]\!],\ \tilde{Cv}[\![Pr,P]\!] \rangle$$

de sorte qu'une preuve ait la forme :

$$[\exists \tilde{I} \in \tilde{A_S}[\![Pr,P]\!].\ \tilde{Cv}[\![Pr,P]\!](\tilde{I})]$$

et consiste donc à inventer un invariant $\tilde{I}$ (qui se présente généralement sous forme d'une conjonction d'invariants locaux associés à certains points du programme) puis à démontrer qu'il satisfait certaines conditions de vérification $\tilde{Cv}[\![Pr,P]\!]$.

On peut choisir $\tilde{A_S}[\![Pr,P]\!] = A_S[\![S[\![Pr]\!]]\!]$ et $\tilde{Cv}[\![Pr,P]\!] = Cv[\![S[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!], t[\![Pr]\!], \varepsilon[\![Pr]\!], \psi[\![Pr]\!]]\!]$ mais ce choix ne permet pas de faire toutes les simplifications désirables.

C'est pourquoi nous avons proposé une autre démarche (Cousot-Cousot [80a], Cousot-R [81], Cousot-Cousot [82a], Cousot-Cousot [84]) qui se justifie comme suit :

La méthode de preuve doit être correcte :

$$[\exists \tilde{I} \in \tilde{A_S}[\![Pr,P]\!].\ \tilde{Cv}[\![Pr,P]\!](\tilde{I})] \Longrightarrow [\forall p \in \Sigma[\![Pr]\!], i \in |p|.\ \psi[\![Pr,P]\!](p_0,p_i)]$$

et comme le principe d'induction choisi est correct et (sémantiquement) complet, ceci équivaut à :

$$[\exists \tilde{I} \in \tilde{A_S}[\![Pr,P]\!].\ \tilde{Cv}[\![Pr,P]\!](\tilde{I})] \Longrightarrow [\exists I \in A_S[\![S]\!].\ Cv[\![S,A,\Sigma,t,\varepsilon,\psi]\!](I)]$$

ou encore :

$$\forall \tilde{I} \in \tilde{A_S}[\![Pr,P]\!].\ \tilde{Cv}[\![Pr,P]\!](\tilde{I}) \Longrightarrow \exists I \in A_S[\![S]\!].\ Cv[\![S,A,\Sigma,t,\varepsilon,\psi]\!](I)$$

d'où nous déduisons qu'il doit exister une fonction $\gamma \in (\tilde{A_S}[\![Pr,P]\!] \to A_S[\![S[\![Pr]\!]]\!])$ telle que :

$$\tilde{Cv}[\![Pr,P]\!] \Longrightarrow Cv[\![S,A,\Sigma,t,\varepsilon,\psi]\!] \circ \gamma$$

De façon similaire, la méthode de preuve doit être (sémantiquement) complète, d'où nous déduisons qu'il doit exister une fonction $\alpha \in (A_S[\![S[\![Pr]\!]]\!] \to \tilde{A_S}[\![Pr,P]\!])$ telle que :

$$Cv[\![S,A,\Sigma,t,\varepsilon,\psi]\!] \Rightarrow \tilde{Cv}[\![Pr,P]\!] \circ \alpha$$

La fonction $\gamma[\![Pr,P]\!]$ donne la signification $\gamma[\![Pr,P]\!](\tilde{I})$ des invariants "locaux" $\tilde{I} \in \tilde{A_S}[\![Pr,P]\!]$ en terme des invariants "globaux" de $A_S[\![S[\![Pr]\!]]\!]$, tandis que la fonction $\alpha[\![Pr,P]\!]$ donne la représentation $\alpha[\![Pr,P]\!](I)$ de l'invariant global $I \in A_S[\![S[\![Pr]\!]]\!]$ par des invariants "locaux" de $\tilde{A_S}[\![Pr,P]\!]$. Ces éléments conduisent à poursuivre la construction de la méthode de preuve comme suit :

4.3.1.4 Choix d'un langage pour exprimer les invariants locaux

En pratique l'invariant utilisé pour faire la preuve n'est pas un élément de $A\delta[\![S[\![P_r]\!]]\!]$ (ce qui donnerait un invariant global pour tout le programme) mais il s'exprime généralement de manière équivalente mais plus aisée comme élément d'un certain ensemble $A\tilde{\delta}[\![P_r, P]\!]$ (que nous appelerons conventionnellement "langage"), le plus souvent sous forme d'invariants locaux associés à divers points du programme.

Exemple 4.3.1.4-1

Considérons l'exemple 4.2.1.1-1, nous avons pour le programme P_r :

```
1:
    Q := 0;
2:  while X ≥ Y do
3:      Q := Q+1;
4:      X := X-Y;
5:  od;
6:
```

$C = \{1, \dots, 6\}$ états de contrôle

$\mathcal{V} = \{X, Y, Q\}$ variables

$\mathcal{M} = (\mathcal{V} \to \mathbb{Z})$ états mémoire

$S = C \times \mathcal{M}$ états

Un invariant $I \in A\delta[\![S]\!] = (S \times S \to \{tt, ff\})$ est exprimé comme un vecteur $\tilde{I} = \langle P_1, \dots, P_6 \rangle$ où $P_i \in (\mathbb{Z}^5 \to \{tt, ff\})$, $i = 1, \dots, 6$. Par conséquent $A\tilde{\delta}[\![P_r, 4]\!] = \prod_{c \in C} (\mathbb{Z}^5 \to \{tt, ff\})$

□

4.3.1.5 Définition de la sémantique du langage exprimant les invariants locaux

La sémantique du langage $A\tilde{\delta}[\![P_r, P]\!]$ exprimant les invariants locaux se définit en terme de $A\delta[\![S[\![P_r]\!]]\!]$ au moyen de deux fonctions :

$$\alpha[\![P_r, P]\!] \in (A\delta[\![S[\![P_r]\!]]\!] \to A\tilde{\delta}[\![P_r, P]\!])$$

$$\gamma[\![P_r, P]\!] \in (A\tilde{\delta}[\![P_r, P]\!] \to A\delta[\![S[\![P_r]\!]]\!])$$

Exemple 4.3.1.5-1

La signification de $\tilde{I} = \langle P_1, \dots, P_6 \rangle$ (cf. exemple 4.2.1.1-1) est $I = \delta(\tilde{I})$ tel que $I(\underline{\delta}, \delta) = [\exists i \in C, \underline{x}, \underline{y}, \underline{q}, x, y, q \in \mathbb{Z}.\ \underline{\delta} = \langle 1, \langle \underline{x}, \underline{y}, \underline{q} \rangle\rangle \wedge \delta = \langle i, \langle x, y, q \rangle\rangle \wedge P_i(\underline{x}, \underline{y}, x, y, q)]$. Ceci formalise le fait que $P_i(\underline{x}, \underline{y}, x, y, q)$ est vrai entre les valeurs initiales $\underline{x}, \underline{y}$ et les valeurs courantes x, y, q des variables X, Y, Q du programme quand le contrôle est au point i.

Réciproquement, un invariant $I \in (S \times S \to \{tt, ff\})$ peut être représenté par $\alpha(I) = \langle P_1, \dots, P_6 \rangle$ tel que pour tout $i = 1, \dots, 6$, $P_i(\underline{x}, \underline{y}, x, y, q) = [\exists \underline{q} \in \mathbb{Z}.\ I(\langle 1, \langle \underline{x}, \underline{y}, \underline{q} \rangle\rangle, \langle i, \langle x, y, q \rangle\rangle)]$.

□

4.3.1.6 Propriétés du langage exprimant les invariants locaux et sa sémantique

4.3.1.6.1 Treillis complets des invariants locaux

Dans les principes d'induction que nous avons considérés au paragraphe 4.2, $A_S[\![S]\!]$ était de la forme $(S \to \{tt, ff\})$, $(S \times S \to \{tt, ff\})$, etc., donc toujours un treillis complet booléen $\langle A_S[\![S]\!], \Rightarrow, \vee, \wedge, tt, ff, \neg \rangle$.

Quand on choisit le langage $\tilde{A_S}[\![Pr, P]\!]$ pour exprimer les invariants locaux, il faut pouvoir exprimer qu'un invariant est plus fort ou plus faible qu'un autre de façon à pouvoir traduire dans $C\tilde{v}$ l'implication qui figure dans Cv. Il est donc nécessaire de disposer d'un ordre partiel $\tilde{\leq}$ sur $\tilde{A_S}$ correspondant à $\Rightarrow$ sur A_S. Ceci nous conduira à choisir $\langle \tilde{A_S}[\![Pr, P]\!], \tilde{\leq} \rangle$ comme un ensemble partiellement ordonné.

Il arrive souvent que $\langle \tilde{A_S}[\![Pr, P]\!], \tilde{\leq} \rangle$ soit un treillis complet. En effet la propriété que la conjonction d'invariants est invariante doit également être conservée pour $\tilde{A_S}[\![Pr, P]\!]$ car elle permet de combiner des preuves indépendantes en une seule sans aucune vérification supplémentaire. Ceci signifie que $\tilde{A_S}[\![Pr, P]\!]$ doit être muni d'une opération borne inférieure $\tilde{\bigsqcap}_{i \in \Delta} \tilde{I}_i$ de $\{\tilde{I}_i : i \in \Delta\}$ pour $\tilde{\leq}$. Enfin l'invariant tt de $A_S[\![S[\![Pr]\!]]\!]$ exprime qu'on ne sait rien sur le programme Pr. Nous pourrons toujours ajouter l'équivalent à $\tilde{A_S}[\![Pr, P]\!]$ sous la forme d'un supremum $\top$ tel que $\forall \tilde{I} \in \tilde{A_S}[\![Pr, P]\!]. \tilde{I} \tilde{\leq} \top$. Avec ces hypothèses $\langle \tilde{A_S}[\![Pr, P]\!], \tilde{\leq}, \sqcap, \top \rangle$ est un inf-demi-treillis complet avec supremum et par conséquent c'est un treillis complet $\langle \tilde{A_S}[\![Pr, P]\!], \tilde{\leq}, \sqcap, \sqcup, \top, \bot \rangle$.

Quand nous utiliserons les principes d'induction contrapositifs qui utilisent la négation, nous serons amenés à choisir $\langle \tilde{A_S}[\![Pr, P]\!], \tilde{\leq}, \sqcup, \sqcap, \top, \bot, \neg \rangle$ comme étant un treillis complet booléen.

Exemple 4.3.1.6.1 - 1

Dans l'exemple 4.3.1.4 - 1, nous avons :

$S = C \times (\mathcal{V} \to \mathbb{Z})$ où $C = \{1, \ldots, 6\}$ et $\mathcal{V} = \{X, Y, Q\}$

$A_S[\![S]\!] = (S \times S \to \{tt, ff\})$ qui est un treillis complet booléen pour $\Rightarrow, \vee, \wedge, tt, ff, \neg$.

$\tilde{A_S}[\![Pr, \psi]\!] = \prod_{c \in C} (\mathbb{Z}^{\mathcal{V}} \to \{tt, ff\})$ qui est un treillis complet booléen pour $\tilde{\leq}, \sqcup, \sqcap, \top, \bot, \neg$ définis par :

$\langle P_1, \ldots, P_6 \rangle \tilde{\leq} \langle Q_1, \ldots, Q_6 \rangle$ si et seulement si $\bigwedge_{i=1}^{6} P_i \Rightarrow Q_i$

$\bigsqcup_{i \in \Delta} \langle P_1^i, \ldots, P_6^i \rangle = \langle \bigvee_{i \in \Delta} P_1^i, \ldots, \bigvee_{i \in \Delta} P_6^i \rangle$ et de même pour $\sqcap$ et $\wedge$

$\bot = \langle ff, \ldots, ff \rangle$ et de même pour $\top$ et tt

$\neg \langle P_1, \ldots, P_6 \rangle = \langle \neg P_1, \ldots, \neg P_6 \rangle$

□

4.3.1.6.2 Correspondance entre invariants locaux et globaux

La paire (α, γ) de fonctions $\alpha \in (A_S \to \tilde{A_S})$ et $\gamma \in (\tilde{A_S} \to A_S)$ entre invariants globaux A_S et invariants locaux $\tilde{A_S}$ a souvent des propriétés intéressantes qui peuvent être exploitées pour construire la méthode de preuve. Ces propriétés sont les suivantes :

4.3.1.6.2.1 Correspondance monotone

Le fait que l'ordre $\tilde{\leq}$ sur $\tilde{A_S}$ corresponde à l'implication $\Rightarrow$ sur A_S s'exprime par la monotonie de α et de γ :

(a) $\forall I_1, I_2 \in A_S. \; [(I_1 \Rightarrow I_2) \Rightarrow (\alpha(I_1) \tilde{\leq} \alpha(I_2))]$

(b) $\forall \tilde{I}_1, \tilde{I}_2 \in \tilde{A_S}. \; [(\tilde{I}_1 \tilde{\leq} \tilde{I}_2) \Rightarrow (\gamma(\tilde{I}_1) \Rightarrow \gamma(\tilde{I}_2))]$

4.3.1.6.2.2 Demi-correspondance de Galois

Supposant α et γ monotones (et donc que $\sqsubseteq$ correspond à $\Rightarrow$), il se peut qu'on ait $\neg(I \Rightarrow \gamma(\tilde{I}))$ et $\alpha(I) \sqsubseteq \tilde{I}$. Supposons par exemple que I soit le plus fort invariant pour une sémantique $\langle S, A, \tau \rangle$ (c'est-à-dire que $I(\underline{\Delta}, \Delta) = [\exists p \in \Sigma, i \in |p| \cdot p_0 = \underline{\Delta} \wedge p_i = \Delta]$). Alors (le sens de) $\tilde{I} \in \tilde{A_S}$ est invariant si et seulement si $I \Rightarrow \gamma(\tilde{I})$. Nous voudrions pouvoir transposer ce raisonnement dans $\tilde{A_S}$ c'est-à-dire que $\alpha(I) \sqsubseteq \tilde{I}$. Ceci nous amène à supposer que α et γ sont monotones et que

$\forall I \in A_S, \tilde{I} \in \tilde{A_S} \cdot (\alpha(I) \sqsubseteq \tilde{I}) \Rightarrow (I \Rightarrow \gamma(\tilde{I}))$

ou encore, de manière équivalente

(a) $\alpha \in (A_S \to \tilde{A_S})$ est monotone

(b) $\gamma \in (\tilde{A_S} \to A_S)$ est monotone

(c) $\gamma \circ \alpha$ est extensive

4.3.1.6.2.3 Quasi-correspondance de Galois

Quand (α, γ) est une demi-correspondance de Galois, il se peut que $\gamma \circ \alpha(I) \not\Rightarrow J$ même s'il existe $\tilde{I}$ tel que $I \Rightarrow \gamma(\tilde{I}) \Rightarrow J$. Autrement dit la représentation $\alpha(I)$ de I dans $\tilde{A_S}$ donne moins d'informations que si on avait choisi $\alpha(I) = \tilde{I}$. Pour que $\alpha(I)$ soit la meilleure représentation possible de $I \in A_S$ dans $\tilde{A_S}$, il faut supposer que

$\gamma \circ \alpha(I) \Rightarrow \wedge \{\gamma(\tilde{I}) : I \Rightarrow \gamma(\tilde{I})\}$

pour éviter que $\gamma \circ \alpha(I) \not\Rightarrow J$ lorsque $I \Rightarrow \gamma(\tilde{I}) = J$. Comme $\gamma \circ \alpha$ est extensive cette condition revient à supposer que $\gamma \circ \alpha(I) = \wedge \{\gamma(\tilde{I}) : I \Rightarrow \gamma(\tilde{I})\}$, soit

(a) $\alpha \in (A_S \to \tilde{A_S})$ est monotone

(b) $\gamma \in (\tilde{A_S} \to A_S)$ est monotone

(c) $\gamma \circ \alpha$ est extensive

(d) $\forall I \in A_S \cdot [\gamma \circ \alpha(I) = \wedge \{\gamma(\tilde{I}) : I \Rightarrow \gamma(\tilde{I})\}]$

Observons que les propriétés (c) et (d) sont indépendantes et que $\gamma \circ \alpha$ est l'unique opérateur de fermeture supérieure $\varrho \in (A_S \to A_S)$ tel que $\varrho[A_S] = \gamma[\tilde{A_S}]$. Autrement dit, à la représentation près, seuls les éléments de A_S qui appartiennent à $\varrho[A_S]$ sont disponibles quand on raisonne dans $\tilde{A_S}$ au lieu de A_S. Tout invariant I devra être approché par un invariant plus faible, le meilleur étant $\varrho(I)$. En général $\varrho(I) \neq I$ et cette perte d'information peut évidemment conduire à des problèmes d'incomplétude mais ce n'est pas toujours le cas.

4.3.1.6.2.4 Correspondance de Galois

Bien que (α, γ) soit une quasi-correspondance de Galois, α peut ne pas être un morphisme complet pour les disjonctions. Comme le plus fort invariant pour les principes d'induction (positifs) s'exprime comme une disjonction (en général infinie) ceci peut avoir pour conséquence que la représentation du plus fort invariant par α dans $\tilde{A_S}$ conduise à une perte d'information qui est source d'incomplétude. La propriété que α est un morphisme complet pour la disjonction et donc de manière équivalente que la paire (α, γ) est une correspondance de Galois peut donc être utile :

(a) $\alpha \in (A_S \to \tilde{A_S})$ est monotone

(b) $\gamma \in (\tilde{A_S} \to A_S)$ est monotone

(c) $\gamma \circ \alpha$ est extensive

(e) $\alpha \circ \gamma$ est réductive

(on trouvera des définitions équivalentes en annexe II).

4.3.1.6.2.5 Correspondance de Galois surjective

Les éléments de $\tilde{A}_{As} \setminus \alpha[A_{As}]$ ne servent à représenter aucune information de A_{As} et peuvent être éliminés en choisissant $\tilde{A}_{\tilde{s}}' = \alpha[A_{As}]$. Dans ces conditions, nous avons les propriétés :

(a) $\alpha \in (A_{As} \to \tilde{A}_{\tilde{s}})$ est monotone
(b) $\gamma \in (\tilde{A}_{\tilde{s}} \to A_{As})$ est monotone
(c) $\gamma \circ \alpha$ est extensive
(e) $\alpha \circ \gamma$ est réductive
(f) α est surjective

que nous avons fréquemment utilisées (cf. par exemple à Cousot-Cousot[76], Cousot-Cousot[77a]).

4.3.1.6.2.6 Correspondance de Galois injective

Si $I_1, I_2 \in A_{As}$ sont différents et ont la même représentation $\alpha(I_1) = \alpha(I_2)$ dans $\tilde{A}_{\tilde{s}}$, il y a une perte d'information en passant de A_{As} à $\tilde{A}_{\tilde{s}}$ par α puisque des invariants différents ne peuvent plus être distingués. Ceci peut conduire à des problèmes d'incomplétude sémantique puisque la propriété $I_1 \neq I_2$ ne peut pas s'exprimer dans $\tilde{A}_{\tilde{s}}$. Ceci nous amène aux conditions suivantes :

(a) $\alpha \in (A_{As} \to \tilde{A}_{\tilde{s}})$ est monotone
(b) $\gamma \in (\tilde{A}_{\tilde{s}} \to A_{As})$ est monotone
(c) $\gamma \circ \alpha$ est extensive
(e) $\alpha \circ \gamma$ est réductive
(g) α est injective

4.3.1.6.2.7 Isomorphisme complet

Enfin, dans le cas où les raisonnements dans A_{As} ou $\tilde{A}_{\tilde{s}}$ sont équivalents, α est un isomorphisme complet et γ est son inverse de sorte que les hypothèses suivantes sont satisfaites :

(h) $\alpha(\bigvee_{i \in \Delta} I_i) = \bigcup_{i \in \Delta} \alpha(I_i)$
(i) $\alpha(\bigwedge_{i \in \Delta} I_i) = \bigcap_{i \in \Delta} \alpha(I_i)$
(j) α est bijective
(k) $\gamma = \alpha^{-1}$

Exemple 4.3.1.6.2.7-1

Dans l'exemple 4.3.1.4-1 poursuivi en 4.3.1.5-1 et 4.3.1.6.1-1, nous avons :

$\alpha_1(I)_{\tilde{i}}(\underline{x},\underline{y},x,y,q) = [\exists \underline{q} \in \mathbb{Z}.\ I(\langle 1, \langle \underline{x},\underline{y},\underline{q} \rangle\rangle, \langle i, \langle x,y,q \rangle\rangle)]$

$\gamma_1(\tilde{I})(\underline{A},A) = [\exists i \in C, \underline{x},\underline{y},\underline{q}, x,y,q \in \mathbb{Z}.\ \underline{A} = \langle 1, \langle \underline{x},\underline{y},\underline{q} \rangle\rangle \wedge A = \langle i, \langle x,y,q \rangle\rangle \wedge \tilde{I}_{\tilde{i}}(\underline{x},\underline{y},x,y,q)]$

de sorte que les conditions (a) et (b) sont satisfaites mais pas la condition (c). Nous pouvons également choisir :

$\alpha_2(I)_{\tilde{i}}(\underline{x},\underline{y},x,y,q) = [\exists j \in C, \underline{q} \in \mathbb{Z}.\ I(\langle j, \langle x,y,q \rangle\rangle, \langle i, \langle x,y,q \rangle\rangle)]$

$\gamma_2(\tilde{I})(\underline{A},A) = [\exists j, i \in C, \underline{x},\underline{y},\underline{q}, x,y,q \in \mathbb{Z}.\ \underline{A} = \langle j, \langle \underline{x},\underline{y},\underline{q} \rangle\rangle \wedge A = \langle i, \langle x,y,q \rangle\rangle \wedge \tilde{I}_{\tilde{i}}(\underline{x},\underline{y},x,y,q)]$

qui satisfont les conditions (a), (b), (c), (d), (e), (f), (h) mais pas (g), (i), (j) et (k). (Avec ce deuxième choix et contrairement au premier, il n'est pas supposé que $I(\underline{A},A) \Rightarrow \varepsilon(\underline{A})$).

□

4.3.1.7 Dérivation de conditions de vérificationn correctes

Etant donnés un principe d'induction correct et sémantiquement complet

$[\exists I \in A\tilde{s}.\ Cv(I)]$

et une correspondance $\gamma \in (A\tilde{s} \to As)$ entre $A\tilde{s}$ et As, toute preuve

$[\exists \tilde{I} \in A\tilde{s}.\ C\tilde{v}(\tilde{I})]$

telle que

$C\tilde{v} \Rightarrow Cv \circ \gamma$

est correcte. On peut donc choisir $C\tilde{v}[\![Pr,P]\!]$ comme étant $Cv[\![s[\![Pr]\!], A[\![Pr]\!], \Sigma[\![Pr]\!], t[\![Pr]\!], \in[\![Pr]\!], \psi[\![Pr,P]\!]]\!] \circ \gamma[\![Pr,P]\!]$. Par diverses manipulations algébriques on cherchera à exprimer cette condition sous forme d'une conjonction de conditions plus simples correspondant chacune à une commande élémentaire du programme Pr. Des simplifications sont possibles puisque c'est une implication et non pas une égalité qui est requise. La méthode de preuve obtenue de cette manière est correcte par construction. Pourque le résultat soit valable non pas pour un programme Pr particulier mais pour le langage $\mathcal{P}r$ considéré il faut procéder par induction sur la syntaxe du langage.

Exemple 4.3.1.7-1

Dans le cas des principes d'induction pour l'invariance, la condition de vérification

$[\exists I \in As.\ Cv(I)]$

peut s'écrire sous forme d'une conjonction :

$[\exists I \in As.\ Cv_\varepsilon(I) \wedge Cv_i(I) \wedge Cv_\delta(I)]$

Par exemple pour le principe d'induction (-Y), nous avons :

$Cv_\varepsilon(I) = [\forall \underline{a} \in S.\ \varepsilon(\underline{a}) \Rightarrow I(\underline{a},\underline{a})]$

$Cv_i(I) = [\forall \underline{a},a \in S.\ [\exists a' \in S, a \in A.\ \varepsilon(\underline{a}) \wedge I(\underline{a},a') \wedge t_a(a',a)] \Rightarrow I(\underline{a},a)]$

$Cv_\delta(I) = [\forall \underline{a},\tilde{a} \in S.\ [\varepsilon(\underline{a}) \wedge I(\underline{a},\tilde{a})] \Rightarrow \psi(\underline{a},\tilde{a})]$

Dans ce cas, on choisira le plus souvent

$C\tilde{v}(\tilde{I}) = C\tilde{v}_\varepsilon(\tilde{I}) \wedge C\tilde{v}_i(\tilde{I}) \wedge C\tilde{v}_\delta(\tilde{I})$

satisfaisant :

$C\tilde{v}_\varepsilon(\tilde{I}) \Rightarrow Cv_\varepsilon(\gamma(\tilde{I}))$

$C\tilde{v}_i(\tilde{I}) \Rightarrow Cv_i(\gamma(\tilde{I}))$

$C\tilde{v}_\delta(\tilde{I}) \Rightarrow Cv_\delta(\gamma(\tilde{I}))$

De plus dans le cas des principes d'induction pour l'invariance le terme $Cv_i(I)$ est de la forme $F(I) \Rightarrow I$ (ou dualement $I \Rightarrow F(I)$). Par exemple, dans le cas du principe d'induction (-Y), nous avons

$Cv_i(I) = [F(I) \Rightarrow I]$

où

$F(I)(\underline{a},a) = [\exists a' \in S, a \in A.\ \varepsilon(\underline{a}) \wedge I(\underline{a},a') \wedge t_a(a',a)]$

ou bien, nous pouvons intégrer le terme $Cv_\varepsilon(I)$ dans $Cv_i(I)$ ce qui donne

$F(I)(\underline{a},a) = [\exists a' \in S, a \in A.\ \varepsilon(\underline{a}) \wedge ([a = \underline{a}] \vee [I(\underline{a},a') \wedge t_a(a',a)])]$

Dans les deux cas la condition $C\tilde{v}_i(\tilde{I}) \Rightarrow Cv_i(\gamma(\tilde{I}))$ s'écrit :

$C\tilde{v}_i(\tilde{I}) \Rightarrow [F(\gamma(\tilde{I})) \Rightarrow \gamma(\tilde{I})]$

Quand (α,γ) est une demi-correspondance de Galois (cf. 4.3.1.6.2.2), on est amené à choisir

$C\tilde{v}_i(\tilde{I}) \Rightarrow [\alpha \circ F \circ \gamma(\tilde{I}) \sqsubseteq \tilde{I}]$

car $\alpha \circ F \circ \gamma(\tilde{I}) \sqsubseteq \tilde{I}$ implique que $F \circ \gamma(\tilde{I}) \Rightarrow \gamma(\tilde{I})$. Posons $\underline{F} = \alpha \circ F \circ \gamma$. A la suite de Cousot-Cousot [80c], observons que si (α,γ) est une correspondance de Galois injective (cf. 4.3.1.6.2.1) alors $\alpha \circ F = \underline{F} \circ \alpha$, (en effet, γ étant injective nous avons $\gamma \circ \alpha = 1$ et donc $\alpha \circ F = \alpha \circ F \circ \gamma \circ \alpha = \underline{F} \circ \alpha$).

Supposons maintenant que (α n'étant pas injective), il existe un opérateur $\tilde{F}$ sur $\tilde{As}$ tel que $\alpha \circ F = \tilde{F} \circ \alpha$. Nous avons alors $[\underset{\sim}{F} \sqsubseteq \tilde{F}$, l'inégalité peut être stricte et il y a égalité si (mais pas seulement si) α est surjective$]$

(En effet $\underset{\sim}{F} = \alpha \circ F \circ \gamma = \tilde{F} \circ \alpha \circ \gamma \sqsubseteq \tilde{F}$ car $\alpha \circ \gamma$ est réductive. L'inégalité peut être stricte comme le montre l'exemple suivant : $As = \tilde{As} = \{a,b\}$, $a \sqsubset b$, $\alpha(a) = \alpha(b) = a$, $\gamma(a) = \gamma(b) = b$, $F(x) = x$, $\tilde{F}(\tilde{x}) = \tilde{x}$, $\underset{\sim}{F}(a) = \underset{\sim}{F}(b) = a$. Si α est surjective alors $\alpha \circ \gamma$ est l'identité et donc $\underset{\sim}{F} = \tilde{F}$ mais la réciproque n'est pas vraie comme le montre l'exemple suivant : $As = \tilde{As} = \{a,b\}$, $a \sqsubset b$, $F(x) = \underset{\sim}{F}(x) = \tilde{F}(x) = \alpha(x) = a$, $\gamma(x) = b$.)

Dans ces conditions, on peut choisir

$\tilde{Cv}_{\underset{\sim}{F}}(\tilde{I}) = [\bar{F}(\tilde{I}) \sqsubseteq \tilde{I}]$

où $\bar{F}$ est un opérateur sur $\tilde{As}$ tel que $\underset{\sim}{F} \sqsubseteq \bar{F} \sqsubseteq \tilde{F}$ (puisque $\bar{F}(\tilde{I}) \sqsubseteq \tilde{I}$ implique $\underset{\sim}{F}(\tilde{I}) \sqsubseteq \tilde{I}$ donc $\alpha \circ F \circ \gamma(\tilde{I}) \sqsubseteq \tilde{I}$ soit $F \circ \gamma(\tilde{I}) \Rightarrow \gamma(\tilde{I})$ et donc $Cv_{\underset{\sim}{F}}(\gamma(\tilde{I}))$). Si α n'est pas surjective le treillis complet des $\bar{F} \in (\tilde{As} \to \tilde{As})$ tels que $\underset{\sim}{F} \sqsubseteq \bar{F} \sqsubseteq \tilde{F}$ n'est pas réduit à un seul élément de sorte qu'il est possible d'envisager diverses conditions de vérifications.

Parmi ces conditions de vérification, $\tilde{F}$ peut nécessiter de trouver un invariant $\tilde{I}$ plus fort que pour $\underset{\sim}{F}$ puisque $\tilde{F}(\tilde{I}) \sqsubseteq \tilde{I}$ implique $\underset{\sim}{F}(\tilde{I}) \sqsubseteq \tilde{I}$ mais la réciproque n'est pas vraie. Cet argument en faveur du choix de $\underset{\sim}{F}$ doit être modulé par le fait qu'en pratique $\tilde{F}$ conduit à des conditions de vérification plus simples. En pratique, un équilibre doit être trouvé entre des invariants plus faibles et des conditions de vérification plus compliquées ou des invariants plus forts et des conditions de vérification plus simples.

□

4.3.1.8 Vérification de la complétude sémantique

La vérification de la complétude sémantique consiste à montrer que :

$[\exists I \in As.\ Cv(I)] \Rightarrow [\exists \tilde{I} \in \tilde{As}.\ \tilde{Cv}(\tilde{I})]$

Si nous choisissons $\tilde{I}$ comme étant la représentation de I par $\alpha \in (As \to \tilde{As})$, nous obtenons la condition suffisante de complétude :

$Cv \Rightarrow \tilde{Cv} \circ \alpha$

(celle-ci étant d'ailleurs nécessaire pour un α convenablement choisi).

Exemple 4.3.1.8-1

Dans le cas particulier que nous rencontrerons assez souvent où $\gamma \circ \alpha = \underline{1}$ (cf. 4.3.1.C.2.C, 4.3.1.6.2.7) et $\tilde{Cv} = Cv \circ \gamma$ auquel cas la condition suffisante de correction $\tilde{Cv} \Rightarrow Cv \circ \gamma$ est trivialement satisfaite, nous avons $\tilde{Cv} \circ \alpha = Cv \circ \gamma \circ \alpha = Cv$ et la condition suffisante de complétude $Cv \Rightarrow \tilde{Cv} \circ \alpha$ est également trivialement satisfaite.

□

Exemple 4.3.1.8-2

Pour poursuivre l'exemple 4.3.1.7-1, dans le cas où

$Cv(I) = [F(I) \Rightarrow I \ \wedge\ I \Rightarrow \psi]$

$\tilde{Cv}(\tilde{I}) = [\bar{F}(\tilde{I}) \sqsubseteq \tilde{I} \ \wedge\ \tilde{I} \sqsubseteq \alpha(\psi)]$

si α est monotone, $\bar{F} \sqsubseteq \tilde{F}$ et $\alpha \circ F = \tilde{F} \circ \alpha$, il faut vérifier que :

$[F(I) \Rightarrow I \ \wedge\ I \Rightarrow \psi] \Rightarrow [\bar{F}(\alpha(I)) \sqsubseteq \alpha(I) \ \wedge\ \alpha(I) \sqsubseteq \alpha(\psi)]$

nous avons bien $(I \Rightarrow \psi)$ qui implique $\alpha(I) \sqsubseteq \alpha(\psi)$ par monotonie de sorte qu'il suffit de vérifier que

$[F(I) \Rightarrow I] \Rightarrow [\bar{F}(\alpha(I)) \sqsubseteq \alpha(I)]$

ce qui est évident car $[F(I) \Rightarrow I] \Rightarrow [\alpha \circ F(I) \sqsubseteq \alpha(I)]$ (par monotonie) $\Rightarrow [\tilde{F}(\alpha(I)) \sqsubseteq \alpha(I)]$ (car $\alpha \circ F = \tilde{F} \circ \alpha$) $\Rightarrow [\bar{F}(\alpha(I)) \sqsubseteq \alpha(I)]$ (car $\bar{F} \sqsubseteq \tilde{F}$).

□

4.3.2 EXEMPLES DE CONSTRUCTIONS

4.3.2.1 Construction d'une méthode de preuve de non-terminaison, d'absence d'erreurs à l'exécution et d'invariance globale de programmes séquentiels par l'absurde

4.3.2.1.1 La non-terminaison est une propriété d'invariance

Soit $Ps \in \mathcal{P}s$ un programme séquentiel et $\phi \in ((\mathcal{V} \to \mathcal{D}) \to \{tt, ff\})$ une caractérisation des valeurs possibles des variables à l'entrée du programme. Le programme Ps ne se termine pas normalement si aucun état de sortie ne peut être atteint durant l'exécution :

$$\forall p \in \Sigma^{<}\langle S[\![Ps]\!], A[\![Ps]\!], t[\![Ps]\!], \varepsilon\rangle, i \in |p|.\ [\sigma(p_i) \Rightarrow \psi(p_i)]$$

où

$$\varepsilon(\langle L, M\rangle) = [(Ps \equiv L:p) \wedge \phi(M)]$$
$$\sigma(\langle L, M\rangle) = tt$$
$$\psi(\langle L, M\rangle) = [\neg(Ps \equiv \beta L:)]$$

4.3.2.1.2 Choix d'un principe d'induction

Puisque ψ est une assertion sur les états finaux, d'après le paragraphe 4.2.1.3, nous pourrons utiliser les principes d'induction $(\text{-}i\text{-})$, $(\text{-}\tilde{i}\text{-})$, $(\text{-}\bar{i}\text{-})$, $(\text{-}\bar{\tilde{i}}\text{-})$. Nous choisirons $(\text{-}\bar{\tilde{i}}\text{-})$ comme exemple car ce principe d'induction n'est pas du tout conventionnel. $(\text{-}\bar{\tilde{i}}\text{-})$ est de la forme :

$$[\exists I \in As[\![Ps]\!].\ Cv[\![Ps]\!](\varepsilon,\sigma)(\psi)(I)]$$

où

$$As[\![Ps]\!] = (S[\![Ps]\!] \to \{tt, ff\})$$

$$Cv[\![Ps]\!](\varepsilon,\sigma)(\psi)(I) = Cv_\sigma[\![Ps]\!](\psi,\sigma)(I) \wedge Cv_\iota[\![Ps]\!](I) \wedge Cv_\varepsilon[\![Ps]\!](\varepsilon)(I)$$

$$Cv_\sigma[\![Ps]\!](\psi,\sigma)(I) = [\forall \bar{s} \in S[\![Ps]\!].\ [\neg\psi(\bar{s}) \wedge \sigma(\bar{s})] \Rightarrow I(\bar{s})]$$

$$Cv_\iota[\![Ps]\!](I) = [\forall s \in S[\![Ps]\!], a \in A[\![Ps]\!].\ I(s) \Rightarrow \neg[\exists s' \in S[\![Ps]\!].\ \neg I(s') \wedge t[\![Ps]\!]_a(s', s)]]$$

$$Cv_\varepsilon[\![Ps]\!](\varepsilon)(I) = [\forall \underline{s} \in S[\![Ps]\!].\ \varepsilon(\underline{s}) \Rightarrow \neg I(\underline{s})]$$

4.3.2.1.3 Choix d'un langage pour exprimer les invariants locaux

Nous décidons de représenter un invariant global $I \in As[\![Ps]\!]$ par un vecteur $\tilde{I}$ d'invariants locaux sur les états mémoires et associés à chaque point du programme. Alors

$$\tilde{As}[\![Ps]\!] = \prod_{L \in C[\![Ps]\!]} (M[\![Ps]\!] \to \{tt, ff\})$$

où

$$C[\![Ps]\!] = \{L \in \mathcal{L}.\ Ps \equiv \beta L:\bar{s}\}$$
$$M[\![Ps]\!] = (\{x \in \mathcal{V}:\ Ps \equiv \langle X p\} \to \mathcal{D})$$

La signification de ce vecteur d'invariants locaux est que l'invariant $\tilde{I}(L)(M)$ associé au point L est vrai pour l'état mémoire M de tout état $\langle L, M\rangle$ du programme ayant L comme état de contrôle. Plus formellement

$$\gamma[\![Ps]\!] \in (\tilde{As}[\![Ps]\!] \to As[\![Ps]\!])$$
$$\gamma[\![Ps]\!](\tilde{I})(\langle L, M\rangle) = \tilde{I}(L)(M)$$

Réciproquement, un invariant global $I \in As[\![Ps]\!]$ est représenté par $\chi[\![Ps]\!](I) \in \tilde{As}[\![Ps]\!]$, c'est à dire un vecteur $\tilde{I}$ d'invariants locaux sur les états mémoire M tels que :

$$\forall L \in C[\![Ps]\!].\ \tilde{I}(L)(M) = I(\langle L, M\rangle)$$

4.3.2.1.4 Dérivation de conditions de vérification correctes

Nous avons à construire $\tilde{Cv}[\![Ps]\!]$ tel que :

$$\tilde{Cv}[\![Ps]\!](\varepsilon,\sigma)(\psi)(\tilde{I}) \Rightarrow Cv[\![Ps]\!](\varepsilon,\sigma)(\gamma[\![Ps]\!](\tilde{I}))$$

Puisque $Cv[\![Ps]\!]$ est une conjonction de trois conditions de vérification, nous choisirons $\tilde{Cv}[\![Ps]\!]$ de la même forme, c'est-à-dire :

$$\tilde{Cv}[\![Ps]\!](\varepsilon,\sigma)(\psi)(\tilde{I}) = \tilde{Cv}_\sigma[\![Ps]\!](\psi,\sigma)(\tilde{I}) \wedge \tilde{Cv}_i[\![Ps]\!](\tilde{I}) \wedge \tilde{Cv}_\varepsilon[\![Ps]\!](\varepsilon)(\tilde{I})$$

Pour garantir la condition de correction ci-dessus, nous choisirons simplement :

$$\tilde{Cv}_\sigma[\![Ps]\!](\psi,\sigma)(\tilde{I}) = Cv_\sigma[\![Ps]\!](\psi,\sigma)(\gamma[\![Ps]\!](\tilde{I}))$$

$$\tilde{Cv}_i[\![Ps]\!](\tilde{I}) = Cv_i[\![Ps]\!](\gamma[\![Ps]\!](\tilde{I}))$$

$$\tilde{Cv}_\varepsilon[\![Ps]\!](\varepsilon)(\tilde{I}) = Cv_\varepsilon[\![Ps]\!](\varepsilon)(\gamma[\![Ps]\!](\tilde{I}))$$

4.3.2.1.4.1 Base

$$\tilde{Cv}_\sigma[\![Ps]\!](\psi,\sigma)(\tilde{I})$$

$$= Cv_\sigma[\![Ps]\!](\psi,\sigma)(\gamma[\![Ps]\!](\tilde{I}))$$

$$= [\forall \tilde{\sigma} \in S[\![Ps]\!] . [\neg\psi(\tilde{\sigma}) \wedge \sigma(\tilde{\sigma})] \Rightarrow \gamma[\![Ps]\!](\tilde{I})(\tilde{\sigma})]$$

Puisque $S[\![Ps]\!] = C[\![Ps]\!] \times M[\![Ps]\!]$, nous avons :

$$= [\forall L \in C[\![Ps]\!], M \in M[\![Ps]\!] . [\neg\psi(\langle L,M\rangle) \wedge \sigma(\langle L,M\rangle)] \Rightarrow \gamma[\![Ps]\!](\tilde{I})(\langle L,M\rangle)]]$$

Remplaçons σ, ψ et γ par leurs définitions :

$$= [\forall L \in C[\![Ps]\!], M \in M[\![Ps]\!] . (Ps \equiv \beta L:) \Rightarrow \tilde{I}(L)(M)]$$

Ps a un et un seul point de sortie d'où :

$$= [(Ps \equiv \beta L:) \wedge \forall M \in M[\![Ps]\!] . \tilde{I}(L)(M)]$$

4.3.2.1.4.2 Induction

$$\tilde{Cv}_i[\![Ps]\!](\tilde{I})$$

$$= Cv_i[\![Ps]\!](\gamma[\![Ps]\!](\tilde{I}))$$

$$= [\forall \Delta \in S[\![Ps]\!], a \in A[\![Ps]\!] . \gamma[\![Ps]\!](\tilde{I})(\Delta) \Rightarrow \neg[\exists \Delta' \in S[\![Ps]\!] . \neg\gamma[\![Ps]\!](\tilde{I})(\Delta') \wedge t[\![Ps]\!]_a(\Delta',\Delta)]]$$

$$= [\forall \Delta \in S[\![Ps]\!], a \in A[\![Ps]\!] . \gamma[\![Ps]\!](\tilde{I})(\Delta) \Rightarrow [\forall \Delta' \in S[\![Ps]\!] . t[\![Ps]\!]_a(\Delta',\Delta) \Rightarrow \gamma[\![Ps]\!](\tilde{I})(\Delta')]]$$

$$= [\forall L \in C[\![Ps]\!], M \in M[\![Ps]\!], a \in A[\![Ps]\!] . \tilde{I}(L)(M) \Rightarrow [\forall L' \in C[\![Ps]\!], M' \in M[\![Ps]\!] .$$

$$t[\![Ps]\!]_a(\langle L',M'\rangle,\langle L,M\rangle) \Rightarrow \tilde{I}(L')(M')]]$$

$$= \bigwedge_{L \in C[\![Ps]\!]} [\forall M \in M[\![Ps]\!] . \tilde{I}(L)(M) \Rightarrow [\forall L' \in C[\![Ps]\!], M' \in M[\![Ps]\!] .$$

$$(\underline{cond}[\![Ps]\!](L',L)(M') \wedge \underline{succ}[\![Ps]\!](L')(M',M)) \Rightarrow \tilde{I}(L')(M')]]$$

Nous avons une conjonction de conditions de vérification, une pour chaque point du programme, et de la forme :

$$\tilde{I}(L)(M) \Rightarrow [\forall L' \in C[\![Ps]\!], M' \in M[\![Ps]\!] . [\underline{cond}[\![Ps]\!](L',L)(M') \wedge \underline{succ}[\![Ps]\!](L')(M',M)] \Rightarrow \tilde{I}(L')(M')]$$

cette condition peut se décomposer aux différents cas correspondant à la définition de $\underline{cond}[\![Ps]\!]$:

(a) Si $(Ps \equiv L:p)$ alors L désigne le point d'entrée du programme, alors $\forall L' \in C[\![Ps]\!], M' \in M[\![Ps]\!] . \neg \underline{cond}[\![Ps]\!](L',L)(M')$ et la condition de vérification est donc vraie trivialement.

(b) Si $(Ps \equiv \beta L': \underline{skip}; L:\delta)$ ou $(Ps \equiv \beta L': \underline{else}\ Ps'\ \underline{fi}; L:\delta)$ ou $(Ps \equiv \beta L': \underline{fi}; L:\delta)$ alors $\underline{cond}[\![Ps]\!](L',L)(M)$ est vrai et $\underline{succ}[\![Ps]\!](L')(M',M)$ implique $M=M'$ de sorte que la condition de vérification peut être simplifiée en :

$$\tilde{I}(L)(M) \Rightarrow \tilde{I}(L')(M)$$

(c) Si $(Ps \equiv \beta L': V:=E; L:\delta)$ alors nous devons vérifier que

$$\tilde{I}(L)(M) \Rightarrow [\forall M' \in M[\![Ps]\!] . [M' \in \underline{dom}(\mathbb{E}[\![E]\!]) \wedge M = M'[V \leftarrow \mathbb{E}[\![E]\!](M')] \Rightarrow \tilde{I}(L')(M')]$$

Il faut que M' doit être de la forme $M[V \leftarrow m]$ où $m \in \mathcal{D}$ est la valeur de V avant l'affectation. Alors cette condition peut se simplifier en :

$$\tilde{I}(L)(M) \Rightarrow [\forall m \in \mathcal{D} . [M[V \leftarrow m] \in \underline{dom}(\mathbb{E}[\![E]\!]) \wedge M(V) = \mathbb{E}[\![E]\!](M[V \leftarrow m])] \Rightarrow \tilde{I}(L')(M[V \leftarrow m])]$$

Si $(Ps \equiv \beta L': V:=?; L:\delta)$, nous obtenons de même

$$\tilde{I}(L)(M) \Rightarrow [\forall m \in \mathcal{D} . \tilde{I}(L')(M[V \leftarrow m])]$$

(d) Si $(Ps \equiv \beta L' : \underline{if}\ B\ \underline{then}\ L:\delta)$ ou $(Ps \equiv \beta L' : \underline{while}\ B\ \underline{do}\ L:\delta)$ ou $(Ps \equiv \beta\ \underline{while}\ B\ \underline{do}\ L: C_0; \ldots; L_m : C_m ; L' : \underline{od} ; \delta)$ nous obtenons

$$\tilde{I}(L)(M) \Rightarrow [(M \in \underline{dom}(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = tt) \Rightarrow \tilde{I}(L')(M)]$$

(e) Si $(Ps \equiv \beta L' : \underline{if}\ B\ \underline{then}\ Ps'\ \underline{else}\ L:\delta)$ ou $(Ps \equiv \beta L' : \underline{while}\ B\ \underline{do}\ Ps'\ \underline{od} ; L:\delta)$ ou $(Ps \equiv \alpha\ \underline{while}\ B\ \underline{do}\ L_0 : C_0 ; \ldots ; L_m : C_m ; L' : \underline{od} ; L:\delta)$

$$\tilde{I}(L)(M) \Rightarrow [(M \in \underline{dom}(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = ff) \Rightarrow \tilde{I}(L')(M)]$$

4.3.2.1.4.3 Contradiction

$$\begin{aligned}
&Cv^{\sim}_{\varepsilon}[\![Ps]\!](\varepsilon)(\tilde{I}) \\
&\quad = Cv_{\varepsilon}[\![Ps]\!](\varepsilon)(\gamma[\![Ps]\!](\tilde{I})) \\
&\quad = [\forall \underline{s} \in S[\![Ps]\!].\ \varepsilon(\underline{s}) \Rightarrow \neg\gamma[\![Ps]\!](\tilde{I})(\underline{s})] \\
&\quad = [\forall L \in C[\![Ps]\!], M \in M[\![Ps]\!].\ [(Ps \equiv L:\beta) \wedge \phi(M)] \Rightarrow \neg\tilde{I}(L)(M)] \\
&\quad = [(Ps \equiv L:\beta) \wedge (\forall M \in M[\![Ps]\!].\ \phi(M) \Rightarrow \neg\tilde{I}(L)(M))]
\end{aligned}$$

4.3.2.1.5 Résumé informel des conditions de vérification

En utilisant des notations mnémoniques, nous pouvons récapituler les conditions de vérification ci-dessus comme suit (P_i est l'invariant sur les variables du programme associé au point L_i) :

- Base

βL_1: P_1

- Induction

. Commande nulle

βL_1: skip; L_2: δ — $P_2 \Rightarrow P_1$

. Commande d'affectation

βL_1: $V := E$; L_2: δ — $P_2 \Rightarrow [\forall m \in \mathfrak{D}.\ V = E[V \leftarrow m] \Rightarrow P_1[V \leftarrow m]]$

βL_1: $V := ?$; L_2: δ — $P_2 \Rightarrow [\forall m \in \mathfrak{D}.\ P_1[V \leftarrow m]]$

. Commande conditionnelle

βL_1: if B then

L_2: δ — $P_2 \Rightarrow [B \Rightarrow P_1]$

L_3:

else

L_4: δ' — $P_4 \Rightarrow [\neg B \Rightarrow P_1]$

L_5:

fi;

L_6: β' — $P_6 \Rightarrow [P_3 \wedge P_5]$

. Commande itérative

βL_1: while B do

L_2: δ — $P_2 \Rightarrow [B \Rightarrow (P_1 \wedge P_3)]$

L_3:

od;

L_4: β' — $P_4 \Rightarrow [\neg B \Rightarrow (P_1 \wedge P_3)]$

- Contradiction

L_1: β — $\phi \Rightarrow \neg P_1$

4.3.2.1.6 Exemple de preuve avec cette méthode

Soit à prouver que le programme suivant

```
1:
      q:=0;
2:
      while x≥y do
3:
          q:=q+1;
4:
          x:=x-y;
5:
      od;
6:
```

ne se termine pas normalement quand les valeurs initiales x,y des variables x,y sont telle que $x \geq 0$ et $y=0$. Nous supposons que le domaine $\mathcal{D}$ de valeurs de chaque variable est l'ensemble des entiers compris entre deux bornes min et max. Ces bornes sont supposées telles que $min<0<max$.

Les conditions de vérification (après des simplifications triviales) sont les suivantes (où $x,y,q \in [min, max]$) :

$P_6(x,y,q)$

$P_6(x,y,q) \Rightarrow [(x<y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$

$P_5(x,y,q) \Rightarrow P_4(x+y, y, q)$

$P_4(x,y,q) \Rightarrow P_3(x,y,q-1)$

$P_3(x,y,q) \Rightarrow [(x \geq y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$

$P_2(x,y,q) \Rightarrow [\forall q' \in [min, max].\ (q=0) \Rightarrow P_1(x,y,q')]$

$[(0 \leq x \leq max) \wedge (y=0)] \Rightarrow \neg P_1(x,y,q)$

Intuitivement, par l'absurde, si $y=0$ le programme ne peut se terminer que si $x<0$, en contradiction avec l'hypothèse sur la valeur initiale de x. Ceci peut se démontrer formellement en utilisant les invariants suivants :

$P_i(x,y,q) = [(y=0) \Rightarrow (x<0)]$ pour $i=1,\ldots,5$

$P_6(x,y,q) = tt$

4.3.2.1.7 Vérification de la complétude sémantique

Conformément au paragraphe 7.5, nous devons vérifier que

$\forall I \in As[\![Ps]\!].\ Cv[\![Ps]\!](\epsilon,\sigma)(\psi)(I) \Rightarrow C\check{v}[\![Ps]\!](\epsilon,\sigma)(\psi)(\alpha[\![Ps]\!](I))$

En utilisant le fait que $c\check{v}[\![Ps]\!](\epsilon,\sigma)(\psi)(\check{I}) = Cv[\![Ps]\!](\epsilon,\sigma)(\psi)(\delta[\![Ps]\!](\check{I}))$, il est suffisant de vérifier que $\delta[\![Ps]\!](\alpha[\![Ps]\!](I)) = I$, qui est trivial car $\delta[\![Ps]\!]$ est une bijection entre $A\check{s}[\![Ps]\!]$ et $As[\![Ps]\!]$ (et $\alpha[\![Ps]\!]$ est son inverse).

4.3.2.1.8 Preuve d'absence d'erreurs à l'exécution, par l'absurde pour des programmes séquentiels

Soit $Ps \in Ps$ un programme séquentiel et $\phi \in ((\mathcal{V} \to \mathcal{D}) \to \{tt, ff\})$ une caractérisation des valeurs initiales possibles des variables du programme. L'exécution du programme Ps ne conduit pas à une erreur d'exécution si et seulement si tout état atteint durant l'exécution et qui n'est pas un état de sortie a un état successeur, c'est-à-dire

$\forall p \in \mathbb{Z}<s[\![Ps]\!], A[\![Ps]\!], t[\![Ps]\!], \varepsilon>,\ i \in |p|.\ [\sigma(p_i) \Rightarrow \psi(p_i)]$

où

$\varepsilon(<L,M>) = [(Ps \equiv L:\beta) \wedge \phi(M)]$

$\sigma(<L,M>) = [\neg(Ps \equiv \alpha L:)]$

$\psi(<L,M>) = [\exists L' \in C[\![Ps]\!], M' \in M[\![Ps]\!], a \in A[\![Ps]\!].\ t[\![Ps]\!]_a(<L,M>,<L',M'>)]$

avec $M[\![Ps]\!] = (\{x \in \mathcal{V}: Ps \equiv \ldots x \beta\} \to \mathcal{D})$, $C[\![Ps]\!] = \{L \in \mathcal{L}: Ps \equiv \beta L:\delta\}$

Notez que la seule différence avec le paragraphe 4.3.2.1.1 est ε et ψ. En effet, en choisissant le principe d'induction et le langage pour exprimer les invariants locaux, considérés aux paragraphes 4.3.2.1.2 et 4.3.2.1.3, nous utilisons les mêmes conditions de vérification qu'en 4.3.2.1.4 excepté pour la base :

$$\tilde{Co}_\delta [\![Ps]\!](\psi,\sigma)(\tilde{I})$$
$$= Co_\delta [\![Ps]\!](\psi,\sigma)(\gamma [\![Ps]\!](\tilde{I}))$$
$$= [\forall \bar{\sigma} \in S[\![Ps]\!] . [\neg\psi(\bar{\sigma}) \wedge \sigma(\bar{\sigma})] \Rightarrow \gamma [\![Ps]\!](\tilde{I})(\bar{\sigma})]$$
$$= [\forall L \in C[\![Ps]\!],\ M \in M[\![Ps]\!] . [\neg(Ps \equiv \beta L:) \wedge (\forall L' \in C[\![Ps]\!], M' \in M[\![Ps]\!], a \in A[\![Ps]\!] .$$
$$\neg t[\![Ps]\!]_a(\langle L,M\rangle,\langle L',M'\rangle)] \Rightarrow \tilde{I}(L)(M)]$$

Alors pour tous les points L du programme Ps excepté l'état de sortie, nous devons prouver :

$$[\forall L' \in C[\![Ps]\!], M' \in M[\![Ps]\!], a \in A[\![Ps]\!] . \neg t[\![Ps]\!]_a(\langle L,M\rangle,\langle L',M'\rangle)] \Rightarrow \tilde{I}(L)(M)$$

Cette condition se décompose aux différents cas correspondant à $t[\![Ps]\!]_a$:

(a) Si $(Ps \equiv \beta L: \underline{skip}\ ; L':\delta)$ ou $(P \equiv \beta L: \underline{else}\ Ps'\ \underline{fi}\ ; L':\delta)$ ou $(Ps \equiv \beta L: \underline{fi}\ ; L':\delta)$ ou $(Ps \equiv \beta L: V:=?\ ; L':\delta)$ alors le membre de gauche est faux et la condition de vérification est identiquement vraie.

(b) Si $(Ps \equiv \beta L: V:=E\ ; L':\delta)$ alors nous obtenons :
$$[M \notin \underline{dom}(\mathbb{E}[\![E]\!])] \Rightarrow \tilde{I}(L)(M)$$

(c) Si $(Ps \equiv \beta L: \underline{if}\ B\ \underline{then}\ \delta)$ ou $(Ps \equiv \beta L: \underline{while}\ B\ \underline{do}\ \delta)$ ou $(Ps \equiv \beta\ \underline{while}\ B\ \underline{do}\ L_0:C_0\ ;\ldots;L_m:C_m\ ;L:od\ ;\delta)$ alors nous obtenons
$$[M \notin \underline{dom}(\mathbb{B}[\![B]\!])] \Rightarrow \tilde{I}(L)(M)$$

Nous pouvons récapituler ces conditions de vérifications comme suit :

- Commande nulle

$\beta L_1 :$ <u>skip</u> ; $L_2 : \delta$ — $P_2 \Rightarrow P_1$

- Commande d'affectation

$\beta L_1 : V := E\ ; L_2 : \delta$ — $\neg\, \underline{dom}(E) \Rightarrow P_1$

$P_2 \Rightarrow [\forall m \in \mathbb{D} .\ V=E[V \leftarrow m] \Rightarrow P_1[V \leftarrow m]]$

$\beta L_1 : V := ?\ ; L_2 : \delta$ — $P_2 \Rightarrow [\forall m \in \mathbb{D} .\ P_1[V \leftarrow m]]$

- Commande conditionnelle

$\beta L_1 :$ <u>if</u> B <u>then</u> — $\neg\, \underline{dom}(B) \Rightarrow P_1$

$L_2 :$ δ — $P_2 \Rightarrow [B \Rightarrow P_1]$

$L_3 :$

<u>else</u>

$L_4 :$ δ' — $P_4 \Rightarrow [\neg B \Rightarrow P_1]$

$L_5 :$

<u>fi</u>

$L_6 : \beta'$ — $P_6 \Rightarrow [P_3 \wedge P_5]$

- Commande itérative

$\beta L_1 :$ <u>while</u> B <u>do</u> — $\neg\, \underline{dom}(B) \Rightarrow P_1$

$L_2 :$ δ — $P_2 \Rightarrow [B \Rightarrow (P_1 \wedge P_3)]$

$L_3 :$

<u>od</u> ;

$L_4 : \beta'$ — $P_4 \Rightarrow [\neg B \Rightarrow (P_1 \wedge P_3)]$

- Point d'entrée

$L_1 : \beta$ — $\phi \Rightarrow \neg P_1$

Pour notre programme pris comme exemple

```
1:
   q:=0;
2:
   while x≥y do
3:
      q:=q+1;
4:
      x:=x-y;
5:
   od;
6:
```

nous pouvons montrer qu'il n'y a pas d'erreurs à l'exécution quand les valeurs initiales $\underline{x}, \underline{y}$ de x, y sont telles que $\underline{x}\geq 0 \wedge \underline{y}>0$ et $\mathcal{D}=[min, max]$ avec $min<0<max$.

Les conditions de vérification (après des simplifications triviales) sont:

$[(q+1)>max] \rightarrow P_6(x,y,q)$

$[((x-y)<min)\vee((x-y)>max)] \Rightarrow P_4(x,y,q)$

$P_2(x,y,q) \Rightarrow [(x<y)\Rightarrow(P_2(x,y,q)\wedge P_3(x,y,q))]$

$P_5(x,y,q) \Rightarrow P_4(x+y,y,q)$

$P_4(x,y,q) \Rightarrow P_3(x,y,q-1)$

$P_3(x,y,q) \Rightarrow [(x\geq y)\Rightarrow(P_2(x,y,q)\wedge P_5(x,y,q))]$

$P_2(x,y,q) \Rightarrow [\forall q'\in[min,max].(q=0)\Rightarrow P_1(x,y,q')]$

$[x\geq 0 \wedge y>0] \Rightarrow \neg P_1(x,y,q)$

Intuitivement, si initialement $\underline{x}\geq 0$ et $\underline{y}>0$ la seule erreur d'exécution possible est un débordement à la commande 3: q:=q+1;. Puisque q reste positif ceci peut arriver seulement si la valeur initiale $\underline{x}$ de x (c'est-à-dire $q\times y+x$ en terme des valeurs courantes des variables) est supérieure à max, en contradiction avec le fait que $\underline{x}\in\mathcal{D}=[min,max]$.

Ce raisonnement est formalisé en montrant que les invariants locaux suivants satisfont les conditions de vérification (où $x,y,q\in[min,max]$):

$P_1(x,y,q) = [\neg(x\geq 0 \wedge y>0)]$

$P_2(x,y,q) = P_5(x,y,q) = [(x\geq 0 \wedge y>0 \wedge q\geq 0)\Rightarrow(q\times y+x>max)]$

$P_3(x,y,q) = [(x\geq y>0 \wedge q\geq 0)\Rightarrow(q\times y+x>max)]$

$P_4(x,y,q) = [(x\geq y>0 \wedge q\geq 1)\Rightarrow((q-1)\times y+x>max)]$

$P_6(x,y,q) = ff$

4.3.2.1.9 Preuve d'invariance globale, par l'absurde pour des programmes séquentiels

Soit $Ps\in\mathcal{P}s$ un programme séquentiel. Un invariant global d'un programme Ps est une assertion $\delta\in((\mathcal{V}\rightarrow\mathcal{D})\rightarrow\{tt,ff\})$ sur les valeurs des variables qui est vraie tout le temps durant l'exécution du programme. C'est-à-dire

$[\forall p\in\Sigma^{<S[\![Ps]\!],A[\![Ps]\!],t[\![Ps]\!],\varepsilon>}, i\in|\vec{p}|.\ [\sigma(p_i)\Rightarrow\psi(p_i)]$

où

$\varepsilon(<L,M>) = [(Ps\equiv L:\beta)\wedge\phi(M)]$

$\sigma(<L,M>) = tt$

$\psi(<L,M>) = \delta(M)$

En choisissant le principe d'induction et le langage, pour exprimer les invariants locaux, considérés aux paragraphes 4.3.2.1 et 4.3.2.2, nous obtenons les conditions de vérifications du paragraphe 4.3.2.1.4 excepté pour la base:

$\tilde{C}^{\sim}_{\sigma}[\![Ps]\!](\psi,\delta)(\tilde{I})$

$= C_{\delta}[\![Ps]\!](\psi,\sigma)(\delta[\![Ps]\!](\tilde{I}))$

$= [\forall\tilde{s}\in S[\![Ps]\!].\ [\neg\psi(\tilde{s})\wedge\sigma(\tilde{s})]\Rightarrow\delta[\![Ps]\!](\tilde{I})(\tilde{s})]$

$= [\forall L\in C[\![Ps]\!], M\in M[\![Ps]\!].\ \neg\delta(M)\Rightarrow\tilde{I}(L)(M)]$

On écrit informellement (L_i désignant une étiquette quelconque du programme):

- Base

$\beta L_i:\beta' \qquad\qquad \neg\delta\Rightarrow P_i$

Pour notre programme pris comme exemple,

```
1:
        q:=0;
2:
        while x≥y do
3:
            q:=q+1;
4:
            x:=x-y;
5:
        od;
6:
```

les conditions de vérification sont les suivantes (où $x,y,q \in [min, max]$) :

$$\neg\delta(x,y,q) \Rightarrow P_i(x,y,q) \qquad i=1,\dots,6$$

$$P_6(x,y,q) \Rightarrow [(x<y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$$
$$P_5(x,y,q) \Rightarrow P_4(x+y,y,q)$$
$$P_4(x,y,q) \Rightarrow P_3(x,y,q-1)$$
$$P_3(x,y,q) \Rightarrow [(x\geq y) \Rightarrow (P_2(x,y,q) \wedge P_5(x,y,q))]$$
$$P_2(x,y,q) \Rightarrow [\forall q' \in [min,max].\ (q=0) \Rightarrow P_1(x,y,q')]$$

$$\phi(x,y,q) \Rightarrow \neg P_1(x,y,q)$$

Pour prouver que $\phi(x,y,q) = (x\geq 0 \wedge y>0 \wedge q\geq 0)$ est un invariant global de ce programme, nous pouvons choisir les invariants locaux suivants :

$$P_1(x,y,q) = P_2(x,y,q) = P_5(x,y,q) = [x<0 \vee y\leq 0 \vee q<0]$$
$$P_3(x,y,q) = P_4(x,y,q) = [(x\geq y) \Rightarrow (x<0 \vee y\leq 0 \vee q<0)]$$
$$P_6(x,y,q) = [(x<y) \Rightarrow (x<0 \vee y\leq 0 \vee q<0)]$$

4.3.2.2 Extension de la méthode de Morris-Wegbreit dite "Subgoal induction" aux programmes parallèles et généralisation à d'autres propriétés d'invariance

La méthode de Morris-Wegbreit [77] a été conçue pour démontrer la correction partielle de programmes séquentiels. Etant données des spécifications d'entrée $\phi \in ((\mathcal{V} \to \mathcal{D}) \to \{tt, ff\})$ et de sortie $\psi \in ((\mathcal{V} \to \mathcal{D})^2 \to \{tt, ff\})$ d'un programme Pr, il s'agit de démontrer la propriété d'invariance :

$$\forall p \in \Sigma^* \langle S[\![Pr]\!], A[\![Pr]\!], t[\![Pr]\!], \varepsilon \rangle, i \in |p|.\ [\sigma(p_i) \Rightarrow \Psi(p_0, p_i)]$$

où

$$\varepsilon(\langle L, M\rangle) = [(Pr \equiv L:\beta) \wedge \phi(M)]$$
$$\sigma(\langle L, M\rangle) = [Pr \equiv \beta L:]$$
$$\Psi(\langle \underline{L}, \underline{M}\rangle, \langle \bar{L}, \bar{M}\rangle) = \psi(\underline{M}, \bar{M})$$

Nous allons montrer dans un premier temps que l'essence de la méthode de Morris-Wegbreit [77] consiste à appliquer le principe d'induction $(\cdot)^{\mathcal{I}^{-1}}$, c'est-à-dire à une correspondance (α, γ) entre invariants près à démontrer que

$$[\exists I \in A_\Delta[\![Pr]\!].\ Cv[\![Pr]\!](\varepsilon,\sigma)(\Psi)(I)]$$

où

$$A_\Delta[\![Pr]\!] = (S[\![Pr]\!]^2 \to \{tt, ff\})$$
$$Cv[\![Pr]\!](\varepsilon,\sigma)(\Psi)(I) = [Cv_\sigma[\![Pr]\!](\sigma)(I) \wedge Cv_t[\![Pr]\!](\sigma)(I) \wedge Cv_\varepsilon[\![Pr]\!](\varepsilon,\sigma)(\Psi)(I)]$$

avec

$$Cv_\sigma[\![Pr]\!](\sigma)(I) = [\forall \bar{s} \in S[\![Pr]\!].\ \sigma(\bar{s}) \Rightarrow I(\bar{s},\bar{s})]$$
$$Cv_t[\![Pr]\!](\sigma)(I) = [\forall s, s', \bar{s} \in S[\![Pr]\!], a \in A[\![Pr]\!].\ (t_a[\![Pr]\!](s,s') \wedge I(s',\bar{s}) \wedge \sigma(\bar{s})) \Rightarrow I(s,\bar{s})]$$
$$Cv_\varepsilon[\![Pr]\!](\varepsilon,\sigma)(\Psi)(I) = [\forall \underline{s}, \bar{s} \in S[\![Pr]\!].\ (\varepsilon(\underline{s}) \wedge I(\underline{s},\bar{s}) \wedge \sigma(\bar{s})) \Rightarrow \Psi(\underline{s},\bar{s})]$$

Ensuite nous généraliserons la méthode aux programmes parallèles et à d'autres propriétés d'invariance (Cousot-R [81]).

4.3.2.2.1 Programmes séquentiels

Nous considérons des programmes séquentiels comme ils ont été définis au paragraphe 2.8.1.

4.3.2.2.1.1 Choix d'un langage pour exprimer les invariants locaux et sa sémantique

Nous choisissons

$$\tilde{A}_{\Delta}[\![Ps]\!] = \prod_{L \in C[\![Ps]\!]} (M[\![Ps]\!]^2 \to \{tt, ff\})$$

où

$$C[\![Ps]\!] = \{L \in \mathcal{L} : Ps \equiv \beta L : \delta\}$$

$$M[\![Ps]\!] = (\{x \in \mathcal{V} : Ps \equiv \alpha x \beta\} \to \mathcal{D})$$

pour pouvoir associer une relation sur les états mémoires à chaque point du programme. La signification de ce vecteur d'invariants locaux est définie par la fonction sémantique

$$\gamma[\![Ps]\!] \in (\tilde{A}_S[\![Ps]\!] \to A_{\Delta}[\![Ps]\!])$$

$$\gamma[\![Ps]\!](\tilde{I})(\langle L, M\rangle, \langle \bar{L}, \bar{M}\rangle) = \tilde{I}(L)(M, \bar{M})$$

Intuitivement, quand le contrôle est en L, la relation $\tilde{I}(L)$ est vraie entre l'état mémoire courant M et l'état mémoire (final) $\bar{M}$ (qui correspond au point $\bar{L}$ de sortie de Ps).

4.3.2.2.1.2 Dérivation de conditions de vérification correctes

Nous avons à construire $C\tilde{v}[\![Ps]\!]$ tel que

$$C\tilde{v}[\![Ps]\!](\varepsilon, \delta)(\tilde{\Psi})(\tilde{I}) \Rightarrow Cv[\![F:\Pi]\!](\varepsilon, \delta)(\Psi)(\gamma[\![P:\Pi]\!](\tilde{I}))$$

$Cv[\![Ps]\!]$ étant une conjonction de trois conditions, nous choisissons $C\tilde{v}[\![Ps]\!]$ satisfaisant le critère de correc-

4.3.2.2.1.2.1 Finalisation

$$Cv_f[\![Ps]\!](\delta)(\gamma[\![Ps]\!](\tilde{I}))$$

$$= [\forall \bar{\Delta} \in S[\![Ps]\!].\ \delta(\bar{\Delta}) \Rightarrow \gamma[\![Ps]\!](\tilde{I})(\bar{\Delta}, \bar{\Delta})]$$

$$= [\forall \bar{L} \in C[\![Ps]\!], \bar{M} \in M[\![Ps]\!].\ \delta(\langle \bar{L}, \bar{M}\rangle) \Rightarrow \gamma[\![Ps]\!](\tilde{I})(\langle \bar{L}, \bar{M}\rangle, \langle \bar{L}, \bar{M}\rangle)]$$

$$= [\forall \bar{L} \in C[\![Ps]\!], \bar{M} \in M[\![Ps]\!].\ (Ps \equiv \beta \bar{L} :) \Rightarrow \tilde{I}(\bar{L})(\bar{M}, \bar{M})]$$

$$= [\forall \bar{M} \in M[\![Ps]\!].\ \tilde{I}(\bar{L})(\bar{M}, \bar{M})] \quad \text{où} \quad Ps \equiv \alpha \bar{L} :$$

$$= C\tilde{v}_f[\![Ps]\!](\delta)(\tilde{I})$$

Intuitivement, cette condition de vérification établit que l'invariant $\tilde{I}(\bar{L})$ associé au point de sortie doit être vrai pour toute exécution qui se termine.

4.3.2.2.1.2.2 Induction

$$Cv_i[\![Ps]\!](\delta)(\gamma[\![Ps]\!](\tilde{I}))$$

$$= [\forall \Delta, \Delta', \bar{\Delta} \in S[\![Ps]\!], a \in A[\![Ps]\!].\ (t[\![Ps]\!]_a(\Delta, \Delta') \wedge \gamma[\![Ps]\!](\tilde{I})(\Delta', \bar{\Delta}) \wedge \delta(\bar{\Delta})) \Rightarrow \gamma[\![Ps]\!](\tilde{I})(\Delta, \bar{\Delta})]$$

$$= [\forall L \in C[\![Ps]\!], M, \bar{M} \in M[\![Ps]\!].$$

$$[\exists L' \in C[\![Ps]\!], M' \in M[\![Ps]\!].\ \underline{cond}[\![Ps]\!](L, L')(M) \wedge \underline{succ}[\![Ps]\!](L)(M, M') \wedge \tilde{I}(L')(M', \bar{M})] \Rightarrow \tilde{I}(L)(M, \bar{M})]$$

$$= C\tilde{v}_i[\![Ps]\!](\delta)(\tilde{I})$$

Intuitivement cette condition de vérification établit que si un pas atomique du programme peut conduire du point L où l'état mémoire est M à son successeur immédiat L' avec l'état mémoire M' tel que $\underline{succ}[\![Ps]\!](L)(M, M')$ alors l'invariant local $\tilde{I}(L')(M', \bar{M})$ après ce pas doit impliquer l'invariant local $\tilde{I}(L)(M, \bar{M})$ avant ce pas.

Suivant la nature de la commande étiquetée par L et en développant les définitions de $\underline{cond}[\![Ps]\!]$ et $\underline{succ}[\![Ps]\!]$, nous pourrons la simplifier en nous-cas. Par exemple, si L désigne une boucle while, nous obtenons :

$$[\,(Ps \equiv \beta L: \underline{\text{while}}\ B\ \underline{\text{do}}\ L': \delta) \wedge M \in \underline{dom}(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = tt \wedge \tilde{I}(L')(M,\bar{M})]$$
$$\Rightarrow \tilde{I}(L)(M,\bar{M})$$

et

$$[\,(Ps \equiv \beta L: \underline{\text{while}}\ B\ \underline{\text{do}}\ Ps'\ \underline{\text{od}};\ L': \delta) \wedge M \in \underline{dom}(\mathbb{B}[\![B]\!]) \wedge \mathbb{B}[\![B]\!](M) = ff \wedge \tilde{I}(L')(M,\bar{M})]$$
$$\Rightarrow \tilde{I}(L)(M,\bar{M})$$

4.3.2.2.1.2.3 Initialisation

$$Cv_\varepsilon[\![Ps]\!](\varepsilon,\sigma)(\Psi)(\delta[\![Ps]\!](\tilde{I}))$$
$$= [\forall \underline{a}, \bar{a} \in S[\![Ps]\!] .\ (\varepsilon(\underline{a}) \wedge \delta[\![Ps]\!](\tilde{I})(\underline{a},\bar{a}) \wedge \sigma(\bar{a})) \Rightarrow \Psi(\underline{a},\bar{a})]$$
$$= [\forall \underline{L}, \bar{L} \in C[\![Ps]\!],\ \underline{M},\bar{M} \in M[\![Ps]\!].$$
$$(Ps \equiv \underline{L}: \beta \wedge \tilde{I}(\underline{L})(\underline{M},\bar{M}) \wedge Ps \equiv \beta\bar{L}:) \Rightarrow (\phi(\underline{M}) \Rightarrow \psi(\underline{M},\bar{M}))]$$
$$= [(\phi(\underline{M}) \wedge \tilde{I}(\underline{L})(\underline{M},\bar{M})) \Rightarrow \psi(\underline{M},\bar{M})] \quad \text{où } Ps \equiv \underline{L}:\alpha$$
$$= \tilde{Cv}_\varepsilon[\![Ps]\!](\varepsilon,\sigma)(\Psi)(\tilde{I})$$

Intuitivement, la spécification d'entrée et l'invariant-local associé au point d'entrée doivent impliquer la spécification de sortie.

4.3.2.2.1.3 Vérification de la complétude sémantique

Définissons

$$\alpha[\![Ps]\!] \in (A\sigma[\![Ps]\!] \to \tilde{A\sigma}[\![Ps]\!])$$
$$\alpha[\![Ps]\!](I)(L)(M,\bar{M}) = [\forall \bar{L} \in C[\![Ps]\!] .\ (Ps \equiv \beta\bar{L}:) \Rightarrow I(\langle L,M\rangle, \langle \bar{L},\bar{M}\rangle)]$$

qui spécifie comment une hypothèse d'induction $I \in A\sigma[\![Ps]\!]$ peut être codée par un vecteur d'invariants locaux $\tilde{I}(L)$ associés à chaque point L du programme Ps.

Ayant choisi $\tilde{Cv}[\![Ps]\!](\varepsilon,\sigma)(\Psi) = Cv[\![Ps]\!](\varepsilon,\sigma)(\Psi) \circ \delta[\![Ps]\!]$, la vérification de la complétude sémantique est :

$\forall I \in A\sigma[\![Ps]\!] .\ \tilde{Cv}[\![Ps]\!](\varepsilon,\sigma)(\Psi)(\alpha[\![Ps]\!](I)) = Cv[\![Ps]\!](\varepsilon,\sigma)(\Psi)(\delta\circ\alpha[\![Ps]\!](I)) \Leftarrow Cv[\![Ps]\!](\varepsilon,\sigma)(\Psi)(I)$

car $Cv[\![Ps]\!](\varepsilon,\sigma)(\Psi)$ est monotone et $\delta\circ\alpha$ est extensive.

Ce résultat réfute l'argument de Misra [78] que « subgoal induction is not guaranteed to prove a correct program correct »

4.3.2.2.1.4 Résumé des conditions de vérification pour la preuve de correction partielle de programmes séquentiels par induction en arrière

Nous utiliserons des notations mnémoniques informelles. $\underline{x}$ et $\bar{x}$ désignent les vecteurs des valeurs respectivement initiales et finales des variables du programme et P_i est l'invariant associé au point L_i.

- Finalisation

βL_2 : $\quad \forall \underline{x}.\ P_2(\underline{x}, \underline{z})$

- Induction

  . Commande nulle

  βL_1 : skip ; $L_2 : \delta$ $\quad P_2 \Rightarrow P_1$

  . Commande d'affectation

  βL_1 : $V_i := E$; $L_2 : \delta$ $\quad P_2[V \leftarrow E] \Rightarrow P_1$

  βL_1 : $V_i := ?$; $L_2 : \delta$ $\quad \forall m \in \mathbb{D}.\ P_2[V \leftarrow m] \Rightarrow P_1$

  . Commande conditionnelle

  βL_1 : if B then

  L_2 : δ

  L_3 :

  else

  L_4 : δ' $\quad [(P_2 \wedge B) \vee (P_4 \wedge \neg B)] \Rightarrow P_1$

  L_5 : $\quad P_5 \Rightarrow (P_3 \wedge P_5)$

  fi ;

  $L_6 : \beta'$

  . Commande itérative

  βL_1 : while B do

  L_2 : δ $\quad [(P_2 \wedge B) \vee (P_4 \wedge \neg B)] \Rightarrow (P_1 \wedge P_3)$

  L_3 :

  od ;

  $L_4 : \beta'$

- Initialisation

L_1 : $\quad \forall \underline{x}, \underline{z}.\ [(\Phi(\underline{z}) \wedge P_1(\underline{x}, \underline{z})) \Rightarrow \Psi(\underline{x}, \underline{z})]$

4.3.2.2.1.5 Preuves d'autres propriétés d'invariance de programmes séquentiels par induction en arrière

Morris et Wegbreit affirment que "a drawback of subgoal induction is that it cannot be used to prove invariants about non-terminating programs". Le problème est de montrer que lorsque l'exécution d'un programme Ps commence dans un état mémoire initial $\underline{M}$ satisfaisant une spécification d'entrée ϕ et atteint un point L quelconque du programme avec l'état mémoire M, alors l'invariant $\psi(L)(M)$ doit être vrai. Formellement

$$\forall p \in \Sigma^{<S}[\![Ps]\!], A[\![Ps]\!], t[\![Ps]\!], \varepsilon\rangle, i \in |p|.\ [\sigma(p_i) \Rightarrow \Psi(p_0, p_i)]$$

où

$$\varepsilon(\langle L, M\rangle) = [(Ps \equiv L : \beta) \wedge \phi(M)]$$

$$\sigma(\langle L, M\rangle) = \mathit{ff}$$

$$\Psi(\langle \underline{L}, \underline{M}\rangle, \langle \bar{L}, \bar{M}\rangle) = \psi(\bar{L})(\bar{M})$$

Les définitions ci-dessus de σ et Ψ diffèrent du cas de la correction partielle et donc de "subgoal induction", de sorte que la remarque de Morris-Wegbreit est justifiée pour "subgoal induction" mais non pour toutes les méthodes de preuve par induction en arrière.

Pour être complet, construisons une méthode de preuve d'invariants par induction en arrière. Ψ est maintenant une assertion sur les états finaux (final signifiant quelconque dans ce cas). D'après 4.3.3, nous pourrions utiliser le principe d'induction (-I-) contrapositif en arrière. La démarche reste la même et les conditions de vérification sont similaires à celles du paragraphe 4.3.2.2.1.2 pour le cas d'induction 4.3.2.2.1.2.3 tandis que 4.3.2.2.1.2.1 et 4.3.2.2.1.2.2 deviennent :

- Finalisation

$$\rho L_i : \delta \qquad \neg\psi_i \Rightarrow P_i$$

- Initialisation

$$L_1 : \beta \qquad \phi \Rightarrow \neg P_1$$

Exemple

Illustrons cette méthode par le (contre) exemple simple de Morris-Wegbreit [77], qui consiste à montrer que $x>0$ dans le programme suivant :

```
1:
    x := 1
2:  while true do
3:      x := x+1;
4:  od;
5:
```

Nous choisissons $\phi(x) = tt$, $\psi_1(x) = tt$, $\psi_i(x) = [x>0]$, $i = 2,\dots,5$. Les conditions de vérification sont :

$\neg\psi_i \rightarrow P_i \qquad i = 1,\dots,5$

$P_2[x \leftarrow 1] \Rightarrow P_1$

$[(P_3 \wedge \underline{true}) \vee (P_5 \wedge \neg\underline{true})] \rightarrow (P_2 \wedge P_4)$

$P_4[x \leftarrow x+1] \Rightarrow P_3$

$\phi \Rightarrow \neg P_3$

Ces conditions de vérification sont trivialement satisfaites par $P_i = \neg\psi_i$, $i = 1,\dots,5$.

□

4.3.2.2.2 Programmes parallèles asynchrones

4.3.2.2.2.1 Choix d'un langage pour exprimer les invariants locaux et sa sémantique

Pour exprimer une relation entre états courants et finaux, nous associons une relation à chaque point du programme qui ne soit pas dans une section critique. Dans le prélude et le postlude, c'est une relation entre états mémoires courant et final. A chaque point d'un processus c'est une relation entre les valeurs courantes des états de contrôle des autres processus, l'état mémoire courant et l'état mémoire final. Ainsi pour un programme

$$Ppa \equiv Ps \,[\![\, Proc_0 \,\|\, \dots \,\|\, Proc_i \,\|\, \dots \,\|\, Proc_{m-1} \,]\!]\; ; Ps' \qquad (m>1)$$

nous choisissons

$$\tilde{A}_{\mathcal{L}}[\![Ppa]\!] = \{\tilde{I} : [\exists \underline{L} \in \mathcal{C}[\![Ps]\!].\ \tilde{I}(\underline{L}) \in (\mathcal{M}[\![Ppa]\!]^2 \rightarrow \{tt, ff\})]$$
$$\vee [\exists i \in m,\ L \in \mathcal{C}[\![Proc_i]\!].$$
$$\tilde{I}(i)(L) \in ((\underset{j \in (m \setminus i)}{\pi} \mathcal{C}[\![Proc_j]\!] \times \mathcal{M}[\![Ppa]\!]^2) \rightarrow \{tt, ff\})]$$
$$\vee [\exists \bar{L} \in \mathcal{C}[\![Ps']\!].\ \tilde{I}(\bar{L}) \in (\mathcal{M}[\![Ppa]\!]^2 \rightarrow \{tt, ff\})]\}$$

La signification d'un tel vecteur $\tilde{I}$ est définie formellement par la fonction sémantique $\mathcal{I}[\![Ppa]\!]$. Nous avons $\mathcal{I}[\![Ppa]\!](\tilde{I}) = I$ où, par cas :

- $I(\langle L, M\rangle, \langle \bar{L}, \bar{M}\rangle) = \tilde{I}(L)(M, \bar{M})$ quand $L \in (\mathcal{C}[\![Ps]\!] \cup \mathcal{C}[\![Ps']\!])$ et $Ppa \equiv \rho\bar{L}$:
- $I(\langle L_0, \dots, L_{m-1}, M\rangle, \langle \bar{L}, \bar{M}\rangle) = \underset{i \in m}{\wedge} \tilde{I}(L_i)(L_0, \dots, L_{i-1}, L_{i+1}, \dots, L_{m-1}, M, \bar{M})$

quand $L_i \in \mathcal{C}[\![Proc_i]\!]$, $i \in m$ et $Ppa \equiv \rho\bar{L}$:

($\mathcal{I}[\![Ppa]\!](\tilde{I})(\Delta, \bar{\Delta})$ n'a pas besoin d'être définie quand $\bar{\Delta}$ n'est pas un état final).

4.3.2.2.2.2 Construction de conditions de vérification correctes

Les conditions de finalisation et d'initialisation sont similaires à 4.3.2.2.1.2.1 et 4.3.2.2.1.2.3 respectivement et ne présentent pas de difficulté. Le point principal consiste à trouver $\check{co}_i[\![Ppa]\!]$ tel que :

$$\check{Co}_i[\![Ppa]\!](\sigma)(\check{I}) \rightarrow Co_i[\![Ppa]\!](\sigma)(\check{\gamma}[\![Ppa]\!](\check{I}))$$

où

$$Co_i[\![Ppa]\!](\sigma)(\check{\gamma}[\![Ppa]\!](\check{I}))$$

$$= [\forall a, a' \in S[\![Ppa]\!],\ \bar{a} \in A[\![Ppa]\!].\ (t[\![Ppa]\!]_{\bar{a}}(a,a') \wedge \check{\gamma}[\![Ppa]\!](\check{I})(a',\bar{a}) \wedge \sigma(a)) \Rightarrow \check{\gamma}[\![Ppa]\!](\check{I})(a,\bar{a})]$$

$$= [\forall a, a' \in S[\![Ppa]\!],\ \bar{M} \in M[\![Ppa]\!],\ a \in A[\![Ppa]\!].$$

$$(t[\![Ppa]\!]_a(a,a') \wedge \check{\gamma}[\![Ppa]\!](\check{I})(a', \langle \bar{l}, \bar{M} \rangle) \wedge Ppa \equiv p\bar{l}:) \Rightarrow \check{\gamma}[\![Ppa]\!](\check{I})(a, \langle \bar{l}, \bar{M} \rangle)]$$

Nous décomposons cette condition de vérification en sous-cas selon la forme de a définie par $S[\![Ppa]\!]$:

Cas 1 : $A = \langle L, M \rangle$ où $L \in C[\![Ps]\!]$ et $M \in M[\![Ppa]\!]$

Cas 1.1 : $\neg(Ps \equiv pL:)$

Ce cas a été traité en 4.3.2.2.1 pour les programmes séquentiels.

Cas 1.2 : $(Ps \equiv pL:)$

D'après la définition 2.8.2.2.4 de $t[\![Ppa]\!]_a$, A' est nécessairement de la forme $\langle\langle L_0, \ldots, L_{m-1}\rangle, M\rangle$ où $Ppq \equiv pL:[\![L_0 :: p_0 \| \ldots \| L_{m-1} :: p_{m-1}]\!] p'$ et $a = \not p$ de sorte que par définition de $\check{\gamma}[\![Ppa]\!]$, la condition de vérification est dans ce cas

$$\bigwedge_{i \in m} \check{I}(L_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M, \bar{M}) \Rightarrow \check{I}(L)(M, \bar{M})$$

Intuitivement, la conjonction des invariants d'entrée de chaque processus doit impliquer l'invariant de sortie du prélude.

Cas 2 : $A = \langle L, M \rangle$ où $L \in [\![Ps']\!]$ et $M \in M[\![Ppa]\!]$

Ce cas a été traité en 4.3.2.2.1 pour les programmes séquentiels.

Cas 3 : $A = \langle\langle L_0, \ldots, L_{m-1}\rangle, M\rangle$

où $Ppq \equiv Ps[\![Pra_0 \| \ldots \| Pra_{m-1}]\!]; Ps'$, $L_i \in C[\![Pra_i]\!]$ et $M \in M[\![Ppa]\!]$

Suivant la forme possible de A', nous distinguons deux cas :

Cas 3.1 : $A' = \langle L', M' \rangle$ où $L' \in (C[\![Ps]\!] \cup C[\![Ps']\!])$ et $M' \in (\mathcal{V} \rightarrow \mathcal{D})$

D'après la définition 2.8.2.2.4 de $t[\![Ppa]\!]_a$ nous avons nécessairement $M = M'$ et $Ppq \equiv p[\![p_0 L_0 :: \| \ldots \| p_{m-1} L_{m-1} :]\!]; L' : p'$ et $a = \not p'$ de sorte que par définition de $\check{\gamma}[\![Ppa]\!](\check{I})$, la condition de vérification est dans ce cas

$$\check{I}(L')(M, \bar{M}) \Rightarrow \bigwedge_{i \in m} \check{I}(L_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M, \bar{M})$$

Intuitivement, l'invariant d'entrée du postlude doit impliquer la la conjonction des invariants de sortie de chaque processus.

Cas 3.2 : $A' = \langle\langle L'_0, \ldots, L'_{m-1}\rangle, M'\rangle$ où $L'_i \in C[\![Pra_i]\!]$, $i \in m$ et $M' \in M[\![Ppa]\!]$

Par définition de $t[\![Ppa]\!]_a$, nous décomposons ce cas en deux sous-cas suivant que la transition de a à a' correspond ou non à l'exécution d'une section critique.

Cas 3.2.1 : Transition ne correspondant pas à une section critique

Par définition de $t[\![Ppa]\!]_a$ et $\check{\gamma}[\![Ppa]\!]$, la condition de vérification s'équivaut à :

$$[[\exists i \in m.\ Ppa \equiv p[\![Pra_0 \| \ldots \| Pra_i \| \ldots \| Pra_{m-1}]\!] p' \wedge (\forall j \in (m \setminus i).\ L'_j = L_j)$$

$$\wedge\ t[\![Pra_i]\!]_{\underline{a}}(\langle L_i, M\rangle, \langle L'_i, M'\rangle) \wedge \bigwedge_{j \in m} \check{I}(L'_j)(L'_0, \ldots, L'_{j-1}, L'_{j+1}, \ldots, L'_{m-1}, M', \bar{M})]$$

$$\Rightarrow \bigwedge_{k \in m} \check{I}(L_k)(L_0, \ldots, L_{k-1}, L_{k+1}, \ldots, L_{m-1}, M, \bar{M}]$$

Cette condition se décompose en une conjonction de conditions de vérification correspondant aux processus Pra_i, $i \in m$:

$$[[\underline{cond}[\![Pra_i]\!](L_i, L'_i)(M) \wedge \underline{succ}[\![Pra_i]\!](L_i)(M, M')$$

$$\wedge\ \check{I}(L'_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M', \bar{M})$$

$$\wedge \bigwedge_{j \in (m \setminus i)} \check{I}(L_j)(L_0, \ldots, L_{j-1}, L'_i, L_{i+1}, \ldots, L_{j-1}, L_{j+1}, \ldots, L_{m-1}, M', \bar{M})]$$

$$\Rightarrow \bigwedge_{k \in m} \check{I}(L_k)(L_0, \ldots, L_{k-1}, L_{k+1}, \ldots, L_{m-1}, M, \bar{M})]$$

De nouveau, cette condition se décompose en sous-cas suivant k :

Cas 3.2.1.1 : Preuve séquentielle $(k = i)$

$$[[\ \underline{cond}\ [\![Pra_i]\!](L_i, L'_i)(M) \wedge \underline{succ}\ [\![Pra_i]\!](L_i)(M,M')$$
$$\wedge\ \tilde{I}(L'_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M', \bar{M}) \wedge \underline{context}(i,i)]$$
$$\Rightarrow \tilde{I}(L_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M, \bar{M})]$$

où

$$\underline{context}(i,k) = \bigwedge_{j \in (m-1-\{i,k\})} \tilde{I}(L_j)(L_0, \ldots, L_{i-1}, L'_i, L_{i+1}, \ldots; L_{j-1}, L_{j+1}, \ldots, L_{m-1}, M', \bar{M})$$

Cette condition de vérification correspond au cas des preuves séquentielles (cf. 4.3.2.2.1) excepté pour le terme $\underline{context}(i,i)$. Elle signifie que si l'invariant $\tilde{I}(L'_i)$ est vrai après la transition du point L_i au point L'_i, alors l'invariant $\tilde{I}(L_i)$ doit être vrai avant cette transition. De plus, le terme $\underline{context}(i,i)$ établit que nous pouvons utiliser dans la preuve toute l'information disponible sur les autres processus Pra_j, $j \neq i$ avant la transition. Pour avoir des preuves séquentielles similaires dans les cas de programmes séquentiels ou parallèles, nous négligerons le terme $\underline{context}(i,i)$ et choisissons la condition de vérification plus forte :

$$[\underline{cond}\ [\![Pra_i]\!](L_i, L'_i)(M) \wedge \underline{succ}\ [\![Pra_i]\!](L_i)(M,M') \wedge$$
$$\tilde{I}(L'_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M', \bar{M})]$$
$$\Rightarrow \tilde{I}(L_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M, \bar{M})$$

Notons que cette simplification est correcte puisque la condition de vérification ci-dessus implique l'originale. C'est également sémantiquement complet car, intuitivement, l'information donnée par $\underline{context}(i,i)$ peut si nécessaire être incorporée en $\tilde{I}(L'_i)$ en chaque point L'_i de chaque processus Pra_i.

Cas 3.2.1.2 : Absence d'interférences $(k \neq i)$

Pour $k \in (n \setminus i)$, nous devons montrer :

$$[[\ \underline{cond}\ [\![Pra_i]\!](L_i, L'_i)(M) \wedge \underline{succ}\ [\![Pra_i]\!](L_i)(M,M')$$
$$\wedge\ \tilde{I}(L'_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M', \bar{M})$$
$$\wedge\ \tilde{I}(L_k)(L_0, \ldots, L_{i-1}, L'_i, L_{i+1}, \ldots, L_{k-1}, L_{k+1}, \ldots, M', \bar{M}) \wedge \underline{context}(i,k)]$$
$$\Rightarrow \tilde{I}(L_k)(L_0, \ldots, L_{k-1}, L_{k+1}, \ldots, L_{m-1}, M, \bar{M})]$$

Intuitivement, les invariants $\tilde{I}(L_k)$ dans un processus Pra_k ne doivent pas être invalidés par l'exécution de commandes dans d'autres processus Pra_i, $i \neq k$. Comme ci-dessus, il est correct (et complet) de négliger le terme $\underline{context}(i,k)$.

Nous devrions maintenant détailler ces conditions de vérification suivant la nature de la commande désignée par L_i. Ceci est simple et nous donnerons seulement les résultats en 4.3.2.2.4.

Cas 3.2.2 Transition correspondant à une section critique

Lorsque $Ppa \equiv Ps [\![Pra_0]\!] \ldots [\![Pra_i]\!] \ldots [\![Pra_{m-1}]\!]; Ps'$, $Pra_i \equiv pL_i : \$ Pc'\$; L'_i : P'$ $Ps'' \equiv \underline{L}_1 : \$ \bar{L}_2 :$, nous devons, d'après la condition de vérification, montrer que :

$$\underline{cs\text{-}proof}\ (\ t[\![Ps'']\!]^*(\langle \underline{L}_1, M\rangle, \langle \bar{L}_2, M'\rangle))$$

où

$$\underline{cs\text{-}proof}\ (P) =$$
$$[[\ P \wedge \tilde{I}(L'_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M', \bar{M}) \wedge$$
$$\bigwedge_{j \in (n \setminus i)} \tilde{I}(L_j)(L_0, \ldots, L_{i-1}, L'_i, L_{i+1}, \ldots, L_{j-1}, L_{j+1}, \ldots, L_{m-1}, M', \bar{M})]$$
$$\Rightarrow \bigwedge_{k \in n} \tilde{I}(L_k)(L_0, \ldots, L_{k-1}, L_{k+1}, \ldots, L_{m-1}, M, \bar{M})]$$

Comme c'est le cas dans toutes les preuves d'invariance, il n'est pas toujours nécessaire de caractériser exactement la relation $t[\![Ps'']\!]^*(\langle \underline{L}_1, M\rangle, \langle \bar{L}_2, M'\rangle)$ entre les états d'entrée et de sortie de la section critique. Une approximation ψ pourra être utilisée car la formule ci-dessus est égale à :

$$[\exists \psi \in (s[\![Ps'']\!]^2 \to \{tt, ff\}).$$
$$t[\![Ps'']\!]^*(\langle \underline{L}_1, M\rangle, \langle \bar{L}_2, M'\rangle) \Rightarrow \psi(\langle \underline{L}_1, M\rangle, \langle \bar{L}_2, M'\rangle)]$$
$$\wedge \underline{cs\text{-}proof}(\psi(\langle \underline{L}_1, M\rangle, \langle \bar{L}_1, M'\rangle))]$$

Puisque nous avons à montrer que ψ est invariant, nous pouvons appliquer le principe d'induction (-Y-). Nous obtenons :

$$[\exists \psi \in (s[\![Ps'']\!]^2 \to \{tt, ff\}).$$
$$[\exists J \in (s[\![Ps'']\!]^2 \to \{tt, ff\}).$$
$$(\forall M' \in (\mathcal{U} \to \mathcal{B}).\ J(\langle \bar{L}_2, M'\rangle, \langle \bar{L}_2, M'\rangle))$$
$$\wedge (\forall L_1, L_2 \in C[\![Ps'']\!],\ M_1, M_2, M' \in M[\![Ppa]\!].$$
$$(t[\![Ps'']\!](\langle L_1, M_1\rangle, \langle L_2, M_2\rangle) \wedge J(\langle L_2, M_2\rangle, \langle \bar{L}_2, M'\rangle)) \Rightarrow J(\langle L_1, M_1\rangle, \langle \bar{L}_2, M'\rangle))$$
$$\wedge (\forall M, M' \in (\mathcal{U} \to \mathcal{B}).\ J(\langle \underline{L}_1, M\rangle, \langle \bar{L}_2, M'\rangle) \Rightarrow \psi(\langle \underline{L}_1, M\rangle, \langle \bar{L}_2, M'\rangle))]$$
$$\wedge [\underline{cs\text{-}proof}(\psi(\langle \underline{L}_1, M\rangle, \langle \bar{L}_2, M'\rangle))]]$$

Après simplification pour éliminer ψ, nous obtenons

$$[\exists J \in (s[\![Ps'']\!]^2 \to \{tt, ff\}).$$
$$\forall M \in (\mathcal{U} \to \mathcal{B}).\ J(\langle \bar{L}_2, M'\rangle, \langle \bar{L}_2, M'\rangle)$$
$$\wedge \forall L_1, L_2 \in C[\![Ps'']\!],\ M_1, M_2, M' \in M[\![Ppa]\!].$$
$$(t[\![Ps'']\!](\langle L_1, M_1\rangle, \langle L_2, M_2\rangle) \wedge J(\langle L_2, M_2\rangle, \langle \bar{L}_2, M'\rangle)) \Rightarrow J(\langle L_1, M_1\rangle, \langle \bar{L}_2, M'\rangle)$$
$$\wedge \underline{cs\text{-}proof}(J(\langle \underline{L}_1, M\rangle, \langle \bar{L}_1, M'\rangle))]$$

Si nous appliquons 4.3.2.2.1.2.1 et 4.3.2.2.1.2.2, c'est équivalent à

$$[\exists \vec{J} \in \prod_{L \in C[\![Ps'']\!]} (M[\![Ppa]\!]^2 \to \{tt, ff\}).$$
$$\forall M' \in (\mathcal{U} \to \mathcal{B}).\ \vec{J}(\bar{L}_2)(M', M')$$
$$\wedge \forall L, L' \in C[\![Ps'']\!],\ M, M', M'' \in M[\![Ppa]\!].$$
$$(\underline{cond}[\![Ps'']\!](L, L')(M) \wedge \underline{succ}[\![Ps'']\!](L)(M, M'') \wedge \vec{J}(L')(M'', M')) \Rightarrow \vec{J}(L)(M, M')$$
$$\wedge \underline{cs\text{-}proof}(\vec{J}(\underline{L}_1)(M, M'))]$$

Ainsi la condition de vérification pour les sections critiques a été divisée en deux sous-problèmes. Premièrement, le corps Ps'' de la section critique doit être traité indépendamment de son contexte à invariants en chaque point L une relation $\vec{J}(L)(M, M')$ entre

l'état mémoire courant M et l'état mémoire final M', et à montrer que $\vec{J}$ est invariant en utilisant la méthode de preuve par induction en arrière récapitulée en 4.3.2.2.1.4. Puis faire la preuve $\underline{cs\text{-}proof}(\vec{J}(\underline{L}_1)(M, M'))$ où la section critique est considérée comme atomique et sa sémantique définie par $\vec{J}(\underline{L}_1)(M, M')$. Comme pour les autres commandes, cette preuve peut se décomposer en :

- une preuve séquentielle (en omettant $\underline{context}(i, i)$)
$$[\vec{J}(\underline{L}_1)(M, M') \wedge \vec{I}(L'_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M', \bar{M})]$$
$$\Rightarrow \vec{I}(L_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M, \bar{M})$$

- une preuve d'absence d'interférences (en omettant $\underline{context}(i, k)$)
$$[\vec{J}(\underline{L}_1)(M, M') \wedge \vec{I}(L'_i)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M', \bar{M}) \wedge$$
$$\vec{I}(L_k)(L_0, \ldots, L_{i-1}, L'_i, L_{i+1}, \ldots, L_{k-1}, L_{k+1}, \ldots, L_{m-1}, M', \bar{M})]$$
$$\Rightarrow \vec{I}(L_k)(L_0, \ldots, L_{k-1}, L_{k+1}, \ldots, L_{m-1}, M, \bar{M})$$

4.3.2.2.2.3 Vérification de la complétude sémantique

Définissons $\alpha[\![Ppa]\!] \in (As[\![Ppa]\!] \to \vec{As}[\![Ppa]\!])$ qui spécifie comment une hypothèse d'induction $I \in As[\![Ppa]\!]$ peut être codée par un vecteur d'invariants $\vec{I}(L)$ associés en chaque point L du programme Ppa

$$Ppa \equiv Ps[\![Pra_0 \| \ldots \| Pra_i \| \ldots \| Pra_{m-1}]\!]; Ps'$$

par cas :

- $\alpha[\![Ppa]\!](I)(L)(M, \bar{M}) = [\forall \bar{L} \in C[\![Ppa]\!].\ (Ppa \equiv p\bar{L}:) \Rightarrow I(\langle L, M\rangle, \langle \bar{L}, \bar{M}\rangle)]$
quand $L \in (C[\![Ps]\!] \cup C[\![Ps']\!])$

- $\forall i \in m,\ L \in C[\![Pra_i]\!]$,
$$\alpha[\![Ppa]\!](I)(i)(L)(L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M, \bar{M}) =$$
$$[\forall \bar{L} \in C[\![Ppa]\!].\ (Ppa \equiv p\bar{L}:) \Rightarrow I(\langle\langle L_0, \ldots, L_{i-1}, L, L_{i+1}, \ldots, L_{m-1}\rangle, M\rangle, \langle \bar{L}, \bar{M}\rangle)]$$

La preuve de complétude (sémantique) montre qu'il est complet d'omettre les termes context(i,i) dans la preuve séquentielle et context(i,i) dans la preuve d'absence d'interférences. Elle montre aussi que la preuve d'absence d'interférences peut être simplifiée en omettant le terme $\tilde{I}(L_R)$ à gauche de l'implication. (Cependant, en pratique ce terme est utile puisque $\tilde{I}(L'_i)$ pourra s'écrire plus simplement).

Nous devons montrer que :

$$\forall I \in Ao[\![Ppa]\!].\ Co[\![Ppa]\!](\varepsilon,\sigma)(\psi)(I) \to \tilde{Co}[\![Ppa]\!](\varepsilon,\sigma)(\psi)(\alpha[\![Ppa]\!](I))$$

La preuve suit les cas considérés en 4.3.2.2.2. Nous traitons uniquement le cas 3.2.1 (puisque le cas 3.2.2 est similaire tandis que les cas restants se traitent comme en 4.3.2.2.1.2).

Le terme correspondant à $\tilde{Co}[\![Ppa]\!](\varepsilon,\sigma)(\psi)(\alpha[\![Ppa]\!](I))$ dans le cas 3.2.1 est la conjonction d'une preuve séquentielle et d'une preuve d'absence d'interférences :

$$\begin{aligned}
&[\ [(\underline{cond}[\![Pra_i]\!](L_i,L'_i)(M) \wedge \underline{succ}[\![Pra_i]\!](L_i)(M,M') \wedge (\forall \bar{L} \in C[\![Ppa]\!].\\
&\qquad (Ppa \equiv p\bar{L}:) \Rightarrow I(\langle\langle L_0,\dots,L_{i-1},L'_i,L_{i+1},\dots,L_{m-1}\rangle,M'\rangle,\langle\bar{L},\bar{M}\rangle))\ )\\
&\quad \Rightarrow (\forall \bar{L} \in C[\![Ppa]\!].\ (Ppa \equiv p\bar{L}:) \Rightarrow I(\langle\langle L_0,\dots,L_{m-1}\rangle,M\rangle,\langle\bar{L},\bar{M}\rangle)\ ]\\
&\wedge\\
&\bigwedge_{k \in (m\setminus i)} [\ (\underline{cond}[\![Pra_k]\!](L_k,L'_k)(M) \wedge \underline{succ}[\![Pra_k]\!](L_k)(M,M')\\
&\qquad \wedge (\forall \bar{L} \in C[\![Ppa]\!].\ (Ppa \equiv p\bar{L}:) \Rightarrow I(\langle\langle L_0,\dots,L_{i-1},L'_i,L_{i+1},\dots,L_{m-1},M'\rangle,\langle\bar{L},\bar{M}\rangle)))\\
&\qquad \Rightarrow (\forall \bar{L} \in C[\![Ppa]\!].\ (Ppa \equiv p\bar{L}:) \Rightarrow I(\langle\langle L_0,\dots,L_{m-1}\rangle,M\rangle,\langle\bar{L},\bar{M}\rangle))\ ]\ ]
\end{aligned}$$

qui est impliqué par :

$$\begin{aligned}
&[[\ \underline{cond}[\![Pra_i]\!](L_i,L'_i)(M) \wedge \underline{succ}[\![Pra_i]\!](L_i)(M,M')\\
&\wedge (\forall \bar{L} \in C[\![Ppa]\!].\ (Ppa \equiv p\bar{L}:) \Rightarrow I(\langle\langle L_0,\dots,L_{i-1},L'_i,L_{i+1},\dots,L_{m-1}\rangle,M'\rangle,\langle\bar{L},\bar{M}\rangle))\ ]\\
&\Rightarrow (\forall \bar{L} \in C[\![Ppa]\!].\ (Ppa \equiv p\bar{L}:) \Rightarrow I(\langle\langle L_0,\dots,L_{m-1}\rangle,M\rangle,\langle\bar{L},\bar{M}\rangle))\ ]
\end{aligned}$$

qui est lui-même impliqué par

$$[\ \forall \bar{L} \in C[\![Ppa]\!].\ (Ppa \equiv p\bar{L}:) \Rightarrow (\underline{cond}[\![Pra_i]\!](L_i,L'_i)(M) \wedge \underline{succ}[\![Pra_i]\!](L_i)(M,M') \wedge \dots\ M'\rangle,\langle\bar{L},\bar{M}\rangle))\ ] \Rightarrow I(\langle\langle L_0,\dots,L_{m-1}\rangle,M\rangle,\langle\bar{L},\bar{M}\rangle)\ ]$$

4.3.2.2.4 Résumé des conditions de vérification pour la preuve de correction partielle de programmes parallèles asynchrones par induction en arrière

$$Ppa \equiv \alpha_0 \ell_0 : p_0 [\![Pra_0 \|\dots\| Pra_{i-1} \| \alpha \ell_{ij} : p \| Pra_{i+1} \|\dots\| Pra_{m-1}]\!]; \alpha_1 \ell_1 : p_1 ; \bar{\ell} :$$

Les invariants $P_0(x,\bar{x})$ associé au point ℓ_0 du prélude et $P_1(x,\bar{x})$ associé au point ℓ_1 du postlude relient la valeur courante x à la valeur finale $\bar{x}$ des variables (quand l'exécution est en $\bar{\ell}$). L'invariant de sortie $P_{\bar{\ell}}(\bar{x},\bar{x})$ porte sur la valeur finale $\bar{x}$ des variables. L'invariant $P_{ij}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{m-1},x,\bar{x})$ associé au point j du processus i relie les états de contrôle $c_0,\dots,c_{i-1},c_{i+1},\dots,c_{m-1}$ des autres processus, les valeurs courantes x et les valeurs finales $\bar{x}$ des variables.

- Preuve séquentielle

Les conditions de vérification pour le prélude, le postlude et chaque processus Pra_i, $i \in m$ sont les mêmes que pour les programmes séquentiels (cf. 4.3.2.2.1.4) plus

- Finalisation du parallélisme

$$\forall [\![p_0 \ell_0 : \| \dots \| p_{m-1} \ell_{m-1} :]\!]; \ell_f : p$$

$$P_f(x,\bar{x}) \Rightarrow \hat{\wedge}_{i \in m} P_i(\ell_0,\dots,\ell_{i-1},\ell_{i+1},\dots,\ell_{m-1},x,\bar{x})$$

- Section critique

$$\forall \ell_{i_1} : \{ \ell_{i_2} : \{ \ell_{i_3} : \dots \}; \ell_{i_k} : \}$$

A chaque point ℓ_{ij} du corps $\ell_{i2} : \beta\, \ell_{i3}$, un invariant $P_{ij}(x, x')$ relie les valeurs courantes x en ℓ_{ij} aux valeurs finales x' en ℓ_{i3} des variables. Les conditions de vérification sont celles des programmes séquentiels (excepté pour l'initialisation) :

$$[P_{i2}(x,x') \wedge P_{i4}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{m-1},x',\bar{x})]$$
$$\Rightarrow P_{i4}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{m-1},x,\bar{x})$$

. Initialisation du parallélisme

$$\alpha\, \ell_0 : [\![\, \ell_0 : \alpha_0 \,\|\, \dots \,\|\, \ell_{m-1} : \alpha_{m-1} \,]\!]\, \beta$$
$$\Big[\bigwedge_{i\in m} P_i(\ell_0,\dots,\ell_{i-1},\ell_{i+1},\dots,\ell_{m-1},x,\bar{x})\Big] \Rightarrow P_0(x,\bar{x})$$

- Preuve d'absence d'interférences

Pour tout point ℓ_{kj} de tout processus P_k, $k \in (m \vee i)$

. Commande nulle

$$\alpha\, \ell_{i1} : \underline{\text{skip}}\ ; \ell_{i2} : \beta$$
$$(P_{i2}[c_k \leftarrow \ell_{kj}] \wedge P_{kj}[c_i \leftarrow \ell_{i2}]) \Rightarrow P_{kj}[c_i \leftarrow \ell_{i1}]$$

. Commande d'affectation

$$\alpha\, \ell_{i1} : v := E\ ; \ell_{i2} : \beta$$
$$(P_{i3}[c_k \leftarrow \ell_{kj}, v \leftarrow E] \wedge P_{kj}[c_i \leftarrow \ell_{i2}, v \leftarrow E]) \Rightarrow P_{kj}[c_i \leftarrow \ell_{i1}]$$

$$\alpha\, \ell_{i1} : v := ?\ ; \ell_{i2} : \beta$$
$$\forall m \in \mathcal{D}.\ (P_{i2}[c_k \leftarrow \ell_{kj}, v \leftarrow m] \wedge P_{kj}[c_i \leftarrow \ell_{i2}, v \leftarrow m]) \Rightarrow P_{kj}[c_i \leftarrow \ell_{i1}]$$

. Commande conditionnelle

$\alpha\, \ell_{i1}$: *if* B *then*
ℓ_{i2} :
ℓ_{i3} : β
else
ℓ_{i4} : γ
ℓ_{i5} :
fi;
$\ell_{i6} : \delta$

$$[(P_{i2}[c_k \leftarrow \ell_{kj}] \wedge B \wedge P_{kj}[c_i \leftarrow \ell_{i2}]) \vee (P_{i4}[c_k \leftarrow \ell_{kj}] \wedge \neg B \wedge P_{kj}[c_i \leftarrow \ell_{i4}])] \rightarrow P_{kj}[c_i \leftarrow \ell_{i1}]$$

$$(P_{i6}[c_k \leftarrow \ell_{kj}] \wedge P_{kj}[c_i \leftarrow \ell_{i6}]) \rightarrow (P_{kj}[c_i \leftarrow \ell_{i3}] \wedge P_{kj}[c_i \leftarrow \ell_{i5}])$$

. Commande itérative

$\alpha\, \ell_{i1}$: *while* B *do*
ℓ_{i2} :
ℓ_{i3} : β
od;
$\ell_{i4} : \delta$

$$[(P_{i2}[c_k \leftarrow \ell_{kj}] \wedge B \wedge P_{kj}[c_i \leftarrow \ell_{i2}]) \vee (P_{i4}[c_k \leftarrow \ell_{kj}] \wedge \neg B \wedge P_{kj}[c_i \leftarrow \ell_{i4}])] \rightarrow [P_{kj}[c_i \leftarrow \ell_{i1}] \wedge P_{kj}[c_i \leftarrow \ell_{i3}]]$$

. Section critique

$\alpha\, \ell_{i1}$: $\sharp$
ℓ_{i2} :
ℓ_{i3} : β
ℓ_{i4} : γ $\flat$;

$$[P_{i2}(x,x') \wedge P_{i4}(c_0,\dots,c_{k-1},\ell_{kj},c_{k+1},\dots,c_{i-1},c_{i+1},\dots,c_{m-1},x',\bar{x})$$
$$\wedge P_{kj}(c_0,\dots,c_{k-1},c_{k+1},\dots,c_{i-1},\ell_{i4},c_{i+1},\dots,c_{m-1},x',\bar{x})\ ]$$
$$\Rightarrow P_{kj}(c_0,\dots,c_{k-1},c_{k+1},\dots,c_{i-1},\ell_{i1},c_{i+1},\dots,c_{m-1},x,\bar{x})$$

4.3.2.2.2.5 Exemples

Exemple 4.3.2.2.2.5-1

Un exemple très simple pris dans Owicki-Gries [76]. Nous devons montrer que le programme suivant :

```
0:
  [[
    11: <
          12:
            x := x+1;
          13;
        >;
    14:
  ||
    21: <
          22:
            x:=x+1;
          23:
        >;
    24:
  ]];
3:
```

est partiellement correct pour :

$\phi(\underline{x}) = (\underline{x} = 0)$

$\psi(\underline{x}, \bar{x}) = (\bar{x} = 2)$

Les conditions de vérification sont les suivantes :

- Finalisation :

  $P_3(\bar{x}, \bar{x})$

  $P_3(x, \bar{x}) \Rightarrow [P_{24}(14, x, \bar{x}) \wedge P_{14}(24, x, \bar{x})]$

- Preuve séquentielle pour le processus 2 :

  $P_{23}(x', x')$

  $P_{23}(x+1, x') \Rightarrow P_{22}(x, x')$

  $[P_{22}(x, x') \wedge P_{24}(c_1, x', \bar{x})] \Rightarrow P_{21}(c_1, x, \bar{x})$

- Absence d'interférences du processus 2 avec la preuve du processus 1 :

  $[P_{22}(x, x) \wedge P_{24}(11, x', \bar{x}) \wedge P_{11}(24, x', \bar{x})] \Rightarrow P_{11}(21, x, \bar{x})$

  $[P_{22}(x, x') \wedge P_{24}(14, x', \bar{x}) \wedge P_{14}(24, x', \bar{x})] \Rightarrow P_{14}(21, x, \bar{x})$

- De la même manière, nous avons une preuve séquentielle pour le processus 1 et une preuve d'absence d'interférences du processus 1 avec la preuve du processus 2.

- Initialisation :

  $[P_{11}(21, x, \bar{x}) \wedge P_{21}(11, x, \bar{x})] \Rightarrow P_0(x, \bar{x})$

  $[\phi(x) \wedge P_0(x, \bar{x})] \Rightarrow \psi(x, \bar{x})$

Schéma de la preuve :

```
0: {x=x̄-2}
  [[
    11: {(c2=21 ∧ x=x̄-2)∨(c2=24 ∧ x=x̄-1)}
        <
          12: {x=x'-1}
              x:=x+1;
          13: {x=x'}
        >;
    14: {(c2=21 ∧ x=x̄-1)∨(c2=24 ∧ x=x̄)}
  ||
    21: {(c1=11 ∧ x=x̄-2)∨(c1=14 ∧ x=x̄-1)}
        <
          22: {x=x'-1}
              x:=x+1;
          23: {x=x'}
        >;
    24: {(c1=11 ∧ x=x̄-1)∨(c1=14 ∧ x=x̄)}
  ]];
3: {x=x̄}
```

Exemple 4.3.2.2.2.5-2

Le programme parallèle asynchrone suivant calcule $f=n!$ quand $n \geq 1$.

```
0:
1: n1:=1; n2:=n;
   [[        11: f1:=1;
             12: while (n1+2)<n2 do
             13: n1:=n1+1;
             14: f1:=f1×n1;
             15: od;
   ||
             21: f2:=n2;
             22: while (n1+2)<n2 do
             23: n2:=n2-1;
             24: f2:=f2×n2;
             25: od;
             26:
   ]];
3:
4: if (n1+1)=n2 then f:=f1×f2; else f:=f1×f2×(n1+1); fi;
```

Les conditions de vérification sont les suivantes :

- Finalisation :

$$\forall \bar{m},\bar{m}_1,\bar{m}_2,\bar{f}_1,\bar{f}_2,\bar{f}\ .\ P_4(\langle \bar{m},\bar{m}_1,\bar{m}_2,\bar{f}_1,\bar{f}_2,\bar{f}\rangle,\langle \bar{m},\bar{m}_1,\bar{m}_2,\bar{f}_1,\bar{f}_2,\bar{f}\rangle)$$

$$[(P_4[\bar{f}\leftarrow \bar{f}_1\times\bar{f}_2]\wedge(m_1+1)=m_2)\vee(P_4[\bar{f}\leftarrow \bar{f}_1\times\bar{f}_2\times(m_1+1)]\wedge(m_1+1)\neq m_2)]\Rightarrow P_3$$

$$P_3\Rightarrow(P_{26}[c_1\leftarrow 16]\wedge P_{16}[c_2\leftarrow 26])$$

- Preuve séquentielle du processus 2 :

$$[(P_{23}\wedge(m_1+2)<m_2)\vee(P_{26}\wedge(m_1+2)\geq m_2)]\Rightarrow[P_{22}\wedge P_{25}]$$

$$P_{25}[f_2\leftarrow f_2\times m_2]\Rightarrow P_{24}$$

$$P_{24}[m_2\leftarrow m_2-1]\Rightarrow P_{23}$$

$$P_{22}[f_2\leftarrow m_2]\Rightarrow P_{21}$$

- Preuve d'absence d'interférences du processus 2 avec la preuve du processus 1 :

Pour $j=1,\dots,6$

$$[(P_{23}[c_1\leftarrow 1j]\wedge(m_1+2)<m_2\wedge P_{1j}[c_2\leftarrow 23])\vee(P_{26}[c_1\leftarrow 1j]\wedge(m_1+2)\geq m_2\wedge P_{1j}[c_2\leftarrow 26])]$$

$$\Rightarrow(P_{1j}[c_2\leftarrow 23]\wedge P_{1j}[c_2\leftarrow 25])$$

$$(P_{25}[c_1\leftarrow 1j, f_2\leftarrow f_2\times m_2]\wedge P_{1j}[c_2\leftarrow 25, f_2\leftarrow f_2\times m_2])\Rightarrow P_{1j}[c_2\leftarrow 24]$$

$$(P_{24}[c_1\leftarrow 1j, m_2\leftarrow m_2-1]\wedge P_{1j}[c_2\leftarrow 24, m_2\leftarrow m_2-1])\Rightarrow P_{1j}[c_2\leftarrow 23]$$

$$(P_{22}[c_1\leftarrow 1j, f_2\leftarrow m_2]\wedge P_{1j}[c_2\leftarrow 23, f_2\leftarrow m_2])\Rightarrow P_{1j}[c_2\leftarrow 21]$$

- La preuve séquentielle du processus 1 et la preuve d'absence d'interférences du processus 1 avec la preuve du processus 2 sont

- Initialisation :

$$(P_{11}[c_2\leftarrow 21]\wedge P_{21}[c_1\leftarrow 11])\Rightarrow P_1$$

$$P_1[n_1\leftarrow 1, n_2\leftarrow n]\Rightarrow P_0$$

$$(n>1\wedge P_0)\Rightarrow(\bar{f}=\bar{n}!\wedge\bar{n}=n)$$

- esquisse de la preuve est :

Nous posons $\pi(a,b)=(a\leq b\rightarrow a\times(a-1)\times\dots\times b\mid 1)$ et

$$I=\big(m=\bar{m}\wedge[(\bar{m}_1+1=m_2\wedge\bar{f}=\bar{f}_1\times\bar{f}_2)\vee(\bar{m}_1+1\neq\bar{m}_2\wedge\bar{f}=\bar{f}_1\times\bar{f}_2\times(\bar{m}_1+1))]\big).$$

```
0: {(n>1)⇒(f̄=n! ∧ n̄=n)}
   n1:=1; n2:=n;
1: {(n1<n2)⇒(1≤n̄2-n̄1≤2 ∧ f̄1=π(n1+1,n̄1) ∧ f̄2=π(n̄2,n2) ∧ I)}
   [[
     11: {[(c2∈{21,22} ∧ n1<n2)∨(c2=23 ∧ n1+2<n2)∨(c2∈{24,25} ∧ n1+1<n2) ∨
           (c2=26 ∧ n1<n2≤n1+2)]⇒[f̄1=π(n1+1,n̄1) ∧ Aup(n1,n̄2-2)≤n̄1<n̄2 ∧ I]}
         f1:=1;
     12: {[(c2∈{21,22} ∧ n1<n2)∨(c2=23 ∧ n1+2<n2)∨(c2∈{24,25} ∧ n1+1<n2) ∨
           (c2=26 ∧ n1<n2≤n1+2)]⇒[f̄1=f1×π(n1+1,n̄1) ∧ Aup(n1,n̄2-2)≤n̄1<n̄2 ∧ I]}
         while (n1+2)<n2 do
     13: {[(c2∈{21,22,23} ∧ n1+2<n2)∨(c2∈{24,25,26} ∧ n1+1<n2)]
           ⇒[f̄1=f1×π(n1+1,n̄1) ∧ Aup(n1+1,n̄2-2)≤n̄1<n̄2 ∧ I]}
         n1:=n1+1;
     14: {[(c2∈{21,22,23} ∧ n1+1<n2)∨(c2∈{24,25,26} ∧ n1<n2)]
           ⇒[f̄1=f1×π(n1,n̄1) ∧ Aup(n1,n̄2-2)≤n̄1<n̄2 ∧ I]}
         f1:=f1*n1;
     15: {[(c2∈{21,22,23} ∧ n1+1<n2)∨(c2∈{24,25,26} ∧ n1<n2)]
           ⇒[f̄1=f1×π(n1+1,n̄1) ∧ Aup(n1,n̄2-2)≤n̄1<n̄2 ∧ I]}
         od;
     16: {[1≤n2-n1≤2]⇒[n̄2-2≤n1=n̄1<n̄2 ∧ f1=f̄1 ∧ I]}
   ||
     21: {[(c1∈{11,12} ∧ n1<n2)∨(c1=13 ∧ n1+2<n2)∨(c1∈{14,15} ∧ n1+1<n2) ∨
           (c1=16 ∧ n1<n2≤n1+2)]⇒[f̄2=π(n̄2,n2) ∧ n̄1<n̄2≤inf(n̄1+2,n2) ∧ I]}
         f2:=n2;
     22: {[(c1∈{11,12} ∧ n1<n2)∨(c1=13 ∧ n1+2<n2)∨(c1∈{14,15} ∧ n1+1<n2) ∨
           (c1=16 ∧ n1<n2≤n1+2)]⇒[f̄2=π(n̄2,n2-1)×f2 ∧ n̄1<n̄2≤inf(n̄1+2,n2) ∧ I]}
         while (n1+2)<n2 do
     23: {[(c1∈{11,12,13} ∧ n1+2<n2)∨(c1∈{14,15,16} ∧ n1+1<n2)]
           ⇒[f̄2=π(n̄2,n2-1)×f2 ∧ n̄1<n̄2≤inf(n̄1+2,n2-1) ∧ I]}
         n2:=n2-1;
     24: {[(c1∈{11,12,13} ∧ n1+1<n2)∨(c1∈{14,15,16} ∧ n1<n2)]
           ⇒[f̄2=π(n̄2,n2)×f2 ∧ n̄1<n̄2≤inf(n̄1+2,n2) ∧ I]}
         f2:=f2*n2;
     25: {[(c1∈{11,12,13} ∧ n1+1<n2)∨(c1∈{14,15,16} ∧ n1<n2)]
           ⇒[f̄2=π(n̄2,n2-1)×f2 ∧ n̄1<n̄2≤inf(n̄1+2,n2) ∧ I]}
         od;
     26: {[1≤n2-n1≤2]⇒[f̄2=f2 ∧ n̄1<n̄2=n2≤n̄1+2 ∧ I]}
   ]];
3: {I ∧ n1=n̄1 ∧ n2=n̄2 ∧ f1=f̄1 ∧ f2=f̄2 ∧ I}
4: if (n1+1)=n2 then f:=f1×f2; else f:=f̄1×f2×(n1+1); fi;
   {f=n̄ ∧ n1=n̄1 ∧ n2=n̄2 ∧ f1=f̄1 ∧ f2=f̄2 ∧ f=f̄}
```

Remarque 4.3.2.2.2.5-3 (Un principe d'induction avant-arrière symétrique)

Les invariants associés en chaque point de chaque processus sont de la forme $A \Rightarrow R$ où A dépend des valeurs courantes des états de contrôle des autres processus et des valeurs courantes des variables et R est une relation entre les valeurs courantes et finales des variables. A décrit ce qui a été accompli jusqu'ici et R décrit ce qui reste à faire. Ceci aurait été encore plus clair si nous avions utilisé des invariants redondants qui décrivent plus précisément le comportement du programme, par exemple

4: $\{[m>1 \wedge f=m!] \Rightarrow [n=\bar{n} \wedge n_1=\bar{n}_1 \wedge n_2=\bar{n}_2 \wedge f_1=\bar{f}_1 \wedge f_2=\bar{f}_2 \wedge f=\bar{f}]\}$

23: $\{[[(c_1=11 \wedge n_1=1 \wedge n_1+2<n_2) \vee (c_1 \in \{12,13\} \wedge n_2>1 \wedge f_2=n_1! \wedge n_1+2<n_2) \vee$

$(c_1=14 \wedge n_1>1 \wedge f_2=(n_1-1)! \wedge n_1+1<n_2) \vee (c_1 \in \{15,16\} \wedge n_2>1 \wedge f_2=n_1! \wedge n_1+1<n_2)]$

$\wedge [n_2 \leq n \wedge f_{23}=\pi(n_2,n)]] \Rightarrow [\bar{f}_2=\pi(\bar{n}_2,n_2-1)\times f_2 \wedge \bar{n}_1<\bar{n}_2 \leq \inf(\bar{n}_1+2,n_2) \wedge I]\}$

Cette possibilité offerte par l'induction en arrière devrait être contrastée avec l'induction en avant (Lamport [77], Owicki-Gries [76a] qui permet de spécifier ce qui a été fait (i.e. A) mais non ce qui reste à faire (i.e. R). Alors pour comprendre le programme, il faut inventer ce qui reste à faire à partir de ce qui a été fait et de la spécification de sortie.

Cette constatation pour le programme "factorielle" est en fait générale. La preuve est que $J=(A \Rightarrow R)$ où $A(\underline{A}) = [\exists \underline{\Lambda} \in S[\![P_2]\!] . \ \varepsilon(\underline{\Lambda}) \wedge t[\![P_2]\!]^*(\underline{\Lambda},\underline{A})]$ et $R(\underline{A},\bar{A}) = [t[\![P_2]\!]^*(\underline{A},\bar{A}) \wedge \sigma(\bar{A})]$, est toujours un invariant pour l'induction positive en arrière ($\dashv^{-1}$). Noter que nous aurions pu démontrer la complétude sémantique pour l'induction asertionnelle en avant ($\vdash$) en utilisant A. Alors en utilisant le principe d'induction en arrière ($\dashv^{-1}$) on peut spécifier le maximum d'information A sur l'état courant du programme, qui pourrait se faire en utilisant le principe d'induction en avant ($\vdash$), plus une certaine information R sur ce qui reste à faire. Ceci ne permet pas toutefois de spécifier de relation entre

l'état courant et l'état initial du programme (par exemple dans le programme 4.3.2.2.2.5-2, on peut exprimer que $n=\underline{n}$ à la ligne 0: et donc que les valeurs initiales $\underline{n}$ et finales $\bar{n}$ de n sont égales, mais on ne peut pas exprimer que $\underline{n}=n=\bar{n}$ en tout point du programme). Pour ce faire nous proposons d'utiliser un invariant $K(\underline{A},A,\bar{A})$ qui est la conjonction de l'invariant $I(\underline{A},A)$ utilisé dans le principe d'induction ($\vdash$) et de l'invariant $J(A,\bar{A})$ utilisé dans le principe d'induction ($\dashv^{-1}$), ce qui conduit au principe d'induction suivant :

$$
\begin{array}{l}
[\exists K \in (S^3 \to \{\mathrm{tt},\mathrm{ff}\}) . \ \forall \underline{A},A,A',\bar{A} \in S, a \in A. \\
\quad [\varepsilon(\underline{A}) \wedge \sigma(\underline{A})] \Rightarrow [K(\underline{A},\underline{A},\bar{A}) \wedge K(\underline{A},\bar{A},\bar{A})] \\
\quad \wedge \\
\quad [\varepsilon(\underline{A}) \wedge K(\underline{A},A',\bar{A}) \wedge t_a(A',A) \wedge \sigma(\bar{A})] \Leftrightarrow \\
\qquad [\varepsilon(\underline{A}) \wedge t_a(A',A) \wedge K(\underline{A},A,\bar{A}) \wedge \sigma(\bar{A})] \\
\quad \wedge \\
\quad [\varepsilon(\underline{A}) \wedge (K(\underline{A},\underline{A},\bar{A}) \vee K(\underline{A},\bar{A},\bar{A})) \wedge \sigma(\bar{A})] \Rightarrow \psi(\underline{A},\bar{A})] \\
\Longleftrightarrow \\
[\forall p \in \Sigma^{<S,A,t,\varepsilon>}, i \in |p| . \ (\sigma(p_i) \Rightarrow \psi(p_0,p_i))^{\mathrm{vf}}]
\end{array}
\qquad (\dashv\!\!\dashv^{-1})
$$

Pour la preuve de correction, nous avons $[\forall p \in \Sigma^{<S,A,t,\varepsilon>}, i \in |p|, j \in i . \ (\sigma(p_i) \Rightarrow K(p_0,p_j,p_i))]$. Pour la preuve de complétude sémantique, il suffit de choisir $K(\underline{A},A,\bar{A}) = [\exists p \in \Sigma^{<S,A,t,\varepsilon>}, i \in |p|, j \in i . \ p_0=\underline{A} \wedge p_j=A \wedge p_i=\bar{A} \wedge \sigma(\bar{A})]$.

□

4.3.2.2.3 Construction d'une méthode d'absence d'interblocages dans les programmes parallèles asynchrones par induction en arrière

La méthode de preuve developpée au paragraphe 4.3.2.2.2 pour les programmes parallèles asynchrones se généralise sans difficultés pour les programmes parallèles synchrones définis en 2.8.5. Par exemple pour la sémantique libérale des sémaphores considérée en 2.8.5.3, nous obtenons les conditions de vérification suivantes :

- Preuve séquentielle
  - $\alpha \ell_{i_1} : \underline{P}(Se);\ \ell_{i_2} : P$
    $(Se > 0 \wedge P_{i_2}[Se \leftarrow Se-1]) \Rightarrow P_{i_1}$
  - $\alpha \ell_{i_1} : \underline{V}(Se);\ \ell_{i_2} : P$
    $P_{i_2}[Se \leftarrow Se+1] \Rightarrow P_{i_1}$

- Preuve d'absence d'interférences

(pour chaque point ℓ_{k_j} de chaque processus Pro_k, $k \in (m \vee i)$) :

  - $\alpha \ell_{i_1} : \underline{P}(Se);\ \ell_{i_2} : P$
    $(Se > 0 \wedge P_{i_2}[c_k \leftarrow \ell_{k_j}, Se \leftarrow Se-1] \wedge P_{k_j}[c_i \leftarrow \ell_{i_2}, Se \leftarrow Se-1]) \Rightarrow P_{k_j}[c_i \leftarrow \ell_{i_1}]$
  - $\alpha \ell_{i_1} : \underline{V}(Se);\ \ell_{i_2} : P$
    $(P_{i_2}[c_k \leftarrow \ell_{k_j}, Se \leftarrow Se+1] \wedge P_{k_j}[c_i \leftarrow \ell_{i_2}, Se \leftarrow Se+1]) \Rightarrow P_{k_j}[c_i \leftarrow \ell_{i_1}]$

L'exécution d'un programme synchrone est globalement bloquée si les processus n'ont pas tous terminé et si tous les processus dont l'exécution n'est pas terminée sont en attente pour prendre un sémaphore. Plus formellement, si

$Pps \equiv Ps [\![Pro_0 \| \dots \| Pro_{m-1}]\!]; Ps'$

Pps est (globalement) bloqué dans l'état s si et seulement si $\beta [\![Pps]\!](s)$ est vrai, avec :

$$\beta [\![Pps]\!](s) = [\exists L_0 \in C[\![Pro_0]\!], \dots, L_{m-1} \in C[\![Pro_{m-1}]\!],\ M \in M[\![Pps]\!] .\ s = \langle\langle L_0, \dots, L_{m-1} \rangle, M \rangle \wedge [[\forall i \in m .\ (Pro_i \equiv \alpha L_i : \underline{P}(Se); S \wedge M(Se) \leq 0) \vee (Pro_i \equiv \alpha L_i :)] \wedge [\exists j \in m .\ \neg (Pro_j \equiv \alpha L_j :)]]]$$

Un programme Pps est exempt d'interblocage si aucune exécution de Pps ne conduit à un état où Pps est bloqué, c'est-à-dire :

$\forall p \in \Sigma < S[\![Pps]\!], A[\![Pps]\!], t[\![Pps]\!], \varepsilon >,\ i \in |p| .\ \neg \beta [\![Pps]\!](p_i)$

C'est une propriété d'invariance où $\varphi(\bar{s}) = tt$ et $\psi(\underline{s}, \bar{s}) = \neg \beta [\![Pps]\!](\bar{s})$ ne dépend pas des états initiaux. D'après 4.2.1.3, l'absence d'interblocage peut être prouvée par induction en arrière, en utilisant le principe d'induction contrapositif $(\dashv \bar{I})$.

Le choix d'un langage pour exprimer les invariants locaux et sa sémantique est similaire à celui de la correction partielle (cf. 4.3.2.2.2.1) excepté qu'au point L_i du processus i, nous avons un invariant local de la forme :

$\check{I}(L_i)(L_0, \dots, L_{i-1}, L_{i+1}, \dots, L_{m-1}, M)$

au lieu de

$\check{I}(L_i)(L_0, \dots, L_{i-1}, L_{i+1}, \dots, L_{m-1}, M, \bar{M})$

puisqu'il n'est plus nécessaire de relier les états courants et finaux.

La dérivation des conditions de vérification correspondant au pas d'induction de $(\dashv \bar{I})$ est la même que dans les paragraphes 4.3.2.2.2 et 4.3.2.2.3 excepté que les états mémoires finaux sont omis. Les cas qui restent sont :

. Initialisation

$[\forall \underline{\lambda} \in S[\![Pps]\!]\,.\ \varepsilon(\underline{\lambda}) \Rightarrow \neg\, \gamma[\![Pps]\!](\tilde{I})(\underline{\lambda})]$

$= [\forall \underline{L} \in C[\![Pps]\!], \underline{M} \in (\mathcal{V} \to \mathcal{N})\,.\ (Pps \equiv \underline{L} : \alpha \wedge \phi(\underline{M})) \Rightarrow \neg \tilde{I}(\underline{L})(\underline{M})]$

. Finalisation

$[\forall \bar{\lambda} \in S[\![Pps]\!]\,.\ \beta[\![Pps]\!](\bar{\lambda}) \Rightarrow \gamma[\![Pps]\!](\tilde{I})(\bar{\lambda})]$

quand $\bar{\lambda}$ n'est pas de la forme $\langle\langle L_0, \dots, L_{m-1}\rangle, M\rangle$, $\beta[\![Pps]\!](\bar{\lambda})$ est faux de sorte que la condition est trivialement vérifiée, sinon elle est équivalente à :

$[\forall L_0 \in C[\![Pre_0]\!], \dots, L_{m-1} \in C[\![Pre_{m-1}]\!], M \in M[\![Pps]\!]\,.$

$[\,Pps \equiv Ps[\![Pre_0 \| \dots \| Pre_{m-1}]\!]; Ps' \wedge (\exists j \in m\,.\ \neg(Pre_j \equiv \alpha L_j :)) \wedge$

$(\forall i \in m\,.\ (Pre_i \equiv \alpha L_i :) \vee (Pre_i \equiv \alpha L_i : \sharp(se); \delta \wedge M(se) \in 0))]$

$\Rightarrow \bigwedge_{k \in m} \tilde{I}(L_k)(L_0, \dots, L_{k-1}, L_{k+1}, \dots, L_{m-1}, M)]$

Informellement, pour tout tuple $L_0, \dots, L_{m-1}$ d'étiquettes telles que

- L_i désigne une commande $L_i : \sharp(se)$;
  (auquel cas $\underline{\text{bloqué}}[\![Pps]\!](i, L_i)(M) = [M(se) \leq 0]$)

ou bien

- L_i désigne le point de sortie du processus Pre_i
  (auquel cas $\underline{\text{bloqué}}[\![Pps]\!](i, L_i)(M) = \mathit{tt}$)

et telles qu'elles ne désignent pas toutes des points de sortie, nous devons montrer que :

$\bigwedge_{i \in m} \underline{\text{bloqué}}[\![Pps]\!](i, L_i)(M) \Rightarrow \bigwedge_{i \in m} \tilde{I}(L_i)(L_0, \dots, L_{i-1}, L_{i+1}, \dots, L_{m-1}, M)$

Exemple 4.3.8.2.3-1

Considérons le programme très simple suivant :

```
0: [[
       11: while true do 12: ♯(m) ; 13: ♮(m); 14: od ; 15:
    ||
       21: while true do 22: ♯(m) ; 23: ♮(m); 24: od ; 25:
   ]];
3:
```

Les conditions de vérification sont :

- Initialisation

  $\phi(m) \Rightarrow \neg P_0(m)$

- Induction

  . $[P_{11}(21, m) \wedge P_{21}(11, m)] \Rightarrow P_0(m)$

  . Preuve séquentielle du processus 1 :

  $P_{12}(c_1, m) \Rightarrow [P_{11}(c_1, m) \wedge P_{14}(c_1, m)]$

  $[m > 0 \wedge P_{13}(c_1, m-1)] \Rightarrow P_{12}(c_1, m)$

  $P_{14}(c_1, m+1) \Rightarrow P_{13}(c_1, m)$

  . Absence d'interférences du processus 1 avec la preuve du processus 2 :

  Pour $j = 1, \dots, 5$

  $[P_{12}(2_j, m) \wedge P_{2j}(12, m)] \Rightarrow [P_{2j}(11, m) \wedge P_{2j}(14, m)]$

  $[m > 0 \wedge P_{13}(2_j, m-1) \wedge P_{2j}(13, m-1)] \Rightarrow P_{2j}(12, m)$

  $[P_{14}(2_j, m+1) \wedge P_{2j}(14, m+1)] \Rightarrow P_{2j}(13, m)$

  . La preuve séquentielle du processus 2 et la preuve d'absence d'interférences du processus 2 avec la preuve du processus 1 est similaire.

  . $P_3(m) \Rightarrow [P_{15}(25, m) \wedge P_{25}(15, m)]$

- Finalisation

  . Interblocage possible en 13 et 23 :

  $m \leq 0 \Rightarrow [P_{12}(23, m) \wedge P_{23}(13, m)]$

. Interblocage possible en 12 et 25 :

$m \leq 0 \Rightarrow [P_{12}(25, m) \wedge P_{25}(12, m)]$

. Interblocage possible en 15 et 22 :

$m \leq 0 \Rightarrow [P_{15}(22, m) \wedge P_{22}(15, m)]$

L'esquisse de la preuve est :

```
0: {m<1}
[[      11: {[m≤0 ∧ c2∈{21,22,24}]∨[m<0 ∧ c2=23]∨[c2=25]}
        while true do
        12:   {[m≤0 ∧ c2∈{21,22,24}]∨[m<0 ∧ c2=23]∨[c2=25]}
              P(m);
        13:   {[m<0 ∧ c2∈{21,22,24}]∨[m<-1 ∧ c2=23]∨[c2=25]}
              V(m);
        14:   {[m≤0 ∧ c2∈{21,22,24}]∨[m<0 ∧ c2=23]∨[c2=25]}
        od;
        15: {tt}
||      21: {[m≤0 ∧ c1∈{11,12,14}]∨[m<0 ∧ c1=13]∨[c1=15]}
        while true do
        22:   {[m≤0 ∧ c1∈{11,12,14}]∨[m<0 ∧ c1=13]∨[c1=15]}
              P(m);
        23:   {[m<0 ∧ c1∈{11,12,14}]∨[m<-1 ∧ c1=13]∨[c1=15]}
              V(m);
        24:   {[m≤0 ∧ c1∈{11,12,14}]∨[m<0 ∧ c1=13]∨[c1=15]}
        od;
        25: {tt}
]];
3: {tt}
```

Informellement, en chaque point du programme nous donnons une condition sur m qui est nécessaire (mais peut-être pas suffisante) pour que le programme soit bloqué plus tard. Puisque cette condition n'est pas satisfaite par les états d'entrée quand $m \geq 1$, le programme ne peut pas être bloqué.

□

2.3.2.2.4 Construction d'une méthode de preuve d'exclusion mutuelle dans les programmes parallèles asynchrones par induction en arrière

Deux sections d'un programme sont en exclusion mutuelle si elles ne contiennent pas de commandes qui peuvent s'exécuter en même temps. Supposons que cs_i (respectivement cs_j) est un prédicat qui caractérise l'ensemble des étiquettes appartenant à la section critique du processus Pro_i (respectivement Pro_j) du programme :

$Pps \equiv Ps[\![Pro_0 \| \ldots \| Pro_{m-1}]\!]; Ps'$

Les sections critiques sont en exclusion mutuelle si et seulement si :

$\forall p \in \Sigma^{<S[\![Pps]\!], A[\![Pps]\!], t[\![Pps]\!], \varepsilon>}, k \in |p| .\ \underline{me}[\![Pps]\!](i,j)(cs_i, cs_j)(p_k)$

où

$\underline{me}[\![Pps]\!](i,j)(cs_i, cs_j)(\bar{s}) = [\forall L_0 \in C[\![Pro_0]\!], \ldots, L_{m-1} \in C[\![Pro_{m-1}]\!], M \in M[\![Pps]\!].$

$\bar{s} = \langle\langle L_0, \ldots, L_{m-1}\rangle, M\rangle \Rightarrow \neg(cs_i(L_i) \wedge cs_j(L_j))]$

L'exclusion mutuelle est une propriété d'invariance qui peut être démontrée en utilisant le principe d'induction contrapositif en arrière ($\tilde{\forall}$). Les conditions de vérification sont alors similaires à celles pour l'absence d'interblocages, excepté pour la finalisation qui est :

- Finalisation

$[\neg \underline{me}[\![Pps]\!](i,j)(cs_i, cs_j)(\bar{s}) \Rightarrow \gamma[\![Pps]\!](\tilde{I})(\bar{s})]$

$= [(cs_i(L_i) \wedge cs_j(L_j)) \Rightarrow \bigwedge_{k \in m} \tilde{I}(L_k)(L_0, \ldots, L_{k-1}, L_{k+1}, \ldots, L_{m-1}, M)]$

Informellement, pour toutes les étiquettes L_i de Pro_i telles que $cs_i(L_i)$ et L_j de Pro_j telles que $cs_j(L_j)$,

$\tilde{I}(L_i)(c_0, \ldots, c_{j-1}, L_j, c_{j+1}, \ldots, c_{i-1}, c_{i+1}, \ldots, c_{m-1}, M)$

et

$\tilde{I}(L_j)(c_0, \ldots, c_{j-1}, c_{j+1}, \ldots, c_{i-1}, L_i, c_{i+1}, \ldots, c_{m-1}, M)$

doivent être vrais. De plus, pour toutes les étiquettes L_k du processus k, $k \in (m \setminus \{i,j\})$ nous devons avoir :

$$[cs_i(c_i) \wedge cs_j(c_j)] \Rightarrow \check{I}(L_k)(c_0, \ldots, c_{k-1}, c_{k+1}, \ldots, c_{n-1}, M)$$

Exemple 4.3.2.2.4-1

Les points 13 et 23 du programme 4.3.2.2.3-1 sont en exclusion mutuelle quand $m \leq 1$. Les conditions de vérification sont celles de 4.3.2.2.3-1 excepté pour la finalisation qui est :

$P_{13}(23, m)$

$P_{23}(13, m)$

L'esquisse de la preuve est :

```
0: {m>1}
[[
        11: {[m>0 ∧ c2=23]∨[m>1 ∧ c2∈{21,22,24}]}
            while true do
        12:     {[m>0 ∧ c2=23]∨[m>1 ∧ c2∈{21,22,24}]}
                ♯(m);
        13:     {[c2=23]∨[m>0 ∧ c2∈{21,22,24}]}
                ♮(m);
        14:     {[m>0 ∧ c2=23]∨[m>1 ∧ c2∈{21,22,24}]}
            od;
        15: {ff}
||
        21: {[m>0 ∧ c1=13]∨[m>1 ∧ c1∈{11,12,14}]}
            while true do
        22:     {[m>0 ∧ c1=13]∨[m>1 ∧ c1∈{11,12,14}]}
                ♯(m);
        23:     {[c1=13]∨[m>0 ∧ c1∈{11,12,14}]}
                ♮(m);
        24:     {[m>0 ∧ c1=13]∨[m>1 ∧ c1∈{11,12,14}]}
            od;
        25: {ff}
]];
3: {tt}
```

□

4.3.2.2.5 Construction d'une méthode de preuve de non-terminaison de programmes parallèles par induction en arrière

Un programme Pps ne se termine pas si et seulement si

$$\forall p \in \Sigma^{<S[\![Pps]\!]}, A[\![Pps]\!], t[\![Pps]\!], \epsilon>, i \in |p|.\ \psi(p_i)$$

où

$$\psi(\bar{\lambda}) = \neg [\exists \bar{L} \in C[\![Pps]\!], \bar{M} \in M[\![Pps]\!].\ \bar{\lambda} = \langle \bar{L}, \bar{M} \rangle \wedge Pps \equiv \ll \bar{L} :]$$

C'est une propriété que nous pouvons montrer par induction en arrière en utilisant $(\check{I})$. Les conditions de vérification sont celles du paragraphe 4.3.2.2.4 excepté pour la finalisation qui est :

$$\forall \bar{M} \in M[\![Pps]\!].\ \check{I}(\bar{L})(\bar{M}) \quad \text{où } Pps \equiv \ll \bar{L} :$$

Exemple 4.3.2.2.5-1

Les conditions de vérification sont celles de l'exemple 4.3.2.2.4-1 excepté pour la finalisation qui est :

$P_3(m)$

Esquisse de la preuve :

```
0: {ff}
[[
    ||  11: {ff} while true do 12: {ff} ♯(m); 13: {ff} ♮(m); 14: {ff} od; 15: {tt}
    ||  21: {ff} while true do 22: {ff} ♯(m); 23: {ff} ♮(m); 24: {ff} od; 25: {tt}
]];
3: {tt}
```

4.3.2.2.6 Conclusion sur la preuve de propriétés d'invariance de programmes par induction en arrière

La méthode de Morris-Wegbreit [77] dite "subgoal induction" n'a jamais remporté le succès qu'a eu la méthode de Floyd [67]. Diverses raisons ont été avancées dont certaines sont incorrectes (comme l'incomplétude sémantique Misra [78], cf. 4.3.2.2.1.3) ou superficielles (comme les limitations concernant les programmes qui ne se terminent pas Morris-Wegbreit [77], cf. 4.3.2.2.1.5). Dijkstra [88, p.224-225] démontre un cas particulier du théorème 4.2.1.4 N 2 à propos d'un programme séquentiel consistant en une boucle "while" et conclut (à la page xiii) que "we can ignore subgoal-induction because it is nothing but the Invariance Theorem in a complicated disguise". Cette conclusion porte sur l'équivalence des méthodes de preuve et reste donc valable pour la méthode de preuve de correction partielle que nous avons introduite (en généralisant la méthode de Morris-Wegbreit dite "subgoal induction") quand on la compare par exemple aux méthodes de preuve d'invariance en avant à la Lamport [77], Owicki-Gries [76a] (qui généralisent la méthode de Floyd [67]). Cependant comme nous l'avons remarqué en 4.3.2.2.2.5-3, l'invariant A associé en tout point d'un programme pour une méthode de preuve en avant peut être utilisé pour une méthode de preuve en arrière dans la forme A⇒R. quand la preuve est utilisée comme commentaires, ceci est utile pour le lecteur du programme puisque A décrit ce qui a été fait jusqu'à ce point et R ce qui reste à faire. R doit être inventé par le lecteur du programme quand les méthodes de preuve en avant sont utilisées.

Ces arguments nous semblent en fait peu convaincants. La véritable raison de l'insuccès de l'induction en arrière nous semble être que pour les méthodes de preuve en avant, les mêmes invariants peuvent être utilisés pour la correction partielle, l'absence d'interblocage

l'exclusion mutuelle, l'absence d'erreurs à l'exécution, la non-terminaison, etc. ceci parce que dans l'induction en avant, le même principe d'induction (-) peut être utilisé pour la preuve de toutes ces propriétés d'invariance. Ce qui n'est pas le cas pour les méthodes de preuve en arrière pour lesquelles nous devons utiliser deux principes d'induction ($-\downarrow^{-1}$) et ($-\bar{\iota}$) qui conduisent à des conditions de vérifications différentes.

Un compromis heureux pourrait consister à utiliser une combinaison des principes d'induction en avant et en arrière comme nous l'avons proposé en 4.3.2.2.2.5-3. Pour les programmes parallèles, par exemple, il faut souvent démontrer une propriété de la forme $\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|.\ (\sigma(p_i) \Rightarrow \psi(p_0,p_i))$ (comme la correction partielle) et une propriété de la forme $\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|.\ \Gamma(p_i)$ (où Γ est une conjonction de conditions correspondant par exemple à l'absence d'interblocages globaux permanents, l'exclusion mutuelle de certaines sections critiques, etc.). On pourra alors choisir le principe d'induction suivant :

$$
\begin{aligned}
&[\exists K \in (S^3 \to \{tt, ff, \bot\}).\ \forall \underline{s},s,s',\bar{s} \in S,\ a \in A \\
&\quad [\varepsilon(\underline{s}) \wedge \sigma(\bar{s})] \Rightarrow [K(\underline{s},\underline{s},\bar{s}) \wedge K(\underline{s},\bar{s},\bar{s})] \\
&\quad \wedge\ [\varepsilon(\underline{s}) \wedge K(\underline{s},s,\bar{s}) \wedge t_a(s,s') \wedge \sigma(\bar{s})] \Leftrightarrow \\
&\qquad [\varepsilon(\underline{s}) \wedge t_a(s,s') \wedge K(\underline{s},s',\bar{s}) \wedge \sigma(\bar{s})] \\
&\quad \wedge\ [\varepsilon(\underline{s}) \wedge (K(\underline{s},\underline{s},\bar{s}) \vee K(\underline{s},\bar{s},\bar{s})) \wedge \sigma(\bar{s})] \Rightarrow \psi(\underline{s},\bar{s}) \\
&\quad \wedge\ [\varepsilon(\underline{s}) \wedge K(\underline{s},s,\bar{s})] \Rightarrow \Gamma(s)\ ] \\
\Leftrightarrow\ &[\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|.\ [\forall j \in i.\neg\sigma(p_j) \wedge \sigma(p_i)] \Rightarrow [\forall j \in i.\Gamma(p_j) \wedge \psi(p_0,p_i)]]
\end{aligned}
$$

(La preuve de correction consiste essentiellement à démontrer que $\forall i \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|.\ [\forall j \in i.\ \neg\sigma(p_i) \wedge \sigma(p_i)] \Rightarrow [\forall i \in i.\ K(p_0,\ldots,\ldots)]$ et

la preuve de complétude se fait en choisissant $K(\Lambda,\Delta,\bar{\Lambda}) =$
$[\exists p \in \Sigma^{<S,A,t,\varepsilon>}, i \in |p|, j \leq i.\ p_0 = \underline{\Lambda} \wedge p_j = \Delta \wedge p_i = \bar{\Lambda} \wedge \forall j \in i.\ \neg\sigma(p_j) \wedge \sigma(p_i)]$.

Ce principe d'induction est implicitement utilisé dans les preuves informelles que donnent Ricart-Agrawala [81].

4.3.2.3 Construction d'une méthode de preuve pour les programmes parallèles communicants

En appliquant la méthode de construction définie au paragraphe 4.3.1, nous avons proposé dans Cousot-Cousot [80a] une méthode de preuve pour un sous-ensemble de CSP dont l'originalité consistait à proposer une décomposition des preuves pour les programmes CSP en des preuves pour chaque processus (ne faisant référence qu'à l'état du processus), pour chaque canal (faisant référence à l'état de tous les processus au moment des communications (la référence aux seuls états des processus qui communiquent sur le canal étant incomplète)) et en des preuves d'absence d'interférence (entre les assertions associées aux canaux et l'exécution des processus). L'idée essentielle était de considérer qu'entre l'initialisation et l'attente du premier rendez-vous, ou entre deux attentes de rendez-vous ou entre l'attente du dernier rendez-vous et la terminaison, l'exécution d'un processus est indivisible. Cette idée n'est plus valable pour les programmes communicants considérés au paragraphe 4.8.3 pour la raison que les processus peuvent, entre deux communications par les canaux, interférer au moyen des variables globales partagées.

Soit

$$Ppc \equiv Ps \llbracket Prc_0 \parallel \ldots \parallel Prc_{n-1} \rrbracket ; Ps'$$

Pour construire une méthode de preuve basée sur le principe d'induction (Y), nous pouvons utiliser la décomposition définie par :

$$A_S \llbracket Ppc \rrbracket = (S \llbracket Ppc \rrbracket^{\varsigma} \to \{tt, \underline{ff}\})$$
$$\tilde{r}_S \llbracket Ppc \rrbracket = (\tilde{Prel} \llbracket Ppc \rrbracket \cup \tilde{Proc} \llbracket Ppc \rrbracket \cup \tilde{Postl} \llbracket Ppc \rrbracket)$$

Une relation reliant les états mémoire courants et initiaux est associée à chaque point du prélude et du postlude :

$\tilde{Prél}[\![Ppc]\!] = \{\tilde{I} : \exists \underline{L} \in C[\![Ps]\!] . \tilde{I}(\underline{L}) \in (M[\![Ppc]\!]^2 \to \{tt, ff\})\}$

$\tilde{Postl}[\![Ppc]\!] = \{\tilde{I} : \exists \underline{L} \in C[\![Ps']\!] . \tilde{I}(\underline{L}) \in (M[\![Ppc]\!]^2 \to \{tt, ff\})\}$

Une relation reliant l'état mémoire et contrôle courant à l'état mémoire initial est associée à chaque point de chaque processus :

$\tilde{Proc}[\![Ppc]\!] = \{\tilde{I} : \exists i<m, L \in C[\![Prc_i]\!] . \tilde{I}(i)(L) \in (M[\![Ppc]\!] \times (\prod_{j\in(m\setminus i)} C[\![Prc_j]\!]) \times M[\![Ppc]\!] \to \{tt, ff\})\}$

La signification d'un tel vecteur $\tilde{I}$ est définie formellement par la fonction sémantique $\gamma[\![Ppc]\!](\tilde{I}) = I$, où par cas :

$I(\langle \underline{L}, \underline{M} \rangle, \langle L, M \rangle) = \tilde{I}(L)(\underline{M}, M)$ quand $Ppc \equiv \underline{L} : \beta \wedge L \in (C[\![Ps]\!] \cup C[\![Ps']\!])$

$I(\langle \underline{L}, \underline{M} \rangle, \langle\langle L_0, \ldots, L_{m-1} \rangle, M \rangle) =$

$\bigwedge_{i\in m} \tilde{I}(L_i)(\underline{M}, L_0, \ldots, L_{i-1}, L_{i+1}, \ldots, L_{m-1}, M)$ quand $Ppc \equiv \underline{L} : \beta$

Nous ne donnerons pas le détail de la construction des conditions de vérification correspondant à cette décomposition, ni la vérification de la complétude sémantique. En résumé, les conditions de vérification sont les suivantes (si on exclut les commandes de communication, ce sont celles proposées par Lamport [76a] et sont similaires aux conditions de Owicki-Gries [77] (sauf pour l'usage de variables auxiliaires qui est remplacé par celui des compteurs ordinaux). Avec les commandes de communication elles sont similaires à celles proposées par Levin [77] (les variables auxiliaires étant remplacées par des compteurs ordinaux)) :

Les invariants $P_i(\underline{x}, x)$ associés aux points L_i des prélude et postlude reliant l'état initial $\underline{x}$ à l'état courant x des variables quand l'exécution est en L_i

Les invariants $P_{ij}(\underline{x}, c_0, \ldots, c_{i-1}, c_{i+1}, \ldots, c_{m-1}, x)$ associés aux points L_j

courant x des variables et l'état $c_0, \ldots, c_{i-1}, c_{i+1}, \ldots, c_{m-1}$ de contrôle des autres processus $Prc_0, \ldots, Prc_{i-1}, Prc_{i+1}, \ldots, Prc_{m-1}$ quand l'exécution est au point L_j du processus Prc_i.

- Preuve séquentielle

(pour le prélude Ps, le postlude Ps' et chaque processus $Prc_0, \ldots, Prc_{m-1}$)

. Initialisation (prélude)

$L_1 : \alpha$ $\qquad\qquad$ $\forall \tilde{x} . P_1(\tilde{x}, \tilde{x})$

. Commande nulle

α L_1 : skip ; $L_2 : \beta$ $\qquad\qquad$ $P_1 \Rightarrow P_2$

. Commande d'affectation

α L_1 : $V := E$; $L_2 : \beta$ $\qquad\qquad$ $P_1 \Rightarrow P_2[V \leftarrow E]$

α L_1 : $V := ?$; $L_2 : \beta$ $\qquad\qquad$ $\forall v \in \mathcal{D} . P_1 \Rightarrow P_2[V \leftarrow v]$

. Commande conditionnelle

α L_1 : if B then

L_2 : β

L_3 : $\qquad\qquad$ $[P_1 \wedge B] \Rightarrow P_2$

else

L_4 : γ $\qquad\qquad$ $[P_1 \wedge \neg B] \Rightarrow P_4$

L_5 : $\qquad\qquad$ $[P_3 \vee P_5] \Rightarrow P_6$

fi ;

$L_6 : \delta$

. Itération

χ L_1 : while B do

L_2 :

L_3 : $\qquad\qquad$ $(P_1 \vee P_3) \Rightarrow [(B \Rightarrow P_2) \wedge (\neg B \Rightarrow P_4)]$

od ;

$L_4 : \delta$

. Section critique (dans le processus $Proc_i$)

$\alpha\, L_{i_1}$: ✗
L_{i_2} : β
L_{i_3} :
✗ ;
L_{i_4} : γ

A chaque point L_{ij} du corps $L_{i_2}: \beta\, L_{i_3}:$ de la section critique, une relation invariante $P_{ij}(x', x)$ relie les valeurs courantes x en L_{ij} aux valeurs initiales x' en L_{i_2} des variables dans la section critique. Les conditions de vérification correspondantes sont celles des programmes séquentiels avec $\forall x'. P_{i_2}(x', x)$. Il faut y ajouter

$$[P_{i_1}[x \leftarrow x'] \wedge P_{i_3}(x', x)] \Rightarrow P_{i_4}$$

. Sortie d'une commande alternative (dans le processus $Proc_i$)

α as P_0; L_{i_0} : or ... or $\beta_{\ell-1}$; $L_{i_{\ell-1}}$: ei; L_{i_ℓ} : β

$\forall j \in \ell.\ P_{i_j} \Rightarrow P_{i_\ell}$

- Initialisation / Finalisation du parallélisme

. $\alpha\, L_n : [\![L_0 : \beta_0 \,\|\, \dots \,\|\, L_{m-1} : \beta_{m-1}]\!]\ \beta$

$$P_m(\underline{x}, x) \Rightarrow [\ \underset{i \in m}{\wedge} P_i(\underline{x}, L_{0,\dots}, L_{i-1}, L_{i+1}, \dots, L_{m-1}, x)\ ]$$

. $\alpha\, [\![\beta_0\, L_0 : \| \dots \| \beta_{m-1}\, L_{m-1} :]\!]\ ;\ L_n : \beta$

$$[\ \underset{i \in m}{\wedge} P_i(\underline{x}, L_{0,\dots}, L_{i-1}, L_{i+1}, \dots, L_{m-1}, x)\ ] \Rightarrow P_m(\underline{x}, x)$$

- Preuve de correction des communications

. ch!E équivalent à as true ; ch!E then skip ; es se traite comme ci-dessous

. ch?V équivalent à as true ; ch?V then skip ; es se traite comme ci-dessous

. Pour toutes paires de commandes alternatives dans des processus différents i et j,

$\alpha\, L_{i_1}$: as ... or B_1 ; ch!E then $L_{i_2}: \beta$ or ... es γ

$\alpha'\, L_{j_1}$: as ... or B_2 ; ch?V then $L_{j_2}: \beta'$ or ... es γ'

$$(P_{i_2}[c_j \leftarrow L_{j_1}] \wedge B_1 \wedge P_{j_1}[c_i \leftarrow L_{i_2}] \wedge B_2)$$
$$\Rightarrow (\ P_{i_2}[c_j \leftarrow L_{j_2}, V \leftarrow E] \ \wedge\ P_{j_2}[c_i \leftarrow L_{i_2}, V \leftarrow E])$$

- Preuve d'absence d'interférences

Pour tout point L_{k_ℓ} de tout processus $Proc_k$, $k \in (m \setminus i)$,

. Commande nulle

$\alpha\, L_{i_1}$: skip ; $L_{i_2}: \beta$

$$(P_{i_1}[c_k \leftarrow L_{k_\ell}] \wedge P_{k_\ell}[c_i \leftarrow L_{i_2}]) \rightarrow P_{k_\ell}[c_i \leftarrow L_{i_2}]$$

. Commande d'affectation

$\alpha\, L_{i_1}$: $V := E$; $L_{i_2}: \beta$

$$(P_{i_1}[c_k \leftarrow L_{k_\ell}] \wedge P_{k_\ell}[c_i \leftarrow L_{i_2}]) \Rightarrow P_{k_\ell}[c_i \leftarrow L_{i_2}, V \leftarrow E]$$

$\alpha\, L_{i_1}$: $V := ?$; $L_{i_2}: \beta$

$$(P_{i_1}[c_k \leftarrow L_{k_\ell}] \wedge P_{k_\ell}[c_i \leftarrow L_{i_2}]) \Rightarrow (\forall v \in \mathbb{D}.\ P_{k_\ell}[c_i \leftarrow L_{i_2}, V \leftarrow v])$$

. Commande conditionnelle

αL_{i_1} : **if** B **then**

L_{i_2} :
β
L_{i_3} :

else

L_{i_4} : γ
L_{i_5} :

fi;

L_{i_6} : δ

$(P_{i_2}[c_k \leftarrow L_{k_\ell}] \wedge P_{k_\ell}[c_i \leftarrow L_{i_2}]) \Rightarrow [(B \Rightarrow P_{k_\ell}[c_i \leftarrow L_{i_2}]) \wedge (\neg B \Rightarrow P_{k_\ell}[c_i \leftarrow L_{i_4}])]$

$[(P_{i_3}[c_k \leftarrow L_{k_\ell}] \wedge P_{k_\ell}[c_i \leftarrow L_{i_3}]) \vee (P_{i_5}[c_k \leftarrow L_{k_\ell}] \wedge P_{k_\ell}[c_i \leftarrow L_{i_5}])] \Rightarrow P_{k_\ell}[c_i \leftarrow L_{i_6}]$

. Itération

αL_{i_1} : **while** B **do**

L_{i_2} :
β
L_{i_3} :

od;

L_{i_4} : γ

$[(P_{i_2}[c_k \leftarrow L_{k_\ell}] \wedge P_{k_\ell}[c_i \leftarrow L_{i_2}]) \vee (P_{i_3}[c_k \leftarrow L_{k_\ell}] \wedge P_{k_\ell}[c_i \leftarrow L_{i_3}])]$
$\Rightarrow [(B \Rightarrow P_{k_\ell}[c_i \leftarrow L_{i_2}]) \wedge (\neg B \Rightarrow P_{k_\ell}[c_i \leftarrow L_{i_4}])]$

. Section critique

αL_{i_1} : ☆

L_{i_2} :
β
L_{i_3} :

☆;
L_{i_4} : γ

$[P_{i_2}[c_k \leftarrow L_{k_\ell}, x \leftarrow x'] \wedge P_{k_\ell}[c_i \leftarrow L_{i_2}, x \leftarrow x'] \wedge P_{i_3}(x', x)] \Rightarrow P_{k_\ell}[c_i \leftarrow L_{i_4}]$

. Communication $(i \neq k,\ j \neq k)$

αL_{i_1} : **ae** … **or** B_1 ; ch!E **then** L_{i_2} : β **or** … **es** γ

$\alpha' L_{j_1}$: **ae** … **or** B_2 ; ch?v **then** L_{j_2} : β' **or** … **es** γ'

$[P_{i_1}[c_j \leftarrow L_{j_2}, c_k \leftarrow L_{k_\ell}] \wedge B_1 \wedge P_{j_2}[c_i \leftarrow L_{i_2}, c_k \leftarrow L_{k_\ell}] \wedge B_2 \wedge P_{k_\ell}[c_i \leftarrow L_{i_2}, c_j \leftarrow L_{j_2}]]$
$\Rightarrow P_{k_\ell}[c_i \leftarrow L_{i_2}, c_j \leftarrow L_{j_2}, v \leftarrow E]$

4.3.2.4 Comparaison des méthodes de preuve pour les programmes parallèles connues dans la littérature

La comparaison des méthodes de preuve de propriétés d'invariance de programmes parallèles (comme Ashcroft [77], Hoare [75], Howard [76], Keller [76], Lamport [77], Mazurkiewicz [77], Newton [75], Owicki-Gries [76a, 76b], etc.) est souvent difficile à cause de la diversité des formalismes syntaxiques qui sont utilisés pour représenter les algorithmes. Nous proposons de les comparer en montrant que toutes ces méthodes dérivent du même principe d'induction (-I-) (ou ses variantes (-I'-), (-i-), (i) ou leurs transformés par ν) et ne diffèrent que par la façon de décomposer l'invariant global utilisé dans (-I-) en invariants locaux associés à des points du programme.

4.3.2.4.1 Utilisation d'un seul invariant global

Dans les méthodes de preuve d'invariance de Ashcroft [75] et Keller [76], un seul invariant est utilisé dans la preuve. Ashcroft justifie sa méthode de preuve par un théorème qui peut s'exprimer en utilisant notre formalisme comme suit :

$$[\forall \Delta, \Delta' \in S.\ \varepsilon(\Delta) \Rightarrow \psi(\Delta) \wedge [\exists \underline{\Delta} \in S.\ \varepsilon(\underline{\Delta}) \wedge t^*(\underline{\Delta}, \Delta) \wedge \psi(\Delta) \wedge (\exists a \in A.\ t_a(\Delta, \Delta'))] \Rightarrow \psi(\Delta')]$$

$$\Leftrightarrow$$

$$[\forall \Delta \in S.\ (\exists \underline{\Delta} \in S.\ \varepsilon(\underline{\Delta}) \wedge t^*(\underline{\Delta}, \Delta)) \Rightarrow \psi(\Delta)]$$

Puis Ashcroft remarque ensuite qu'on ne connaît pas exactement l'ensemble des états Δ satisfaisant $[\exists \underline{\Delta} \in S.\ \varepsilon(\underline{\Delta}) \wedge t^*(\underline{\Delta}, \Delta)]$ de sorte que "we could have left this term out of the verification condition entirely

if ajoute "however, if the impossibility of reaching certain states is crucial for certain properties of a program to hold" then we can explicitely incorporate the impossibility into the assertions we wish to prove valid and check the above conditions for all states". Autrement dit, on peut être amené à remplacer ψ par I tel que $\forall \Delta \in S.\ I(\Delta) \Rightarrow \psi(\Delta)$. Il s'agit donc bien du principe d'induction (-i-) dont la correction a été démontrée ultérieurement par Keller [76].

Dans les deux cas, on choisit l'ensemble des actions comme étant l'ensemble des affectations et tests de chaque processus du programme, de sorte que la méthode consiste à utiliser un seul invariant global et autant de conditions de vérification qu'il y a d'actions. Dans ce cas la méthode de preuve consiste à appliquer directement le principe d'induction. Cependant l'utilisation d'un seul invariant pour décrire le comportement d'un grand programme peut être inadéquate.

Exemple 4.3.2.4.1-1

Si on veut démontrer la correction partielle du programme $[1_1 : x := x+1 ;\ 1_2 :\ \| \ 2_1 : x := x+2 ;\ 2_2 : \|]$ relativement à une spécification ϕ, ψ, il faudra trouver un invariant I tel que :

$$\phi(x) \Rightarrow I(1_1, 2_1, x)$$

$$[I(1_1, c_2, x') \wedge x = x'+1] \Rightarrow I(1_2, c_2, x)$$

$$[I(c_1, 2_1, x') \wedge x = x'+2] \Rightarrow I(c_1, 2_2, x)$$

$$I(1_2, 2_2, x) \Rightarrow \psi(x)$$

□

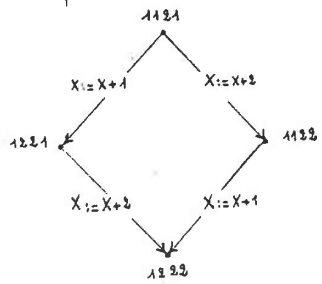
4.3.2.4.2 Utilisation d'invariants sur les variables associés à chaque état de contrôle

Une des premières tentatives de décomposition de l'invariant global a été faite par Ashcroft-Manna [70]. Leur technique consiste à transformer un programme parallèle en un programme non déterministe équivalent puis à appliquer la méthode de Floyd-Hoare-Naur qui utilise un invariant local, portant sur les variables du programme, associé à chaque point du programme transformé.

Exemple 4.3.2.4.2-1

$[\![11 : X := X+1 ; 12 : \| 21 : X := X+2 ; 22 :]\!]$

est transformé en :



auquel on applique la méthode de Floyd [67], ce qui donne :

$\phi(x) \Rightarrow I_{1121}(x)$

$[I_{1121}(x') \wedge x = x'+1] \Rightarrow I_{1221}(x)$

$[I_{1121}(x') \wedge x = x'+2] \Rightarrow I_{1122}(x)$

$[I_{1221}(x') \wedge x = x'+2] \Rightarrow I_{1222}(x)$

$[I_{1122}(x') \wedge x = x'+1] \Rightarrow I_{1222}(x)$

$I_{1222}(x) \Rightarrow \psi(x)$

□

Une autre façon d'expliquer la même idée qui évite d'avoir à transformer le programme consiste à dire que la méthode de Floyd [67] et Ashcroft-Manna [70] s'applique à des programmes dont l'ensemble des états est $S = C \times M$ où C est l'ensemble des états de contrôle et M l'ensemble des états mémoires et consiste à appliquer le principe d'induction (-ii) avec la décomposition :

$A_S = (C \times M \to \{tt, ff\})$

$\tilde{A_S} = (C \to (M \to \{tt, ff\}))$

$\alpha \in (A_S \to \tilde{A_S})$

$\alpha(I)_c(m) = I(\langle c, m \rangle)$

$\gamma \in (\tilde{A_S} \to A_S)$

$\gamma(\tilde{I})(\langle c, m \rangle) = \tilde{I}_c(m)$

ce qui conduit évidemment à des méthodes correctes et sémantiquement complètes (cf. 4.3.1.7-1 et 4.3.1.8-2, les hypothèses de 4.3.1.C.2.7 étant satisfaites).

Si cette décomposition est appliquée à un programme parallèle $[\![Pr_0 \| \ldots \| Pr_{m-1}]\!]$ où chaque processus Pr_i a m_i points de contrôle, il y a $\tau_i \times m_1 \times \ldots \times m_{m-1}$ invariants locaux à considérer, ce qui conduit évidemment à des preuves très longues. Ceci peut être évité en utilisant des décompositions moins fines.

4.3.2.4.3 Utilisation d'invariants sur les variables associés à chaque point de contrôle du programme

Considérons un programme dont l'ensemble des états est $S = C_0 \times \ldots \times C_{m-1} \times M$ où C_i, $i \in m$ est l'ensemble des points de contrôle du $i$ème processus et M l'état mémoire.

On peut remplacer l'invariant global utilisé dans le principe d'induction (-i) par des invariants locaux portant sur l'état mémoire et associés à chaque point de contrôle. Ceci revient à choisir (on suppose $i \neq j \Rightarrow (C_i \cap C_j = \phi)$) :

$$A_S = (C_0 \times \ldots \times C_{m-1} \times M \to \{tt, ff\})$$

$$\tilde{A}_S = \tilde{A}_{S_0} \times \ldots \times \tilde{A}_{S_{m-1}} \quad \text{où} \quad \tilde{A}_{S_i} = (C_i \to (M \to \{tt, ff\}))$$

$$\alpha \in (A_S \to \tilde{A}_S)$$

$$\alpha(I)_{i,c}(m) = (\exists c_0 \in C_0, \ldots, c_{i-1} \in C_{i-1}, c_{i+1} \in C_{i+1}, \ldots, c_{m-1} \in C_{m-1} \cdot I(\langle\langle c_0, \ldots, c_{i-1}, c, c_{i+1}, \ldots, c_{m-1}\rangle, m\rangle))$$

$$\gamma \in (\tilde{A}_S \to A_S)$$

$$\gamma(\tilde{I})(\langle\langle c_0, \ldots, c_{m-1}\rangle, m\rangle) = [\forall i \in m \cdot \tilde{I}_{i,c_i}(m)]$$

Exemple 4.3.2.4.3-1 Méthode de Owicki-Gries [76a]

Pour démontrer la correction partielle de :

$[\![11: x := x+1 ; 12: \| 21: x := x+2 ; 22:]\!]$

par la méthode de Owicki-Gries [76a], il faut vérifier les conditions suivantes :

$$\phi \Rightarrow [I_{11}(x) \wedge I_{21}(x)]$$

$$[I_{11}(x') \wedge x = x'+1] \Rightarrow I_{12}(x)$$

$$[I_{21}(x') \wedge x = x'+2] \Rightarrow I_{22}(x)$$

$$[I_{11}(x') \wedge I_{21}(x') \wedge x = x'+2] \Rightarrow I_{11}(x)$$

$$[I_{12}(x') \wedge I_{21}(x') \wedge x = x'+2] \Rightarrow I_{12}(x)$$

$$[I_{21}(x') \wedge I_{11}(x') \wedge x = x'+1] \Rightarrow I_{21}(x)$$

$$[I_{22}(x') \wedge I_{11}(x') \wedge x = x'+1] \Rightarrow I_{22}(x)$$

$$[I_{12}(x) \wedge I_{22}(x)] \Rightarrow \psi(x)$$

(Ces conditions s'obtiennent à partir du principe d'induction (-i) en utilisant 4.3.1.7-1, ce point sera illustré au paragraphe suivant).

□

Keller [76] et Owicki-Gries [76a] ont montré que la méthode de preuve correspondante n'est pas complète en utilisant un argument intuitif basé sur un exemple. Ce raisonnement peut être fait plus rigoureusement en observant que partant du principe d'induction (-i) qui s'écrit :

$$[\exists I \in A_S \cdot (\phi(I) \Rightarrow I) \wedge (I \Rightarrow \psi)]$$

où

$$\phi(I)(s) = [\varepsilon(s) \vee (\exists s' \in S \cdot I(s') \wedge t(s', s))]$$

la méthode de preuve consiste à démontrer que

$$[\exists \tilde{I} \in \tilde{A}_S \cdot \alpha \circ \phi \circ \gamma(\tilde{I}) \Rightarrow \tilde{I} \wedge \tilde{I} \Rightarrow \alpha(\psi)]$$

où

$\tilde{I} \Rightarrow \tilde{J}$ si et seulement si $\forall i \in m, c \in C_i, m \in M \cdot \tilde{I}_{i,c}(m) \Rightarrow \tilde{J}_{i,c}(m)$

Posons $\tilde{\phi} = \alpha \circ \phi \circ \gamma$, c'est un opérateur monotone (pour $\Rightarrow$) sur le treillis complet $\tilde{A}_S$. D'après le théorème de Tarski (Tarski [55], Cousot-Cousot [79]) le plus fort invariant $\tilde{I}$ satisfaisant $\tilde{\phi}(\tilde{I}) \Rightarrow \tilde{I}$ est le plus petit point fixe $\mathrm{lfp}(\tilde{\phi})$ de $\tilde{\phi}$ (tel que $\tilde{\phi}(\mathrm{lfp}(\tilde{\phi})) = \mathrm{lfp}(\tilde{\phi})$ et si $\tilde{\phi}(\tilde{I}) \Rightarrow \tilde{I}$ alors $\mathrm{lfp}(\tilde{\phi}) \Rightarrow \tilde{I}$). Si on peut trouver un programme et un ψ tels que $\mathrm{lfp}(\tilde{\phi}) \not\Rightarrow \alpha(\psi)$, alors la méthode est incomplète. Il suffit de choisir le programme

$0: x := 0 ; 1: [\![11: x := x+1 ; 12: \| 21: x := x+1 ; 22:]\!]$

pour lequel $\mathit{fff}(\tilde{\Phi})$ est :

$I_0(x) = \underline{true}$

$I_1(x) = [x=0]$

$I_{11}(x) = [x>0]$ $\quad I_{21}(x) = [x>0]$

$I_{12}(x) = [x>0]$ $\quad I_{22}(x) = [x>0]$

qui ne permet pas de démontrer l'invariance de $\psi(x) = [0 \leq x \leq 2]$.

Remarquons que si $m=1$, la situation est celle de 4.3.2.4.2 et la méthode est sémantiquement complète.
Quand $m>1$, la méthode est incomplète car il n'est pas possible d'exprimer dans $\tilde{As}$ que certains états sont inaccessibles (dans le contre-exemple, on ne peut pas exprimer que lorsque l'exécution est en 11 avec $x=1$ dans le processus 1, alors c'est qu'elle est en 22 dans le processus 2). Pour ce faire, il faut pouvoir exprimer des relations entre les états de contrôle des divers processus. On peut utiliser à cet effet des compteurs ordinaux (comme dans Lamport[77]) ou des variables auxiliaires (comme dans Owicki-Gries[76a, 76b], Levin[79]).

4.3.2.4.4 Utilisation d'invariants sur l'état de contrôle et les variables associés à chaque point de contrôle du programme

Nous pouvons extraire de Newton[75] l'idée exprimée plus clairement dans Lamport[77] qui consiste à associer à chaque point de contrôle du programme une assertion portant sur l'état mémoire et l'état de contrôle du programme.
Nous avons donc :

$S = (C_0 \times \dots \times C_{m-1}) \times M$

$As = (S \to \{tt, ff\})$

$\tilde{As} = \tilde{As}_0 \times \dots \times \tilde{As}_{m-1}$ où $\tilde{As}_i = (C_i \to (C_0 \times \dots \times C_{i-1} \times C_{i+1} \times \dots \times C_{m-1} \times M \to \{tt, ff\}))$

$\alpha \in (As \to \tilde{As})$

$\alpha(I)_{i,c}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{m-1}, m) = I(\langle\langle c_0, \dots, c_{i-1}, c, c_{i+1}, \dots, c_{m-1}\rangle, m\rangle)$

$\gamma \in (\tilde{As} \to As)$

$\gamma(\tilde{I})(\langle\langle c_0, \dots, c_{m-1}\rangle, m\rangle) = [\forall i \in m.\ \tilde{I}_{i,c_i}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{m-1}, m)]$

Comme (α, γ) est une correspondance de Galois injective, nous pouvons construire une méthode de preuve correcte et sémantiquement complète comme en 4.3.1.7-1 et 4.3.1.8-2.

Exemple 4.3.2.4.4-1

Considérons l'application du principe d'induction (-i) aux programmes parallèles asynchrones de la forme :

$Ppa \equiv [\![Pra_0 \| \dots \| Pra_{m-1}]\!]$

(cf. 2.8.2, les prélude et postlude étant vides et les affectations aléatoires et sections critiques étant exclues pour alléger la présentation) en utilisant la décomposition de l'invariant global en invariants locaux définie ci-dessus.

Le principe d'induction (-i) s'écrit :

$[\exists I \in As.\ \varepsilon \Rightarrow I \wedge F(I) \Rightarrow I \wedge I \Rightarrow \psi]$

où

$\varepsilon(\langle\langle c_0, \dots, c_{m-1}\rangle, m\rangle) = [\forall i \in m.\ \exists \underline{L}_i \in \mathcal{L}.\ c_i = \underline{L}_i \wedge Pra_i \equiv \underline{L}_i : \hat{p}]$

$F(I)(\Lambda) = [\exists \Lambda' \in S, i \in m.\ I(\Lambda') \wedge t[\![Ppa]\!]_i(\Lambda', \Lambda)]$

$[\![Ppa]\!]_i(\langle\langle c'_0, \dots, c'_{m-1}\rangle, m'\rangle, \langle\langle c_0, \dots, c_{m-1}\rangle, m\rangle) =$

$[\forall j \in (m \setminus i).\ c'_j = c_j \wedge t[\![Pra_i]\!](\langle c'_i, m'\rangle, \langle c_i, m\rangle)]$

Comme en 4.3.1.7-1, posons $\tilde{F} = \alpha \circ F \circ \gamma$. Nous avons

$$\tilde{F}(\tilde{I})_{i,c_i}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{n-1},m)$$
$$= [\exists \lambda' \in S, k \in m.\ \gamma(\tilde{I})(\lambda') \wedge t[\![Ppa]\!]_k(\lambda', \ll c_0,\dots,c_{n-1}\gg, m))]$$
$$= [\exists c'_0 \in C_0,\dots,c'_{n-1} \in C_{n-1}, m' \in M, k \in m.\ \forall j \in m \setminus k.\ \tilde{I}_{j,c'_j}(c'_0,\dots,c'_{j-1},c'_{j+1},\dots,c'_{n-1},m)$$
$$\wedge\ \forall j \in (m \setminus k).\ c'_j = c_j \ \wedge\ t[\![Pra_k]\!](\langle c'_k, m'\rangle, \langle c_k, m\rangle)]$$
$$= [\exists c'_i \in C_i, m' \in M.\ \tilde{I}_{i,c'_i}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{n-1},m')$$
$$\wedge\ \underline{contexte}\{i\}(\tilde{I})(c_0,\dots,c_{i-1},c'_i,c_{i+1},\dots,c_{n-1},m') \wedge t[\![Pra_i]\!](\langle c'_i,m'\rangle,\langle c_i,m\rangle)]$$
$$\vee\ [\exists k \in (m \setminus i), c'_k \in C_k, m' \in M.\ \tilde{I}_{i,c_i}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{k-1},c'_k,c_{k+1},\dots,c_{n-1},m')$$
$$\wedge\ \tilde{I}_{k,c'_k}(c_0,\dots,c_{k-1},c_{k+1},\dots,c_{n-1},m') \wedge \underline{contexte}\{i,k\}(\tilde{I})(c_0,\dots,c_{k-1},c'_k,c_{k+1},\dots,c_{n-1},m)$$
$$\wedge\ t[\![Pra_k]\!](\langle c'_k,m'\rangle,\langle c_k,m\rangle)]$$

où

$$\underline{contexte}\ E(\tilde{I})(c_0,\dots,c_{n-1},m) = \forall j \in (m \setminus E).\ \tilde{I}_{j,c_j}(c_0,\dots,c_{j-1},c_{j+1},\dots,c_{n-1},m)$$

Le premier terme correspond à la preuve séquentielle (du processus i), le deuxième terme correspond à la preuve d'absence d'interférences (entre i et tout $k \neq i$), le terme de contexte apportant l'information disponible dans les autres processus n'apparaissant pas dans l'exemple [77].

Remarquons que le terme correspondant à l'absence d'interférences disparait quand $n=1$ et que dans ce cas les conditions de vérification correspondent exactement dans tous les cas à celle de la méthode de Floyd [67].

Nous illustrons la condition de vérification :

$$\forall i \in n,\ c_i \in C_i.$$
$$\tilde{F}(\tilde{I})_{i,c_i}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{n-1},m) \Rightarrow \tilde{I}_{i,c_i}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{n-1},m)$$

sur l'exemple suivant :

[
11: $\{\tilde{I}_{11}(c_2, x)\}$
$X := X+1;$
12: $\{\tilde{I}_{12}(c_2, x)\}$
‖
21: $\{\tilde{I}_{21}(c_1, x)\}$
$X := X+2;$
22: $\{\tilde{I}_{22}(c_1, x)\}$
]

ce qui donne :

- Preuve séquentielle :

$$[\tilde{I}_{11}(c_2,x') \wedge (c_2 = 21 \Rightarrow \tilde{I}_{21}(11,x')) \wedge (c_2 = 22 \Rightarrow \tilde{I}_{22}(11,x')) \wedge x = x'+1] \Rightarrow \tilde{I}_{12}(c_2,x)$$
$$[\tilde{I}_{21}(c_1,x') \wedge (c_1 = 11 \Rightarrow \tilde{I}_{11}(21,x')) \wedge (c_1 = 12 \Rightarrow \tilde{I}_{12}(21,x')) \wedge x = x'+2] \Rightarrow \tilde{I}_{22}(c_1,x)$$

- Absence d'interférences :

$$[\tilde{I}_{11}(21,x') \wedge \tilde{I}_{21}(11,x') \wedge x = x'+2] \Rightarrow \tilde{I}_{11}(22,x)$$
$$[\tilde{I}_{12}(21,x') \wedge \tilde{I}_{21}(12,x') \wedge x = x'+2] \Rightarrow \tilde{I}_{12}(22,x)$$
$$[\tilde{I}_{21}(11,x') \wedge \tilde{I}_{11}(21,x') \wedge x = x'+1] \Rightarrow \tilde{I}_{21}(12,x)$$
$$[\tilde{I}_{22}(11,x') \wedge \tilde{I}_{11}(22,x') \wedge x = x'+1] \Rightarrow \tilde{I}_{22}(12,x)$$

Nous définissons maintenant $\tilde{F} \in (A\tilde{\lambda} \to A\tilde{\lambda})$ par :

$$\tilde{F}(\tilde{I})_{i,c_i}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{n-1},m)$$
$$= [\exists c'_i \in C_i, m' \in M.\ \tilde{I}_{i,c_i}(c_0,\dots,c_{i-1},c_{i+1},\dots,c_{n-1},m') \wedge$$
$$t[\![Pra_i]\!](\langle c'_i,m'\rangle,\langle c_i,m\rangle)$$
$$\vee\ [\exists k \in (m \setminus i), c'_k \in C_k, m' \in M.\ \tilde{I}_{k,c'_k}(c_0,\dots,c_{k-1},c_{k+1},\dots,c_{n-1},m') \wedge$$
$$t[\![Pra_k]\!](\langle c'_k,m'\rangle,\langle c_k,m\rangle)]$$

Nous observons alors que $\alpha \circ F = \tilde{F} \circ \alpha$. Les conditions de vérification correspondant à $\tilde{F}(\tilde{I}) \Rightarrow \tilde{I}$ sont similaires à celles introduites par Newton [75] (bien que la similitude soit difficile à saisir car Newton utilise une définition des programmes parallèles un peu singulière).

Sur notre exemple, nous obtenons :

- Preuve séquentielle :

$[\tilde{I}_{11}(c_2, x') \wedge x = x'+1] \Rightarrow \tilde{I}_{12}(c_2, x)$

$[\tilde{I}_{21}(c_1, x') \wedge x = x'+2] \Rightarrow \tilde{I}_{22}(c_1, x)$

- Absence d'interférences :

$[\tilde{I}_{21}(11, x') \wedge x = x'+2] \Rightarrow \tilde{I}_{11}(22, x)$

$[\tilde{I}_{21}(12, x') \wedge x = x'+2] \Rightarrow \tilde{I}_{12}(22, x)$

$[\tilde{I}_{11}(21, x') \wedge x = x'+1] \Rightarrow \tilde{I}_{21}(12, x)$

$[\tilde{I}_{11}(22, x') \wedge x = x'+1] \Rightarrow \tilde{I}_{22}(12, x)$

Comme $\underset{\sim}{F} \neq \tilde{F}$, nous obtenons un treillis complet de conditions de vérification $\bar{F}(\tilde{I}) \Rightarrow \tilde{I}$ où $\underset{\sim}{F} \Rightarrow \bar{F} \Rightarrow \tilde{F}$ correspondant chacune à une méthode de preuve particulière. Par exemple, la méthode de Lamport [77] (ainsi bien que celle de Owicki-Gries [76a] si on avait utilisé la définition de α et γ donnée en 4.3.2.4.3) correspond à :

$$\bar{F}(\tilde{I})_{i,c_i}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{n-1}, m)$$
$$= [\exists c'_i \in C_i,\ m' \in M.\ \tilde{I}_{i,c'_i}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{n-1}, m') \wedge t[\![Pra_i]\!](\langle c'_i, m'\rangle, \langle c_i, m\rangle)]$$
$$\vee [\exists k \in (n \setminus i),\ c'_k \in C_k,\ m' \in M.\ \tilde{I}_{i,c_i}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{k-1}, c'_k, c_{k+1}, \dots, c_{n-1}, m') \wedge$$
$$\tilde{I}_{k,c'_k}(c_0, \dots, c_{k-1}, c_{k+1}, \dots, c_{n-1}, m') \wedge t[\![Pra_k]\!](\langle c'_k, m'\rangle, \langle c_k, m\rangle)]$$

D'après 4.3.1.7-1 la méthode de Lamport [77] est donc correcte et d'après 4.3.1.8-2, elle est sémantiquement complète.

Pour notre exemple, nous obtenons :

- Preuve séquentielle :

$[\tilde{I}_{11}(c_2, x') \wedge x = x'+1] \Rightarrow \tilde{I}_{12}(c_2, x)$

$[\tilde{I}_{21}(c_1, x') \wedge x = x'+2] \Rightarrow \tilde{I}_{22}(c_1, x)$

- Absence d'interférences

$[\tilde{I}_{11}(21, x') \wedge \tilde{I}_{21}(11, x') \wedge x = x'+2] \Rightarrow \tilde{I}_{11}(22, x)$

$[\tilde{I}_{12}(21, x') \wedge \tilde{I}_{21}(12, x') \wedge x = x'+2] \Rightarrow \tilde{I}_{12}(22, x)$

$[\tilde{I}_{21}(11, x') \wedge \tilde{I}_{11}(21, x') \wedge x = x'+1] \Rightarrow \tilde{I}_{21}(12, x)$

$[\tilde{I}_{12}(11, x') \wedge \tilde{I}_{11}(22, x') \wedge x = x'+1] \Rightarrow \tilde{I}_{22}(12, x)$

Observons que bien que toutes les méthodes dérivées d'un F tel que $\underset{\sim}{F} \Rightarrow F \Rightarrow \tilde{F}$ soient sémantiquement complètes, il se peut que pour certains programmes, on puisse montrer que des assertions sont invariantes en utilisant $\underset{\sim}{F}$ et qu'on ne puisse pas le faire en utilisant $\tilde{F}$. C'est le cas dans notre exemple pour :

$\tilde{I}_{11}(c_2, x) = [\text{pair}(x)]$ $\qquad$ $\tilde{I}_{21}(c_1, x) = [x = 0 \vee x = 1]$

$\tilde{I}_{12}(c_2, x) = [x = 1 \vee x = 3]$ $\qquad$ $\tilde{I}_{22}(c_1, x) = [x = 2 \vee x = 3]$

Toutefois, on peut toujours renforcer l'invariant et remplacer

$\tilde{I}_{i,c_i}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{n-1}, m)$

par

$\tilde{I}_{i,c_i}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{n-1}, m) \wedge \underline{\text{contexte}}\{i\}(\tilde{I})(c_0, \dots, c_{n-1}, m)$

donc toutes les conditions de vérification $F(\tilde{I}) \Rightarrow \tilde{I}$ sont fortement équivalentes. □

Exemple 4.3.2.4.4-2

Nous démontrons la correction partielle du programme 2.8.2.3, mais au lieu d'utiliser le principe d'induction (-i), nous utilisons (-i), de manière à pouvoir relier la valeur courante n de la variable N à sa valeur initiale $\underline{n}$.

Puisque les deux processus sont symétriques, nous n'avons besoin de raisonner que sur le processus 1. Nous allons montrer que la relation :

$$Inv(\underline{m}, c_2, m, p_1, p_2) = [(c_2 = 21 \wedge p_1 = 2^{\underline{m}-m}) \vee (c_2 \neq 21 \wedge p_1 \times p_2 = 2^{\underline{m}-m})]$$

est invariante dans le processus 1 après initialisation de la variable P1. Puisque la correction partielle découle de l'invariant quand $c_2 = 25$ (le processus 2 a terminé) et $0 \leq m \leq 1$, nous allons aussi montrer que la valeur de N après l'exécution de la commande parallèle est soit 0 ou 1. Puisque la valeur initiale $\underline{m}$ de N est supposée positive, la seule difficulté est quand $\underline{m} > 1$. Dans ce cas N est décrémenté jusqu'à atteindre la valeur 2. D'une part, chacun des deux processus peut tester que $N > 1$ avant qu'il ne soit décrémenté par l'autre, le décrémenter et terminer. Dans ce cas N vaudra 0 après l'exécution de la commande parallèle. D'autre part, un processus peut tester si $N > 1$ et le décrémenter avant que l'autre ne teste si $N > 1$. Alors les deux processus terminent et $N = 1$ après exécution de la commande parallèle. Pour une preuve d'invariance, ces arguments opérationnels peuvent s'exprimer "indépendemment du temps", d'où les invariants locaux suivants :

$$\tilde{I}_{11}(\underline{m}, c_2, m, p_1, p_2) = \tilde{I}_{12}(\underline{m}, c_2, m, p_1, p_2)$$

$$\tilde{I}_{12}(\underline{m}, c_2, m, p_1, p_2) = [Inv(\underline{m}, c_2, m, p_1, p_2) \wedge [(c_2 \in \{21, 22\} \wedge m = \underline{m} \wedge m > 0) \vee (c_2 = 23 \wedge m > 1) \vee (c_2 = 24 \wedge m \geq 1) \vee (c_2 = 25 \wedge 0 \leq m \leq 1)]]$$

$$\tilde{I}_{13}(\underline{m}, c_2, m, p_1, p_2) = [Inv(\underline{m}, c_2, m, p_1, p_2) \wedge [(c_2 \in \{21, 22, 23\} \wedge m > 1) \vee (c_2 = 24 \wedge m > 1) \vee (c_2 = 25 \wedge m = 1)]]$$

$$\tilde{I}_{14}(\underline{m}, c_2, m, p_1, p_2) = [Inv(\underline{m}, c_2, m, p_1, p_2) \wedge [(c_2 \in \{21, 22, 23\} \wedge m > 0) \vee (c_2 = 24 \wedge m \geq 0) \vee (c_2 = 25 \wedge 0 \leq m \leq 1)]]$$

$$\tilde{I}_{15}(\underline{m}, c_2, m, p_1, p_2) = [Inv(\underline{m}, c_2, m, p_1, p_2) \wedge [(c_2 = 23 \wedge m = 1) \vee (c_2 \neq 23 \wedge 0 \leq m \leq 1)]]$$

$$\tilde{I}_1(\underline{m}, m, p_1, p_2) = [p_1 \times p_2 = 2^{\underline{m}-m} \wedge 0 \leq m \leq 1]$$

$$\tilde{I}_2(\underline{m}, p) = [p = 2^{\underline{m}}]$$

C'est simple de montrer que ces invariants locaux satisfont les conditions de vérification suivantes :

- Initialisation :

$$[\underline{m} > 0] \Rightarrow [\tilde{I}_{11}(\underline{m}, 21, \underline{m}, p_1, p_2) \wedge \tilde{I}_{21}(\underline{m}, 11, \underline{m}, p_2, p_1)]$$

- Preuve séquentielle :

$$\tilde{I}_{11}(\underline{m}, c_2, m, p_1', p_2) \Rightarrow \tilde{I}_{12}(\underline{m}, c_2, m, 1, p_2)$$

$$[\tilde{I}_{12}(\underline{m}, c_2, m, p_1, p_2) \wedge m > 1] \Rightarrow \tilde{I}_{13}(\underline{m}, c_2, m, p_1, p_2)$$

$$[\tilde{I}_{12}(\underline{m}, c_2, m, p_1, p_2) \wedge m \leq 1] \Rightarrow \tilde{I}_{15}(\underline{m}, c_2, m, p_1, p_2)$$

$$\tilde{I}_{13}(\underline{m}, c_2, m', p_1', p_2) \Rightarrow \tilde{I}_{14}(\underline{m}, c_2, m'-1, 2p_1', p_2)$$

$$[\tilde{I}_{14}(\underline{m}, c_2, m, p_1, p_2) \wedge m > 1] \Rightarrow \tilde{I}_{13}(\underline{m}, c_2, m, p_1, p_2)$$

$$[\tilde{I}_{14}(\underline{m}, c_2, m, p_1, p_2) \wedge m \leq 1] \Rightarrow \tilde{I}_{15}(\underline{m}, c_2, m, p_1, p_2)$$

- Preuve d'absence d'interférences :

Pour $k = 1, \ldots, 5$

$$[\tilde{I}_{1k}(\underline{m}, 21, m, p_1, p_2') \wedge \tilde{I}_{21}(\underline{m}, 1k, m, p_1, p_2')] \Rightarrow \tilde{I}_{1k}(\underline{m}, 22, m, p_1, 1)$$

$$[\tilde{I}_{1k}(\underline{m}, 22, m, p_1, p_2) \wedge \tilde{I}_{22}(\underline{m}, 1k, m, p_1, p_2) \wedge m > 1] \Rightarrow \tilde{I}_{1k}(\underline{m}, 23, m, p_1, p_2)$$

$$[\tilde{I}_{1k}(\underline{m}, 22, m, p_1, p_2) \wedge \tilde{I}_{22}(\underline{m}, 1k, m, p_1, p_2) \wedge m \leq 1] \Rightarrow \tilde{I}_{1k}(\underline{m}, 25, m, p_1, p_2)$$

$$[\tilde{I}_{1k}(\underline{m}, 23, m', p_1, p_2') \wedge \tilde{I}_{23}(\underline{m}, 1k, m', p_1, p_2')] \Rightarrow \tilde{I}_{1k}(\underline{m}, 24, m'-1, p_1, 2 \times p_2')$$

$$[\tilde{I}_{1k}(\underline{m}, 24, m, p_1, p_2) \wedge \tilde{I}_{24}(\underline{m}, 1k, m, p_1, p_2) \wedge m > 1] \Rightarrow \tilde{I}_{1k}(\underline{m}, 23, m, p_1, p_2)$$

$$[\tilde{I}_{1k}(\underline{m}, 24, m, p_1, p_2) \wedge \tilde{I}_{24}(\underline{m}, 1k, m, p_1, p_2) \wedge m \leq 1] \Rightarrow \tilde{I}_{1k}(\underline{m}, 25, m, p_1, p_2)$$

- Finalisation :

$$[\tilde{I}_{15}(\underline{m}, 25, m, p_1, p_2) \wedge \tilde{I}_{25}(\underline{m}, 15, m, p_2, p_1)] \Rightarrow \tilde{I}_1(\underline{m}, m, p_1, p_2)$$

$$[\tilde{I}_1(\underline{m}, m, p_1, p_2) \wedge m = 0] \Rightarrow \tilde{I}_2(\underline{m}, p_1 \times p_2)$$

$$[\tilde{I}_1(\underline{m}, m, p_1, p_2) \wedge m \neq 0] \Rightarrow \tilde{I}_2(\underline{m}, 2 \times p_1 \times p_2)$$

$$\tilde{I}_2(\underline{m}, p) \Rightarrow [p = 2^{\underline{m}}]$$

Noter que pour tout le programme, nous avons obtenu 77 conditions de vérification, ce qui est évidemment énorme mais peut être réduit considérablement en choisissant une décomposition moins fine (ex : 3.3.1.5 - 3).

4.3.2.4.5 Utilisation d'invariants sur les variables et des variables auxiliaires associés à chaque point de contrôle du programme

Après s'être rendus compte que l'utilisation d'invariants sur les variables associés à chaque point de contrôle du programme (comme en 4.3.2.4.3 qui leur semblait la généralisation la plus naturelle de la méthode de Floyd[67] au cas des programmes parallèles) était incomplète, Owicki-Gries[76a] ont introduit la technique des variables auxiliaires. Nous montrons que ces variables auxiliaires ne sont qu'un moyen de raisonner implicitement sur les états de contrôle du programme.

Pour présenter la technique, nous considérons un programme asynchrone $Ppa \equiv [\![Pra_0 \| \dots \| Pra_{n-1}]\!]$ dont, pour simplifier, le prélude et le postlude sont supposés vides. L'ensemble des états du programme est donc $S = (C_0 \times \dots \times C_{n-1}) \times M$ où C_i est l'ensemble des points de contrôle du processus Pra_i et M l'ensemble des états mémoire. Notons L_i le point d'entrée du processus Pra_i (de sorte que $Pra_i \equiv L_i : \alpha$) et $C = \bigcup_{i \in n} C_i$ l'ensemble des points de contrôle du programme.

En chaque point $L \in C$ du programme Ppa est associé un commentaire qui est une relation $\psi_L \in (M^2 \to \{t, ff\})$ ne portant que sur les états mémoire et qui s'interprète comme signifiant que :

$$\Psi \in (S \times S \to \{t, ff\})$$

$$\Psi(\langle\langle L_0, \dots, L_{n-1}\rangle, m\rangle, \langle\langle L'_0, \dots, L'_{n-1}\rangle, m'\rangle) = [\bigwedge_{i=0}^{n-1} (L_i = L_i \wedge \psi_{L_i}(m, m'))]$$

est invariante. Remarquons que Ψ permet de décrire les états de contrôle du programme mais sous une forme extrêmement limitée (puisque par exemple, il est possible d'exprimer qu'un point L de processus est inaccessible en n'importe quel état mémoire mais il n'est pas possible d'exprimer de relations entre les états de contrôle des

différents processus du programme, comme par exemple que les points L_i et L_j sont mutuellement exclusifs).

Pour démontrer l'invariance de Ψ pour Ppa, la technique de Owicki-Gries[76a] consiste à considérer un programme transformé Ppa' de la forme $[\![Pra'_0 \| \dots \| Pra'_{n-1}]\!]$ où chaque processus Pra'_i a essentiellement le même comportement que Pra_i mais contient des commandes d'affectation à des variables auxiliaires n'apparaissant pas dans Ppa et des invariants ψ'_L (portant sur les variables de Ppa mais également sur les variables auxiliaires) qui impliquent les ψ_L.

Plus précisément, l'ensemble VA des variables auxiliaires du programme Ppa' est le plus grand ensemble de variables qui n'apparaissent qu'en membres gauches d'affectations ou en membres droits d'affectations à des variables auxiliaires.

Ppa s'obtient à partir de Ppa' en supprimant les affectations aux variables auxiliaires c'est-à-dire en répétant un nombre fini de fois la transformation $\alpha L : V := E ; \beta \to \alpha\beta$ où $V \in VA$ (puis en effectuant s'il y a lieu certaines simplifications évidentes comme par exemple, $1 : \langle 2 : V := E ; 3 : \rangle ; 4 :$ est équivalent à $1 : V := E ; 4 :$ ou $1 : \underline{skip} ; 2 : \underline{skip} ; 3 :$ est équivalent à $1 : \underline{skip} ; 3 :$).

L'ensemble des états mémoire de Ppa' peut donc être compris (à un isomorphisme près) comme étant $M \times M'$ où $M' = (VA \to \mathbb{D})$. Comme précédemment, notons C'_i l'ensemble des points de contrôle du processus Pra'_i $i \in n$, L'_i le point d'entrée du processus Pra'_i et $C' = \bigcup_{i \in n} C'_i$.

En chaque point $L' \in C'$ du programme Ppa' est associé un commentaire qui est une relation $\psi'_{L'} \in ((M \times M')^2 \to \{t, ff\})$ ne portant que sur les variables de Ppa et les variables auxiliaires et qui s'interprète comme précédemment par l'invariance de Ψ' telle que :

$$\Psi'(\langle\langle L_0, \dots, L_{n-1}\rangle, m\rangle, \langle\langle L'_0, \dots, L'_{n-1}\rangle, m'\rangle) = [\bigwedge_{i=0}^{n-1} (L_i = L'_i \wedge \psi'_{L'_i}(m, m'))]$$

Les Ψ'_L doivent être choisis de sorte que pour tous $\underline{m}, \underline{m}', m, m' \in \mathcal{D}$, on ait :

$\Psi'_L(\langle \underline{m}, \underline{m}' \rangle, \langle m, m' \rangle) \rightarrow \Psi_L(\underline{m}, m)$

4.3.2.4.5.1 Correction de la méthode

Observons que la sémantique du programme Ppa s'obtient à partir de la sémantique du programme auxiliaire Ppa' par élimination des états inobservables (appartenant à $((C'_0 \times \ldots \times C'_{m-1}) \vee (C_0 \times \ldots \times C_{m-1})) \times M \times M'$) puis par correspondance à une fonction $f_\Delta \in ((C_0 \times \ldots \times C_{m-1}) \times M \times M' \rightarrow (C_0 \times \ldots \times C_{m-1}) \times M)$ entre états près (définie par $f_\Delta(\langle\langle c_0, \ldots, c_{m-1} \rangle, m, m' \rangle) = \langle\langle c_0, \ldots, c_{m-1} \rangle, m \rangle$).
La preuve formelle, simple mais fastidieuse, repose sur le lemme 4.5.4.1.1. De plus, on a $\Psi'(\underline{\Delta}, \Delta) \Rightarrow \Psi(f_\Delta(\underline{\Delta}), f_\Delta(\Delta))$ quand $\Delta \in ((C_0 \times \ldots \times C_{m-1}) \times M \times M')$.

Ceci permet de conclure que l'utilisation de variables auxiliaires est correcte dans la mesure où une propriété invariante pour le programme auxiliaire l'est également pour le programme original obtenu par élimination des variables auxiliaires. C'est la conséquence du théorème 4.1.1.v5 et du théorème 4.1.2.v1 (qui s'applique au cas où les états initiaux des programmes Ppa et Ppa' sont identiques et se généralise aisément).

Remarquons que, d'après le théorème 4.1.1.v5, la méthode de Owicki-Gries se généralise de manière évidente aux preuves d'invariance conditionnelle (dans la mesure où en pratique ϕ ne porterait pas sur les variables auxiliaires).

De la même façon, on constate que la limitation de l'usage des variables auxiliaires aux affectations dans méthode de Owicki-Gries est inutile. Par exemple, les variables auxiliaires peuvent être utilisées dans le reste dans la mesure où ces modifications laissent le flot de contrôle du programme globalement inchangé

4.3.2.4.5.2 Complétude sémantique de la méthode

Pour démontrer la correction partielle de l'exemple :

0: X:=0; 1: [[11: X:=X+1; 12: || 21: X:=X+1; 22:]]; 2:

utilisé pour démontrer que la méthode 4.3.2.4.3 est incomplète), on peut transformer le programme comme suit :

```
0: < 01: X:=0;  02: v1:=11;  03: v2:=21;  04: >;
1: [[ 11:
        < 111: X:=X+1;  112: v1:=12;  113: >;
     12:
   ||
     21:
        < 211: X:=X+1;  212: v2:=22;  213: >;
     22:
   ]];
2:
```

et utiliser les invariants :

$\Psi'_1(x, v1, v2) = [v1=11 \wedge v2=21 \wedge x=0]$

$\Psi'_{11}(x, v1, v2) = [v1=11 \wedge ((v2=21 \wedge x=0) \vee (v2=22 \wedge x=1))]$

$\Psi'_{12}(x, v1, v2) = [v1=12 \wedge ((v2=21 \wedge x=1) \vee (v2=22 \wedge x=2))]$

$\Psi'_{21}(x, v1, v2) = [v2=21 \wedge ((v1=11 \wedge x=0) \vee (v1=12 \wedge x=1))]$

$\Psi'_{12}(x, v1, v2) = [v2=22 \wedge ((v1=11 \wedge x=1) \vee (v1=12 \wedge x=2))]$

$\Psi'_2(x, v1, v2) = [x=2]$

qui sont ceux qu'on aurait utilisé avec la méthode (à la différence près que les variables v1 et v2 sont utilisées pour simuler les compteurs ordinaux des deux processus).

De manière plus générale, considérons un programme parallèle synchrone Ppa de la forme $[\![Pra_0 \| \ldots \| Pra_{m-1}]\!]$ et des Ψ_L, $L \in C$ tels que Ψ, défini comme précédemment soit invariant. Il faut montrer qu'on peut trouver Ψ' dont on peut démontrer l'invariance pour un programme séquentiel Ppa' sans se référer aux compteurs ordinaux du programme Ppa.

Construisons Ppa' en simulant le compteur ordinal de chaque processus Pra_i au moyen d'une variable auxiliaire V'_i (n'apparaissant pas dans le programme Ppa). Pour cela nous supposerons qu'il existe un ensemble de valeurs de cette variable qui, au moyen d'un codage c bijectif, est équipotent à l'ensemble des étiquettes apparaissant dans le processus Pra_i et nous supposerons également qu'aux étiquettes L de Ppa correspondent des étiquettes $L', L'_1, L'_2, \ldots$ n'apparaissant pas dans Ppa et ces étiquettes étant toutes deux à deux distinctes. Ppa' s'obtient à partir de Ppa par la transformation τ suivante :

. $\tau(\underline{L}: [\![Pra_0 \| \ldots \| Pra_{m-1}]\!]; \bar{L}) =$

$\underline{L}: V_0 := \underline{L}_0;\ \underline{L}'_1: V_1 := \underline{L}_1; \ldots;\ \underline{L}'_{m-1}: V_{m-1} := \underline{L}_{m-1};\ \underline{L}'_m:$

$[\![\tau_0(Pra_0) \| \ldots \| \tau_{m-1}(Pra_{m-1})]\!]; \bar{L}:$

. $\tau_i(L:) = L:$

. $\tau_i(L1: Ca;\ L2:\alpha) =$

$L1: \not\downarrow L1'1: Ca;\ L1'2: V_i := c(L2);\ L1'3: \not\downarrow;\ \tau_i(L2:\alpha)$

quand Ca est skip, $V_i := E$, $V_i := ?$ ou $\not\downarrow Ps \not\downarrow$

. $\tau_i(L1:$ if B then $L2:\alpha;\ L3:$ else $L4:\beta;\ L5:$ fi; $L6:\gamma)$

$L1: \not\downarrow L1'1:$ if B then

$L1'2: T_i := true;\ L1'3: V_i := c(L2);\ L1'3:$

else

$L1'5: T_i := false;\ L1'6: V_i := c(L3);\ L1'4:$

fi; $L1'8: \not\downarrow;$

$L1'':$ if T_i then $\tau_i(L2:\alpha; L3:)$ $V_i := c(L6);\ L3''':$

else $\tau_i(L4:\beta; L5:)$ $V_i := c(L5);\ L5''':$

fi;

$\tau(L6:\gamma)$

$\tau_i(L1:$ while B do $L2:\alpha; L3:$ od; $L4:\beta) =$

$L1: \not\downarrow L1'1:$ if B then

$L1'2: T_i := true;\ L1'3: V_i := c(L2);\ L1'4:$

else

$L1'5: T_i := false;\ L1'6: V_i := c(L4);\ L1'7:$

fi; $L1'8: \not\downarrow;$

$L1'':$ while T_i do

$\tau_i(L2:\alpha;\ L3:)$

$\not\downarrow L3'1:$ if B then

$L3'2: T_i := true;\ L3'3: V_i := c(L2);\ L3'4:$

else

$L3'5: T_i := false;\ L3'6: V_i := c(L4);\ L3'7:$

fi; $L3'8: \not\downarrow;$

$L3'':$

od;

$\tau_i(L4:\beta)$

Définissons maintenant Ψ' comme caractérisant exactement les descendants des états d'entrée de Ppa (au codage des états de contrôle de Ppa par des valeurs des variables auxiliaires de Ppa' près):

Pour tous états $\underline{s}, s$ de Ppa, définissons :

$$\Psi''(\underline{s}, s) = [\, \varepsilon(\underline{s}) \wedge (\bigvee_{i \in m} t[\![Pra_i]\!])^*(\underline{s}, s)\,]$$

avec

$$\varepsilon(\langle\!\langle L_0, \ldots, L_{m-1} \rangle, M \rangle) = (\bigwedge_{i \in m} L_i = \underline{L}_i)$$

de sorte que Ψ'' est invariante pour Ppa. Posons alors :

- Pour toutes les étiquettes L qui figurent dans Ppa et Ppa', nous supposons que la transformation de Ppa en Ppa' est telle que lorsque le contrôle est au point L du $i$ème processus de Ppa', on a $V_i = c(L)$. On choisit donc :

$\Psi'_L(\langle \underline{M},\underline{M}'\rangle,\langle M,M'\rangle) =$

$\quad \Psi''(\langle\langle c^{-1}(\underline{M}'(V_0)),\dots,c^{-1}(\underline{M}'(V_{m-1}))\rangle,\underline{M}\rangle,$

$\qquad \langle\langle c^{-1}(M'(V_0)),\dots,c^{-1}(M'(V_{m-1}))\rangle, M\rangle)$

de sorte que l'anertion en L dans Ppa' est le plus fort invariant de Ppa en L obtenu en transcrivant le contrôle en terme de variables auxiliaires.

- Pour toutes les étiquettes notées Li'j qui figurent dans une section critique de Ppa' mais pas dans Ppa, on utilise des assertions intermédiaires liant les valeurs courantes des variables (y compris auxiliaires) aux valeurs (symboliques) de ces mêmes variables au début de la section critique, ce qui permet de traiter les sections critiques comme une action atomique (cf. 4.3.2.3.2.4).

- Les autres étiquettes figurant dans Ppa' mais pas dans Ppa ont été introduites à cause des tests (dans une commande alternative ou itérative). Pour les étiquettes notées L'' qu'on trouve sous la forme

... L: ‖ L'1 : if B then

L'2 : T_L := true ; L'3 : V_L := c(L2) ; L'4 :

else

L'5 : T_L := false ; L'6 : V_L := c(L4) ; L'7 :

fi ; L'8 : ‖ ;

L'' : ...

dans Ppa', on choisit :

$\Psi'_{L''}(\langle \underline{M},\underline{M}'\rangle,\langle M,M'\rangle) = [M \in \underline{\mathrm{dom}}(\mathbb{B}[\![B]\!]) \wedge$

$\quad [\mathbb{B}[\![B]\!](M) \wedge M'(T_L) \wedge M'(V_L) = c(L2) \wedge \Psi'_{L2}(\langle \underline{M},\underline{M}'\rangle,\langle M,M'\rangle)]$

$\quad \vee [\neg\mathbb{B}[\![B]\!](M) \wedge \neg M'(T_L) \wedge M'(V_L) = c(L4) \wedge \Psi'_{L4}(\langle \underline{M},\underline{M}\rangle,\langle M,M'\rangle)]]$

- Pour les étiquettes notées L''' qu'on trouve sous la forme

... L1 :

V_L := c(L2) ;

L''' : ...

dans Ppa' et qui précèdent un else ou un fi, on choisira

$\Psi'_{L'''}(\langle \underline{M},\underline{M}'\rangle,\langle M,M'\rangle) =$

$\quad [\Psi'_{L1}(\langle \underline{M},\underline{M}'\rangle,\langle M,M'\rangle) \wedge M'(V_L) = c(L2)]$

On vérifie alors aisément (bien que les calculs soient très fastidieux) que les conditions de vérification correspondant à l'application de la méthode 4.3.2.4.3 pour Ψ' et Ppa' sont satisfaites (car Ψ'' satisfait les conditions de vérification de la méthode 4.3.2.4.4 pour Ppa) et que

$\Psi'_L(\langle \underline{M},\underline{M}'\rangle,\langle M,M'\rangle) \rightarrow \Psi_L(\underline{M},M)$

pour toutes les étiquettes figurant dans Ppa et Ppa'.

4.3.2.4.6 Utilisation d'invariants sur l'état de contrôle et les variables associés à chaque processus du programme

Si on utilise les décompositions 4.3.2.4.3 ou 4.3.2.4.4 pour un programme $[\![Pra_0 \| \dots \| Pra_{n-1}]\!]$ où chaque processus Pra_i, $i \in n$ a m_i points de contrôle, il y a $\sum_{i \in n} (m_i - 1)$ conditions de vérifications correspondant à la preuve séquentielle et $\sum_{i \in n} \sum_{j \in m_i} \sum_{k \in (m \setminus i)} (m_k - 1) = \sum_{i \neq j} m_i (m_j - 1)$ conditions de vérification correspondant à la preuve d'absence d'interférences.

Pour éviter la prolifération des conditions de vérification correspondant à la preuve d'absence d'interférences, on peut choisir une décomposition moins fine, comme celle proposée par Lamport [80] qui consiste à associer un invariant global à chaque processus du programme.

Exemple 4.3.2.4.6-1

Pour démontrer la correction partielle du programme

$[\![11 : x := x+1 ; 12 : \| 21 : x := x+2 ; 22 :]\!]$

il faut vérifier les conditions suivantes :

$\phi(x) \Rightarrow [I_1(11, 21, x) \wedge I_2(11, 21, x)]$

$[I_1(11, c_2, x') \wedge x = x'+1] \Rightarrow I_1(12, c_2, x)$

$[I_2(c_1, 21, x') \wedge x = x'+2] \Rightarrow I_2(c_1, 22, x)$

$[I_1(c_1, 21, x') \wedge x = x'+2] \Rightarrow I_1(c_1, 22, x)$

$[I_2(11, c_2, x') \wedge x = x'+1] \Rightarrow I_2(12, c_2, x)$

$[I_1(12, 22, x) \wedge I_2(12, 22, x)] \Rightarrow \psi(x)$

□

La décomposition choisie est la suivante (Cousot-Cousot [84]) :

$S = (C_0 \times \dots \times C_{n-1}) \times M$

$A\Delta = (S \to \{\mathit{tt}, \mathit{ff}\})$

$\tilde{A\Delta} = (m \to (C_0 \times \dots \times C_{n-1} \times M \to \{\mathit{tt}, \mathit{ff}\}))$

$\alpha \in (A\Delta \to \tilde{A\Delta})$

$\alpha(I)_i\ (c_0, \dots, c_{n-1}, m) = I(\langle\langle c_0, \dots, c_{n-1} \rangle, m \rangle)$

$\gamma \in (\tilde{A\Delta} \to A\Delta)$

$\gamma(\tilde{I})(\langle\langle c_0, \dots, c_{n-1} \rangle, m \rangle) = \bigwedge_{i \in n} \tilde{I}_i (c_0, \dots, c_{n-1}, m)$

Exemple 4.3.2.4.6-2

La détermination des conditions de vérification pour démontrer la correction partielle de programmes parallèles asynchrones de la forme :

$[\![Pra_0 \| \dots \| Pra_{n-1}]\!]$

est tout à fait similaire à 4.3.2.4.3-1. Nous obtenons (Cousot-Cousot [84]):

- Initialisation

$\phi(m) \to \forall i \in n.\ \tilde{I}_i (\underline{\ }_0, \dots, \underline{\ }_{n-1}, m)$ (où $Pra_i \equiv \underline{\ }_i : \alpha$, $i \in n$)

- Preuve séquentielle

$[\tilde{I}_i (c_0, \dots, c_{i-1}, c'_i, c_{i+1}, \dots, c_{n-1}, m') \wedge t[\![Pra_i]\!](\langle c'_i, m' \rangle, \langle c_i, m \rangle)]$
$\Rightarrow \tilde{I}_i (c_0, \dots, c_{i-1}, c_i, c_{i+1}, \dots, c_{n-1}, m)$

- Preuve d'absence d'interférences, $(i \neq k)$:

$[\tilde{I}_i (c_0, \dots, c_{k-1}, c'_k, c_{k+1}, \dots, c_{n-1}, m') \wedge t[\![Pra_k]\!](\langle c'_k, m' \rangle, \langle c_k, m \rangle)]$
$\Rightarrow \tilde{I}_i (c_0, \dots, c_{k-1}, c_k, c_{k+1}, \dots, c_{n-1}, m')$

- Finalisation

$\forall i \in n.\ \tilde{I}_i (\underline{\ }_0, \dots, \underline{\ }_{n-1}, m) \Rightarrow \psi(m)$ (où $Pra_i \equiv \alpha : \underline{\ }_i$, $i \in n$)

Exemple 4.3.2.4.6-3

Nous illustrons la méthode de preuve 4.3.2.4.6-2 sur l'exemple 2.8.2.3. De manière évidente il ne peut s'agir que d'une reformulation de 4.3.2.4.4-2 en utilisant un invariant global par processus plutôt que des invariants locaux associés à chaque point de contrôle du programme. Comme les processus du programme sont symétriques, nous utilisons le même invariant $\tilde{I}$ pour $\tilde{I}_1$ et $\tilde{I}_2$, ce qui permet de n'avoir à raisonner que sur un seul processus.

Pour pouvoir désigner certains états de contrôle du programme, nous introduisons (cf. Cousot-Cousot [84]) :

Not-started = $(c_1 = 11 \wedge c_2 = 21)$

Pro_2-started = $(c_1 = 11 \wedge c_2 \neq 21)$

Pro_1-started = $(c_1 \neq 11 \wedge c_2 = 21)$

Started = $(c_1 \neq 11 \wedge c_2 \neq 21)$

L'idée centrale du programme 2.8.2.3 est de maintenir invariante la relation suivante :

$$Inv(\underline{m}, c_1, c_2, m, p_1, p_2) = [(\text{Not-started} \wedge m = \underline{m}) \vee (Pro_2\text{-started} \wedge p_2 = 2^{\underline{m}-m}) \vee (Pro_1\text{-started} \wedge p_1 = 2^{\underline{m}-m}) \vee (\text{started} \wedge p_1 \times p_2 = 2^{\underline{m}-m})]$$

L'autre observation essentielle pour la preuve de correction partielle est que le programme peut se terminer seulement quand $0 \leq m \leq 1$. Pour montrer ceci, nous introduisons :

Before-test = $[(c_1 \in \{11,12\} \wedge c_2 \in \{21,22\}) \vee (c_1 = 14 \wedge c_2 = 24)]$

After-test = $[(c_1 = 13 \wedge c_2 \in \{21,22,23\}) \vee (c_1 \in \{11,12,13\} \wedge c_2 = 23)]$

After-test-and-decrement = $[(c_1 = 14 \wedge c_2 \in \{21,22,23\}) \vee (c_1 \in \{11,12,13\} \wedge c_2 = 24)]$

one-decrement-left = $[(c_1 = 15 \wedge c_2 = 23) \vee (c_1 = 13 \wedge c_2 = 25)]$

No-decrement-left = $[(c_1 = 15 \wedge c_2 \neq 23) \vee (c_1 \neq 13 \wedge c_2 = 25)]$

Pour chaque processus, nous pouvons choisir l'invariant global suivant :

$$\tilde{I}(\underline{m}, c_1, c_2, m, p_1, p_2) = [Inv(\underline{m}, c_1, c_2, m, p_1, p_2) \wedge [(\text{Before-test} \wedge m \geq 0) \vee (\text{After-test} \wedge m \geq 1) \vee (\text{After-test-and-decrement} \wedge m \geq 1) \vee (\text{one-decrement-left} \wedge m = 1) \vee (\text{No-decrement-left} \wedge 0 \leq m \leq 1)]]$$

qui satisfait les conditions de vérification suivantes :

- Initialisation

$$[\underline{m} \geq 0] \Rightarrow \tilde{I}(\underline{m}, 11, 21, \underline{m}, p_1, p_2)$$

- Preuve séquentielle du processus 1

$$\tilde{I}(\underline{m}, 11, c_2, m, p_1', p_2) \Rightarrow \tilde{I}(\underline{m}, 12, c_2, m, 1, p_2)$$

$$[\tilde{I}(\underline{m}, 12, c_2, m, p_1, p_2) \wedge m > 1] \Rightarrow \tilde{I}(\underline{m}, 13, c_2, m, p_1, p_2)$$

$$[\tilde{I}(\underline{m}, 12, c_2, m, p_1, p_2) \wedge m \leq 1] \Rightarrow \tilde{I}(\underline{m}, 15, c_2, m, p_1, p_2)$$

$$\tilde{I}(\underline{m}, 13, c_2, m', p_1', p_2) \Rightarrow \tilde{I}(\underline{m}, 14, c_2, m'-1, 2 \times p_1', p_2)$$

$$[\tilde{I}(\underline{m}, 14, c_2, m, p_1, p_2) \wedge m > 1] \Rightarrow \tilde{I}(\underline{m}, 13, c_2, m, p_1, p_2)$$

$$[\tilde{I}(\underline{m}, 14, c_2, m, p_1, p_2) \wedge m \leq 1] \Rightarrow \tilde{I}(\underline{m}, 15, c_2, m, p_1, p_2)$$

- La preuve d'absence d'interférences du processus 2 avec l'invariant global du processus 1 est exactement la preuve séquentielle du processus 2

- Finalisation

$$\tilde{I}(\underline{m}, 15, 25, m, p_1, p_2) \Rightarrow \tilde{I}_1(\underline{m}, m, p_1, p_2)$$

En comparaison avec 4.3.2.4.4-2, l'utilisation d'une décomposition pour fine conduit, pour cet exemple, à une factorisation naturelle de conditions de vérification similaires et donc seulement à 8 conditions de vérification (au lieu de 44 dans 4.3.2.4.4-2). Cet exemple montre que le choix de la bonne décomposition de l'invariant global en invariants locaux dépend du problème à résoudre.

□

Exemple 4.3.2.4.6-3

Nous illustrons la méthode de preuve 4.3.2.4.5-2 sur l'exemple 2.8.2.3. De manière évidente il ne peut s'agir que d'une reformulation de 4.3.2.4.4-2 en utilisant un invariant global par processus plutôt que des invariants locaux associés à chaque point de contrôle du programme. Comme les processus du programme sont symétriques, nous utilisons le même invariant $\tilde{I}$ pour $\tilde{I}_1$ et $\tilde{I}_2$, ce qui permet de n'avoir à raisonner que sur un seul processus.

Pour pouvoir désigner certains états de contrôle du programme, nous introduisons (cf. Cousot-Cousot [84]) :

Not-started = $(c_i = 11 \wedge c_2 = 21)$

Pre_2-started = $(c_1 = 11 \wedge c_2 \neq 21)$

Pre_1-started = $(c_1 \neq 11 \wedge c_2 = 21)$

Started = $(c_1 \neq 11 \wedge c_2 \neq 21)$

L'idée centrale du programme 2.8.2.3 est de maintenir invariante la relation suivante :

$$Inv(\underline{m}, c_1, c_2, m, p_1, p_2) = [(\text{Not-started} \wedge m = \underline{m}) \vee (Pre_2\text{-started} \wedge p_2 = 2^{\underline{m}-m}) \vee (Pre_1\text{-started} \wedge p_1 = 2^{\underline{m}-m}) \vee (\text{started} \wedge p_1 \times p_2 = 2^{\underline{m}-m})]$$

L'autre observation essentielle pour la preuve de correction partielle est que le programme peut se terminer seulement quand $0 \leq m \leq 1$. Pour montrer ceci, nous introduisons :

Before-test = $[(c_1 \in \{11, 12\} \wedge c_2 \in \{21, 22\}) \vee (c_1 = 14 \wedge c_2 = 24)]$

After-test = $[(c_1 = 13 \wedge c_2 \in \{21, 22, 23\}) \vee (c_1 \in \{11, 12, 13\} \wedge c_2 = 23)]$

After-test-and-decrement = $[(c_1 = 14 \wedge c_2 \in \{21, 22, 23\}) \vee (c_1 \in \{11, 12, 13\} \wedge c_2 = 24)]$

one-decrement-left = $[(c_1 = 15 \wedge c_2 = 23) \vee (c_1 = 13 \wedge c_2 = 25)]$

No-decrement-left = $[(c_1 = 15 \wedge c_2 \neq 23) \vee (c_1 \neq 13 \wedge c_2 = 25)]$

Pour chaque processus, nous pouvons choisir l'invariant global suivant :

$$\tilde{I}(\underline{m}, c_1, c_2, m, p_1, p_2) = [Inv(\underline{m}, c_1, c_2, m, p_1, p_2) \wedge [(\text{Before-test} \wedge m \geq 0) \vee (\text{After-test} \wedge m > 1) \vee (\text{After-test-and-decrement} \wedge m \geq 1) \vee (\text{one-decrement-left} \wedge m = 1) \vee (\text{No-decrement-left} \wedge 0 \leq m \leq 1)]]$$

qui satisfait les conditions de vérification suivantes :

- Initialisation

$$[\underline{m} \geq 0] \Rightarrow \tilde{I}(\underline{m}, 11, 21, \underline{m}, p_1, p_2)$$

- Preuve séquentielle du processus 1

$$\tilde{I}(\underline{m}, 11, c_2, m, p_1', p_2) \Rightarrow \tilde{I}(\underline{m}, 12, c_2, m, 1, p_2)$$

$$[\tilde{I}(\underline{m}, 12, c_2, m, p_1, p_2) \wedge m > 1] \Rightarrow \tilde{I}(\underline{m}, 13, c_2, m, p_1, p_2)$$

$$[\tilde{I}(\underline{m}, 12, c_2, m, p_1, p_2) \wedge m \leq 1] \Rightarrow \tilde{I}(\underline{m}, 15, c_2, m, p_1, p_2)$$

$$\tilde{I}(\underline{m}, 13, c_2, m', p_2', p_2) \Rightarrow \tilde{I}(\underline{m}, 14, c_2, m'-1, 2 \times p_2', p_2)$$

$$[\tilde{I}(\underline{m}, 14, c_2, m, p_1, p_2) \wedge m > 1] \Rightarrow \tilde{I}(\underline{m}, 13, c_2, m, p_1, p_2)$$

$$[\tilde{I}(\underline{m}, 14, c_2, m, p_1, p_2) \wedge m \leq 1] \Rightarrow \tilde{I}(\underline{m}, 15, c_2, m, p_1, p_2)$$

- La preuve d'absence d'interférences du processus 2 avec l'invariant global du processus 1 est exactement la preuve séquentielle du processus 2

- Finalisation

$$\tilde{I}(\underline{m}, 15, 25, m, p_1, p_2) \Rightarrow \tilde{I}_1(m, m, p_1, p_2)$$

En comparaison avec 4.3.2.4.4-2, l'utilisation d'une décomposition moins fine conduit, pour cet exemple, à une factorisation naturelle des conditions de vérification similaires et donc seulement à 8 conditions de vérification (au lieu de 41 dans 4.3.2.4.4-2). Cet exemple montre que le choix de la bonne décomposition de l'invariant global en invariants locaux dépend du problème à résoudre. □

4.3.2.4.7 Utilisation d'un invariant global et d'invariants locaux

Un certain nombre de méthodes de preuve de propriétés d'invariance des programmes utilisent un invariant global sur l'état mémoire (qui s'appelle le "resource invariant" chez Hoare[72], Owicki-Gries[76b], le "monitor invariant" chez Howard[76], le "global invariant" chez Apt-Francez-deRoever[30], etc.) en même temps que des invariants locaux associés à divers points de contrôle du programme et portant sur l'état mémoire et l'état de contrôle (ou bien sur l'état mémoire uniquement en utilisant des variables auxiliaires pour simuler l'état de contrôle).

Si $S = (C_0 \times \dots \times C_{n-1}) \times M$ et $A_S = (S \to \{tt, ff\})$, cette décomposition se définit comme suit :

$\tilde{A_S} = \tilde{A_S g} \times \tilde{A_S l}$

où $\tilde{A_S g} = (M \to \{tt, ff\})$

$\tilde{A_S l} = (\tilde{A_{S_0}} \times \dots \times \tilde{A_{S_{n-1}}})$

$\tilde{A_{S_i}} = (C_i \to (C_0 \times \dots \times C_{i-1} \times C_{i+1} \times \dots \times C_{n-1} \times M \to \{tt, ff\}))$, $i \in m$

$\alpha \in (A_S \to \tilde{A_S})$

$\alpha(I) = \langle \tilde{I}_g, \tilde{I}_l \rangle$

où $\tilde{I}_g(m) = (\exists c_0 \in C_0, \dots, c_{n-1} \in C_{n-1} \cdot I(\langle\langle c_0, \dots, c_{n-1} \rangle, m\rangle))$

$\tilde{I}_{l_{i,c_i}}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{n-1}, m) = I(\langle\langle c_0, \dots, c_i, \dots, c_{n-1} \rangle, m\rangle)$

$\gamma \in (\tilde{A_S} \to A_S)$

$\gamma(\langle \tilde{I}_g, \tilde{I}_l \rangle)(\langle\langle c_0, \dots, c_{n-1} \rangle, m\rangle) = [\tilde{I}_g(m) \wedge \forall i \in m, c \in C_i \cdot \tilde{I}_{l_{i,c}}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{n-1}, m)]$

Cette décomposition est évidemment sémantiquement complète puisque l'invariant global $\tilde{G}$ est redondant et les invariants locaux $\tilde{I}$ peuvent être définis comme en [illegible]

Pour utiliser les avantages des décompositions 4.3.2.4.4 et 4.3.2.4.6 on peut envisager de faire porter l'invariant global $\tilde{G}$ sur l'état mémoire mais également sur l'état de contrôle, ce qui donne :

$\tilde{A_S g} = (C_0 \times \dots \times C_{n-1} \times M \to \{tt, ff\})$

$\tilde{G}(c_0, \dots, c_{n-1}, m) = I(\langle\langle c_0, \dots, c_{n-1} \rangle, m\rangle)$

Dans le cas de programmes parallèles comportant des variables globales qui peuvent être modifiées par tous les processus et des variables locales qui ne sont visibles que dans le processus où elles sont déclarées, on peut imaginer de faire porter l'invariant global sur les variables globales et l'état de contrôle et de faire porter l'invariant local sur les variables locales visibles, sur les variables globales et sur l'état de contrôle. On a alors :

$S = (C_0 \times \dots \times C_{n-1}) \times M \times (M'_0 \times \dots \times M'_{n-1})$

$A_S = (S \to \{tt, ff\})$

$\tilde{A_S} = \tilde{A_S g} \times \tilde{A_S l}$

où $\tilde{A_S g} = (C_0 \times \dots \times C_{n-1} \times M \to \{tt, ff\})$

$\tilde{A_S l} = (\tilde{A_{S_0}} \times \dots \times \tilde{A_{S_{n-1}}})$

$\tilde{A_{S_i}} = (C_i \to (C_0 \times \dots \times C_{i-1} \times C_{i+1} \times \dots \times C_{n-1} \times M \times M'_i \to \{tt, ff\}))$, $i \in m$

$\alpha \in (A_S \to \tilde{A_S})$

$\alpha(I) = \langle \tilde{I}_g, \tilde{I}_l \rangle$

où $\tilde{I}_g(c_0, \dots, c_{n-1}, m) = [\exists m'_0 \in M'_0, \dots, m'_{n-1} \in M'_{n-1} \cdot I(\langle\langle c_0, \dots, c_{n-1} \rangle, m, \langle m'_0, \dots, m'_{n-1} \rangle\rangle)]$

$\tilde{I}_{l_{i,c_i}}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{n-1}, m, m') = [\exists m'_0 \in M'_0, \dots, m'_{i-1} \in M'_{i-1}, m'_{i+1} \in M'_{i+1}, \dots, m'_{n-1} \in M'_{n-1} \cdot$
$I(\langle\langle c_0, \dots, c_{n-1} \rangle, m, \langle m'_0, \dots, m'_{i-1}, m', m'_{i+1}, \dots, m'_n \rangle\rangle)]$

$\gamma \in (\tilde{A_S} \to A_S)$

$\gamma(\langle \tilde{I}_g, \tilde{I}_l \rangle)(\langle\langle c_0, \dots, c_{n-1} \rangle, m, \langle m'_0, \dots, m'_{n-1} \rangle\rangle) =$

$[\tilde{I}_g(c_0, \dots, c_{n-1}, m) \wedge \forall i \in m \cdot \tilde{I}_{l_{i,c_i}}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{n-1}, m, m'_i)]$

On pourrait croire que l'utilisation de variables locales au lieu de variables globales permet de localiser les invariants en ce sens que l'invariant associé à un point de programme ne porte que sur les variables du programme qui sont visibles en ce point du programme. Cette approche n'est pas complète, comme le montre l'exemple suivant :

```
var M : integer ;
    F : boolean ;
    S1 : sémaphore :=1 ;
    S2 : sémaphore :=0 ;
M:=0 ; F:= false ;
[[ var T1 : integer ;
   T1:=0 ;
   while T1<10 do
       P(S1) ;
       T1:=T1+1 ;
       V(S2) ;
   od ;
   P(S1) ;
   F:= true ;
   V(S2) ;
   < M:=M+T1 > ;
||
   var T2 : integer ;
   T2:=0 ;
   P(S2) ;
   while not F do
       T2:=T2-1 ;
       V(S1) ;
       P(S2) ;
   od ;
```


Pour démontrer que la valeur finale de M est nulle, il faut évidemment pouvoir établir une relation entre les valeurs finales de T1 et T2 et donc une relation entre les valeurs courantes de T1 et T2 ce qui est impossible avec la décomposition choisie. (Cet argument peut être formalisé à l'aide de points fixes comme en 4.3.2.4.5).

Exemple 4.3.2.4.7-1

La méthode de Levin [79] pour CSP (Hoare [78]) utilise une décomposition similaire mais qui ne porte pas sur l'état de contrôle, les invariants locaux portant sur les variables des processus de CSP et des variables auxiliaires et l'invariant global ne portant que sur des variables auxiliaires. La complétude sémantique de la méthode vient de l'utilisation des variables auxiliaires qui permettent de simuler les relations entre les états de contrôle des processus (comme en 4.3.2.4.5) mais également d'exprimer indirectement les relations entre variables locales des processus à l'aide de variables auxiliaires globales (la technique consistant à recopier les variables locales dans les variables auxiliaires globales après chaque modification des variables locales).

□

4.3.2.4.8 Classification des méthodes de preuve d'invariance selon la finesse de la décomposition de l'invariant global en invariants locaux

Les méthodes de preuve de programmes peuvent être classées selon la classe des propriétés qu'elles permettent de démontrer (invariance conditionnelle, invariance, fatalité, ...).

Les diverses méthodes pour démontrer des propriétés de programmes dans une même classe peuvent être classées selon le principe d'induction dont elles dérivent.

Dans le cas de propriétés d'invariance, les méthodes de preuve dérivant d'un même principe d'induction de la forme :

$[\exists I \in As.\ Co(I)]$

sont de la forme :

$[\exists \tilde{I} \in \tilde{As}.\ \tilde{Co}(\tilde{I})]$

et s'obtiennent au moyen d'une correspondance (α, γ) entre As et $\tilde{As}$ telle que $\tilde{Co} \Rightarrow Co \circ \gamma$ (condition suffisante de correction, et eventuellement $Co \Rightarrow \tilde{Co} \circ \alpha$ (condition suffisante de complétude)).
Nous dirons qu'une méthode M1 de preuve basée sur une décomposition $(\tilde{As}_1, (\alpha_1, \gamma_1))$ de As peut se dériver d'une méthode M2 de preuve basée sur une décomposition $(\tilde{As}_2, (\alpha_2, \gamma_2))$ de As, quand il existe une décomposition $(\tilde{As}, (\alpha, \gamma))$ de $\tilde{As}_2$ telle que $\alpha_1 = \alpha \circ \alpha_2$ et $\gamma_1 = \gamma_2 \circ \gamma$ (auquel cas nous écrirons $M_1 \leftarrow M_2$).

Exemple 4.3.2.4.8-1

Si on a démontré une propriété d'invariance pour un programme $[[Pra_0 \| \dots \| Pra_{m-1}] I]$ en utilisant la méthode 4.3.2.4.6 c'est-à-dire en utilisant des invariants $G_0, \dots, G_{m-1}$ sur l'état de contrôle et des variables auxiliaires

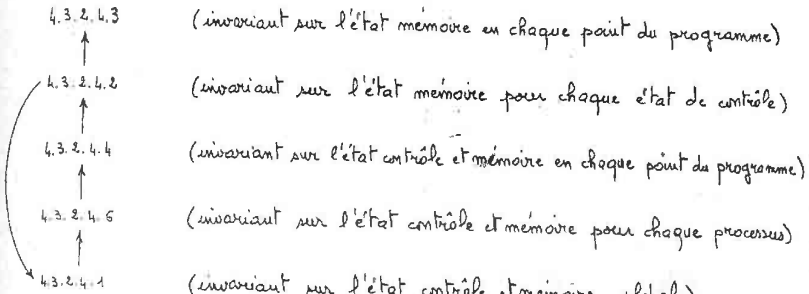
chaque processus $Pra_0, \dots, Pra_{m-1}$ du programme, on peut reformuler cette preuve pour qu'elle corresponde à la méthode 4.3.2.4.4 en utilisant des invariants $I_{i,c}$, $i \in m$, $c \in C_i$ définis par $I = \chi(G)$ tel que

$$I_{i,c}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{m-1}, m) = G_i(c_0, \dots, c_{i-1}, c, c_{i+1}, \dots, c_{m-1}, m)$$

Réciproquement, une preuve utilisant un invariant global par processus (cf. 4.3.2.4.6) peut se dériver d'une preuve utilisant des invariants locaux associés à chaque point du programme (cf. 4.3.2.4.4) en utilisant $G = \gamma(I)$ tel que

$$G_i(c_0, \dots, c_{m-1}, m) = \bigvee_{c \in C_i} I_{i,c}(c_0, \dots, c_{i-1}, c_{i+1}, \dots, c_{m-1}, m)$$

Plus généralement, les méthodes de preuve considérées au paragraphe 4.3.2.4 peuvent se dériver les unes des autres comme suit :



La relation de dérivation est un préordre (transitive et réflexive). Nous disons que deux méthodes de preuve (basées sur un même principe d'induction) sont "sémantiquement équivalentes" quand chacune se dérive de l'autre. La relation de dérivation induite sur l'ensemble des méthodes de preuve quotienté par la relation d'équivalence sémantique est un ordre partiel. Le supremum est le principe d'induction et l'infimum au choix $\tilde{As}_{\perp} = 1$, $\chi(P) = 0$, $\gamma(0) = \Pi$, [illegible] (qui permet de prouver [illegible]). On prouve que cet ensemble ordonné contient le treillis complet des méthodes de preuve d'invariance des programmes considéré dans Cousot-Cousot [79a].

4.3.2.5 Analyse sémantique des programmes

L'analyse sémantique d'un programme (Cousot P. [77, 78, 81], Cousot-Cousot [76, 77a, 77b, 77c, 77d, 79a, 80a, 84]) est une analyse statique (i.e. sans exécuter le programme) des propriétés dynamiques (i.e à l'exécution) de ce programme.

4.3.2.5.1 Analyse d'invariance "en avant"

Soit $\langle S', A', \Sigma' \rangle$ la sémantique d'un programme. L'analyse sémantique d'invariance "en avant" consiste à caractériser la relation $\underline{D}'(\phi, \langle S', A', \Sigma' \rangle)$ entre les états $\underline{s}$ satisfaisant une condition ϕ et leurs descendants possibles $\bar{s}$:

$$\underline{D}'(\phi, \langle S', A', \Sigma' \rangle) \in (S^2 \to \{tt, ff\})$$
$$\underline{D}'(\phi, \langle S', A', \Sigma' \rangle)(\underline{s}, \bar{s}) = [\exists p \in \Sigma', i, j \in |p| . (\phi(\underline{s}) \wedge p_i = \underline{s} \wedge i \leq j \wedge p_j = \bar{s})]$$

Il est toujours possible de se ramener au cas où la condition ϕ ne porte que sur les états initiaux en considérant la sémantique $\langle S, A, \Sigma \rangle$ telle que $S = S'$, $A = A'$ et

$$\Sigma = \{ p \in \Sigma \langle S, A \rangle : \phi(p_0) \wedge \exists q \in \Sigma' . p \to q \}$$

et en posant :

$$\underline{D}(\phi, \langle S, A, \Sigma \rangle) \in (S^2 \to \{tt, ff\})$$
$$\underline{D}(\phi, \langle S, A, \Sigma \rangle)(\underline{s}, \bar{s}) = [\exists p \in \Sigma, i \in |p| . (\phi(p_0) \wedge p_0 = \underline{s} \wedge p_i = \bar{s})]$$

on a :

$$\underline{D}'(\phi, \langle S', A', \Sigma' \rangle) = \underline{D}(\phi, \langle S, A, \Sigma \rangle)$$

La relation $\underline{D}(\phi, \langle S, A, \Sigma \rangle)$ n'étant pas calculable dans tous les cas, on va chercher à [illegible] par une relation $\underline{\tilde{D}}(\phi, \langle S, A, \Sigma \rangle)$ définie comme $\underline{D}(\phi, \langle S, A, \Sigma \rangle) \Rightarrow \underline{\tilde{D}}(\phi, \langle S, A, \Sigma \rangle)$, l'un ou l'autre, ce qui est fonction par dualité.

Soit $\langle S, A, t, \varepsilon \rangle$ le système de transition engendré par $\langle S, A, \Sigma \rangle$. On a $\underline{D}(\phi, \langle S, A, \Sigma \rangle) \Rightarrow \underline{D}(\phi, \langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle)$ (en effet $D(\phi, \langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle)$ est invariante pour $\langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle$ et donc pour $\langle S, A, \Sigma \rangle$ d'après 4.1.3.3). Une première approximation supérieure consiste donc à raisonner sur le système de transition $\langle S, A, t, \varepsilon \rangle$ de sorte qu'en posant :

$$\underline{D}(\langle S, A, t, \varepsilon \rangle) \in (S^2 \to \{tt, ff\})$$
$$\underline{D}(\langle S, A, t, \varepsilon \rangle)(\underline{s}, \bar{s}) = [\varepsilon(\underline{s}) \wedge t^*(\underline{s}, \bar{s})]$$

il s'agit de caractériser $\underline{D}(\langle S, A, t, \varepsilon \wedge \phi \rangle)$.

On remarque alors comme en 4.3.1.7-1 que $\underline{D}(\langle S, A, t, \varepsilon \rangle)$ est le plus petit point fixe $\underset{\Rightarrow}{lfp}(F)$ pour $\Rightarrow$ de l'opérateur F sur $A_{\underline{D}} = (S^2 \to \{tt, ff\})$ défini par :

$$F(I)(\underline{s}, \bar{s}) = [(\varepsilon(\underline{s}) \wedge \underline{s} = \bar{s}) \vee (\exists s' \in S, a \in A . I(\underline{s}, s') \wedge t_a(s', \bar{s}))]$$

de sorte qu'en utilisant une décomposition (α, γ) des invariants globaux de $\langle A_{\underline{D}}, \Rightarrow \rangle$ en des invariants locaux de $\langle A\tilde{s}, \Xi \rangle$ et un opérateur correspondant $\tilde{F}$ monotone sur $A\tilde{s}$ tel que $\alpha \circ F \circ \gamma \Xi \tilde{F}$, on a $\underset{\Rightarrow}{lfp}(F) \Rightarrow \gamma(\underset{\Xi}{lfp}(\tilde{F}))$ quand (α, γ) est une (demi-) correspondance de Galois et $\langle A\tilde{s}, \Xi \rangle$ un treillis complet.

Si le treillis complet $\langle A\tilde{s}, \Xi \rangle$ satisfait la condition de chaine ascendante (toute chaine strictement croissante pour $\sqsubset$ est finie) alors $\underset{\Xi}{lfp}(\tilde{F})$ est calculable itérativement comme $\bigsqcup_{n \geq 0} \tilde{F}^n(\alpha(ff))$.

Si l'itération $X^0 = \alpha(\underline{ff}), \ldots, X^{n+1} = \tilde{F}(X^n)$ peut ne pas converger en un nombre fini de pas, on utilisera une technique d'extrapolation de la plus petite solution $\underset{\Xi}{lfp}(\tilde{F})$ du système d'équations $X = \tilde{F}(X)$ en calculant la limite $\tilde{I} = \bigsqcup_{n \geq 0} \tilde{F}^{\nabla n}(\alpha(\underline{ff}))$ où $\tilde{F}^{\nabla}(X) = X \nabla \tilde{F}(X)$, d'une itération croissante avec rétrécissement ∇ satisfaisant :

$$\forall X, Y \in A\tilde{s} . [(X \sqcup Y) \Xi (X \nabla Y)]$$

[illegible] avec $\underset{\Xi}{lfp}(\tilde{F}) \Xi \tilde{I}$ et la condition sur ∇ n'interdit pas à la séquence d'itérations d'être stationnaire de la forme $X^0, \ldots, X^{n+1} = X^n \nabla \tilde{F}(X^n), \ldots$ (ce qui [illegible] la convergence).

Si $\tilde{I}$ n'est pas un point fixe de $\tilde{F}$, alors $\mathrm{lfp}^{\tilde{\sqsubseteq}}(\tilde{F}) \tilde{\sqsubseteq} \tilde{F}(\tilde{I}) \tilde{\sqsubseteq} \tilde{I}$ et l'approximation supérieure $\tilde{I}$ de $\mathrm{lfp}^{\tilde{\sqsubseteq}}(\tilde{F})$ peut être améliorée puisque $\mathrm{lfp}^{\tilde{\sqsubseteq}}(\tilde{F}) \tilde{\sqsubseteq} \tilde{F}^n(\tilde{I})$ pour tout $n>0$. De manière plus générale il est ensuite possible d'améliorer l'approximation $\tilde{I}$ de $\mathrm{lfp}^{\tilde{\sqsubseteq}}(\tilde{F})$ en calculant la limite $\tilde{J} = \prod_{n>0} \tilde{F}^{\Delta n}(\tilde{I})$ où $\tilde{F}^{\Delta}(X) = X \Delta \tilde{F}(X)$, d'une itération décroissante avec rétrécissement Δ tel que :

$$\forall X, Y \in \tilde{A^\Delta}.\ [(X \sqcap Y) \tilde{\sqsubseteq} (X \Delta Y) \tilde{\sqsubseteq} Y]$$

(de sorte que $\mathrm{lfp}^{\tilde{\sqsubseteq}}(\tilde{F}) \tilde{\sqsubseteq} \tilde{J}$) et toute chaine strictement décroissante de la forme $X^0, \ldots, X^{i+1} = X^i \Delta \tilde{F}(X^i), \ldots$ est finie (de sorte que l'itération converge).

En pratique, la décomposition (α, γ) est choisie sous la forme $(\alpha_q \circ \alpha_p,\ \gamma_p \circ \gamma_q)$ où (α_p, γ_p) est une décomposition utilisée pour une méthode de preuve (cf. par exemple à 4.3.2.4) et (α_q, γ_q) introduit une approximation, qui peut être grossière, déterminée en fonction du problème posé. De ce fait l'équation $X = \tilde{F}(X)$ se précise généralement sous la forme d'un système d'équations :

$$\begin{cases} X_i = \tilde{F}_i(X_0, \ldots, X_{n-1}) \\ i \in n \end{cases}$$

Le calcul de $\bigcup_{n>0} \tilde{F}^n(\alpha(\bot))$ peut alors se faire en utilisant toute stratégie chaotique voire asynchrone équivalente (Cousot-P [77]). En particulier, étant donné le graphe de dépendance du système d'équations (i dépend de j si le résultat de $F_i(X_0, \ldots, X_{n-1})$ dépend de X_j), il peut être avantageux d'itérer en faisant localement converger les composantes X_i correspondant à une même composante fortement connexe du graphe de dépendance et ce d'une part en tenant compte des composantes fortement connexes en cascade et d'autre part en procédant localement de manière identique à l'extérieur de chacune

composante fortement connexe. Dans le cas d'itérations utilisant une extrapolation (élargissement ou rétrécissement), il suffit d'utiliser cette extrapolation pour les composantes X_i telles que i est un point de coupure du graphe de dépendance (les points de coupure étant choisis de sorte que tout cycle dans le graphe de dépendance passe par un point de coupure).

Exemple 4.3.2.5.1-1

Pour étendre Cousot-Cousot [76, 77a] au cas des programmes parallèles de la forme :

$$[\![P_{\ell_0} \| \ldots \| P_{\ell_{n-1}}]\!]$$

où

$S = (C_0 \times \ldots \times C_{n-1}) \times (\mathcal{V} \to \mathcal{D})$ et

$\mathcal{D} = \{ z \in \mathbb{Z} : k_i \leq z \leq k_i \}$

on peut utiliser la décomposition :

$A^\Delta = (S \to \{\alpha, \mathit{ff}\})$

$\tilde{A^\Delta} = \prod_{i \in n} \prod_{c \in C_i} (\mathcal{V} \to \mathbb{Z} \times \mathbb{Z})$

où

$$\alpha(I)_{i,c}[v] = \langle \underline{\inf}\{v \in \mathbb{Z} : P(v)\},\ \underline{\sup}\{v \in \mathbb{Z} : P(v)\} \rangle \quad \text{si } \exists v.\ P(v)$$
$$= \bot \quad \text{si } \forall v. \neg P(v)$$

et

$$P(v) = [\exists c_0 \in C_0, \ldots, c_{i-1} \in C_{i-1}, c_{i+1} \in C_{i+1}, \ldots, c_{n-1} \in C_{n-1}, m \in (\mathcal{V} \to \mathcal{D}).$$
$$I(\langle\langle c_0, \ldots, c_{i-1}, c, c_{i+1}, \ldots, c_{n-1} \rangle, m[\mathcal{V} \leftarrow v]\rangle)]$$

Cette décomposition ne permet pas d'exprimer de relations entre compteurs ordinaux des différents processus. On obtiendra donc des résultats plus précis en utilisant la décomposition suivante :

$$\prod_{i \in n} \prod_{c \in C_i} (\ldots \times (C_{n-1} \to \mathcal{V} \to \mathbb{Z} \times \mathbb{Z}))$$

où $\alpha(I)_{i,c}[c_0,\dots,c_{i-1},c_{i+1},\dots,c_{n-1},V] = \langle \underline{\inf}\{v\in\mathbb{Z} : Q(v)\}, \underline{\sup}\{v\in\mathbb{Z} : Q(v)\}\rangle$ si $\exists v.\ Q(v)$

$= \bot$ si $\forall v.\neg Q(v)$

et $Q(v) = [\exists\, m \in (\mathbb{V}\to\mathbb{Z}).\ I(\langle\langle c_0,\dots,c_{i-1},c,c_{i+1},\dots,c_{n-1}\rangle, m[V\leftarrow v]\rangle)]$

Par exemple pour le programme 2.3.2.3 :

```
0: {m≥0}
   ||
   11:
       P1 := 1 ;
   12:
       while N>1 do
   13:
           ⌊ N:=N-1 ; P1:=2×P1 ⌋ ;
   14:
       od ;
   15:
   ||
   21:
       P2:=1 ;
   22:
       while N>1 do
   23:
           ⌊ N:=N-1 ; P2:=2×P2 ⌋ ;
   24:
       od ;
   25:
   ||;
1:
   if N=0 then P:=P1×P2 else P:=2×P1×P2 fi ;
2:
```

L'invariant local approché D_{jl} attaché au point l du processus $j=1,2$ est choisi comme étant un intervalle de valeurs pour chaque variable $v\in\mathbb{V}$ et chaque point de contrôle $l\in C_j$ de l'autre processus ($\tilde{j}$ où $\tilde{1}=2$ et $\tilde{2}=1$). $D_{jl}(\tilde{j}l)$ est donc un triplet $\langle m, P_j, P_{\tilde{j}}\rangle$ des valeurs abstraites des variables N, P_j, $P_{\tilde{j}}$ où chaque m, P_j, $P_{\tilde{j}}$ est $\bot$ (qui correspond à $\emptyset$) ou un intervalle de valeurs entières [illegible]

Nous utiliserons les notations suivantes :

- $\langle m, p_j, p_{\tilde{j}}\rangle[N] = m$, $\langle m, p_j, p_{\tilde{j}}\rangle[P_j] = p_j$, $\langle m, p_j, p_{\tilde{j}}\rangle[P_{\tilde{j}}] = p_{\tilde{j}}$

- $\langle m, p_1, p_2\rangle[p_1/p_1'] = \langle m, p_1', p_2\rangle$ (substitution)
  $\langle m, p_1, p_2\rangle[p_2/p_2'] = \langle m, p_1, p_2'\rangle$

- $\bot \wedge x = x \wedge \bot = \bot$ si $x \in (\{\bot\}\cup\{[a,b] : a\le b\})$ (conjonction approchée)
  $[a,b]\wedge[c,d] = [\underline{\sup}(a,c), \underline{\inf}(b,d)]$
  ssi $\underline{\sup}(a,c) \le \underline{\inf}(b,d)$
  $= \bot$ ssi $(b<c)$ ou $(d<a)$

- $\bot \vee x = x \vee \bot = x$ si $x \in (\{\bot\}\cup\{[a,b] : a\le b\})$ (disjonction approchée)
  $[a,b]\vee[c,d] = [\underline{\inf}(a,c), \underline{\sup}(b,d)]$

- $\bot - 1 = \bot$
  $[a,b] - 1 = [a-1, b-1]\wedge[\ell_i, h_i]$ (décrémentation approchée)

- $2\times\bot = \bot$
  $2\times[a,b] = [2\times a, 2\times b]\wedge[\ell_i, h_i]$ (décalage gauche approché)

Dans le système d'équations approchées de point fixe suivant nous supposons qu'initialement nous avons $m\ge 0$. Pour chaque équation nous distinguons un terme correspondant à la preuve séquentielle et un terme correspondant au contrôle d'absence d'interférence :

$D_0 = \langle[0,h_i],[\ell_i,h_i],[\ell_i,h_i]\rangle$

$D_{11}(2\ell) = D_0$

$D_{11}(2k) = inter_{11}(2k)$ $k=2,\dots,5$

$D_{12}(2k) = D_{11}(2k)[P_1/[1,1]] \vee inter_{12}(2k)$ $k=1,\dots,5$

$D_{13}(2k) = (D_{12}(2k)\wedge\langle[2,h_i],[2,h_i],[\ell_i,h_i]\rangle)\vee(D_{12}(2k)\wedge\langle[2,h_i],[2,h_i],[\ell_i,h_i]\rangle)\vee inter_{13}(2k)$ $k=1,\dots,5$

$D_{21}(2\ell) = \langle[1,1],[0,h_i],[\ell_i,h_i]\rangle$

$D_{21}(2k) = (D_{21}(2k)\wedge\langle[2,h_i],[\ell_i,h_i],[\ell_i,h_i]\rangle)\vee(D_{21}(2k)\wedge\langle[2,h_i],[\ell_i,h_i],[\ell_i,h_i]\rangle)\vee inter_{21}(2k)$ $k=1,\dots,5$

$D_{22}(2k) = (D_{21}(2k)\wedge\langle[2,h_i],[2,h_i],[\ell_i,h_i]\rangle)\vee(D_{22}(2k)\wedge\langle[2,h_i],[2,h_i],[\ell_i,h_i]\rangle)\vee$
$inter_{22}(2k)$ $k=1,\dots,5$

où

$$inter_{1R}(21) = \langle \bot, \bot, \bot \rangle$$

$$inter_{1R}(22) = (D_{1R}(21) \wedge D_{24}(1\check{R})[p2/[1,1]])$$

$$inter_{1R}(23) = (D_{1R}(22) \wedge D_{23}(1\check{R}) \wedge \langle [2,ri], [li,ri], [li,ri] \rangle) \vee (D_{1R}(24) \wedge D_{24}(1\check{R}) \wedge \langle [2,ri], [li,ri], [li,ri] \rangle)$$

$$inter_{1R}(24) = \langle (D_{1R}(23)[N] \wedge D_{23}(1\check{R})[N]) - 1,\ D_{1R}(23)[P1] \wedge D_{23}(1\check{R})[P1],\ 2\times (D_{1R}(23)[P2] \wedge D_{23}(1\check{R})[P2]) \rangle$$

$$inter_{1R}(25) = (D_{1R}(22) \wedge D_{22}(1\check{R}) \wedge \langle [li,1], [li,ri], [li,ri] \rangle) \vee (D_{1R}(24) \wedge D_{24}(1\check{R}) \wedge \langle [li,1], [li,ri], [li,ri] \rangle)$$

... équations similaires pour le processus 2 ...

$$D_1 = D_{15}(25) \wedge D_{25}(15)$$

Ce système d'équations peut être résolu au moyen d'une stratégie itérative asynchrone (Cousot-P [77]). Initiallement on pose $D_{ie}^{(\vec{l})=\bar{\perp}}$ pour $i=1,2$, $l \in C_i$, $\vec{l} \in C_{\bar{i}}^{\vec{i}}$. Puis on itère appliquant n'importe quelle équation du système d'équations jusqu'à stabilisation.

La convergence peut être accelerée utilisant les techniques d'extrapolation décrites dans Cousot-Cousot [76, 77]. Ceci consiste à définir un opérateur d'élargissement ∇ tel que :

$$\bot \nabla x = x$$

$$[a,b] \nabla [c,d] = [\underline{if}\ c<a\ \underline{then}\ li\ \underline{else}\ a\ ,\ \underline{if}\ d>b\ \underline{then}\ hi\ \underline{else}\ b]$$

et remplacer les équations D_{j2}, $j=1,2$ par

$$D_{j2}(\vec{j}^{\vec{i}}2) = D_{j2}(\vec{j}^{\vec{i}}2) \nabla \bar{\perp} (D_{j2}(\vec{j}^{\vec{i}}2) \wedge \langle [2,ri], [2i,fi], [li,ri] \rangle) \vee (D_{j4}(\vec{j}^{\vec{i}}2) \wedge \langle [2,ri], [li,fi], [2i,2i] \rangle) \vee inter_{j2}(\vec{j}^{\vec{i}}2)$$

pour résoudre itérativement. Le résultat que nous obtenons (pour le processus 1) est :

	$k=1$			$k=2$			$k=3$		
	n	p1	p2	n	p1	p2	n	p1	p2
$D_{11}(1\check{2})$	$\langle [0,ri], [li,ri], [li,ri] \rangle$			$\langle [0,ri], [li,ri], [1,1] \rangle$			$\langle [2,ri], [li,ri], [1,ri] \rangle$		
$D_{12}(2\check{2})$	$\langle [0,ri], [1,1], [li,ri] \rangle$			$\langle [0,ri], [1,1], [1,1] \rangle$			$\langle [2,ri], [1,1], [1,ri] \rangle$		
$D_{13}(3\check{2})$	$\langle [2,ri], [1,ri], [li,ri] \rangle$			$\langle [2,ri], [1,ri], [1,1] \rangle$			$\langle [2,ri], [1,ri], [1,ri] \rangle$		
$D_{14}(4\check{2})$	$\langle [1,ri-1], [2,ri], [li,ri] \rangle$			$\langle [1,ri-1], [2,ri], [1,1] \rangle$			$\langle [1,ri-1], [2,ri], [1,ri] \rangle$		
$D_{15}(5\check{2})$	$\langle [0,1], [1,ri], [li,ri] \rangle$			$\langle [0,1], [1,ri], [1,1] \rangle$			$\langle [1,1], [2,ri], [1,ri] \rangle$		

	$k=4$			$k=5$		
	n	p1	p2	n	p1	p2
$D_{11}(2\check{2})$	$\langle [1,ri-1], [li,ri], [2,ri] \rangle$			$\langle [0,1], [li,ri], [1,ri] \rangle$		
$D_{12}(2\check{2})$	$\langle [1,ri-1], [1,1], [2,ri] \rangle$			$\langle [0,1], [1,1], [1,ri] \rangle$		
$D_{13}(2\check{2})$	$\langle [1,ri-1], [1,ri], [2,ri] \rangle$			$\langle [1,1], [1,ri], [2,ri] \rangle$		
$D_{14}(2\check{2})$	$\langle [0,ri-2], [2,ri], [2,ri] \rangle$			$\langle [0,1], [2,ri], [1,ri] \rangle$		
$D_{15}(2\check{2})$	$\langle [0,1], [1,ri], [2,ri] \rangle$			$\langle [0,1], [1,ri], [1,ri] \rangle$		

Observons que nous obtenons :

$$D_1 = \langle [0,1], [1,ri], [1,ri] \rangle$$

ce qui montre que $0 \le N \le 1$ à la sortie de la commande parallèle du programme, ce qui n'est pas complètement trivial à démontrer à la main.

□

Exemple 4.3.9.5.1-2

Pour étendre Cousot-Halbwachs [78] au cas des programmes parallèles, il suffit de considérer une décomposition de la forme $(\alpha_q \circ \alpha_p, \gamma_p \circ \gamma_q)$ où (α_p, γ_p) décompose $(\mathcal{D}^n \to \{\bot, \top\}^{\vec{n}}, \{\bot, \top\}^{\vec{n}} \to \mathcal{D}^n)$ en $(\alpha_{i \in n} \prod_{c \in C_i^{\vec{n}}} c, \prod_{i \in n} \prod_{c \in C_i^{\vec{n}}} \gamma_{i,c})$ $(\mathcal{D}^n \to \{\bot, \top\}^{\vec{n}}, \{\bot, \top\}^{\vec{n}} \to \mathcal{D}^n)$ tandis que (α_q, γ_q) décompose $(\mathcal{D}^n \to \{\bot, \top\}^{\vec{n}}, \{\bot, \top\}^{\vec{n}} \to \mathcal{D}^n)$ en $(\alpha_{i \in n} \prod_{c \in C_i^{\vec{n}}} c, \prod_{i \in n} \prod_{c \in C_i^{\vec{n}}} \gamma_{i,c})$ $(\mathcal{D}^n \to \{\bot, \top\}^{\vec{n}}, \{\bot, \top\}^{\vec{n}} \to \mathcal{D}^n)$ tandis que (α_q, γ_q) consiste pour chaque composante $i \in n$ à appliquer $\alpha_{i,c}$ de $(\mathcal{D}^n \to \{\bot, \top\}^{\vec{n}}, \{\bot, \top\}^{\vec{n}} \to \mathcal{D}^n)$ pour le produit et à considérer l'ensemble des $\{x : P_c(x)\}$. Les opérateurs $\sqcup$ et ∇ correspondants ont été proposés [Cousot-Cousot 77]. En particulier si P et Q caractérisent deux polyèdres

convexes de $\mathbb{R}^m$ tels que $P \bowtie Q$ et $P \not\subseteq Q$ alors l'élargissement $P \nabla Q$ de P par Q est obtenu en éliminant du système de contraintes linéaires P toutes les inéquations non satisfaites par Q. Lorsque la dimension du polyèdre caractérisé par P est strictement inférieure à m, on réécrit au préalable le système d'inéquations P sous une forme qui maximise le nombre de contraintes satisfaites par Q.

Par exemple, en choisissant la décomposition (α_p, γ_p) comme en 4.3.2.4.4 ce qui permet d'associer à chaque point du programme une relation linéaire entre l'état mémoire initial et les états de contrôle et mémoire courants, on obtient pour le programme 2.8.3.3.1 les résultats suivants (nous ignorons la variable Any qui ne peut prendre qu'une seule valeur) :

0: {true}

$[\![$ 11: $\{c_1 \geq 21 \wedge c_3 \geq 31 \wedge c_3 \geq c_2 + 9 \wedge c_3 \geq 2c_2 - 14 \wedge 2c_3 \leq c_2 + 46 \wedge c_3 \leq c_2 + 11\}$

while true do

12: $\{c_1 \geq 21 \wedge c_3 \geq 31 \wedge c_3 \geq c_2 + 9 \wedge c_3 \geq 2c_2 - 14 \wedge c_3 \leq 34 \wedge c_3 \leq c_2 + 11\}$

P! any;

14: $\{c_3 = 24 \wedge 21 \leq c_2 \leq 23\}$

V! any;

13: $\{c_2 \geq 21 \wedge c_3 \geq 32 \wedge c_2 \geq 2c_2 - 14 \wedge c_3 \leq 34 \wedge c_3 \leq c_2 + 12\}$

od;

15: {false}

$\|$

21: $\{c_1 \geq 11 \wedge c_3 \geq 31 \wedge c_2 \geq c_1 + 19 \wedge c_3 \geq 2c_1 + 6 \wedge 2c_3 \leq c_1 + 54 \wedge c_3 \leq c_2 + 21\}$

while true do

22: $\{c_1 \geq 11 \wedge c_3 \geq 31 \wedge c_3 \geq c_1 + 19 \wedge c_3 \geq 2c_1 + 6 \wedge c_3 \leq 34 \wedge c_3 \leq c_1 + 21\}$

P! any;

24: $\{c_3 = 34 \wedge 11 \leq c_1 \leq 13\}$

V! any;

23: $\{c_1 \geq 11 \wedge c_3 \geq 32 \wedge c_3 \geq 2c_1 + 6 \wedge c_3 \leq 34 \wedge c_3 \leq c_1 + 32\}$

od;

25: {false}

$\|$

31: $\{11 \leq c_2 \leq 12 \wedge 21 \leq c_2 \leq 22\}$

while true do

32: $\{11 \leq c_1 \leq 13 \wedge 21 \leq c_2 \wedge 23 \wedge c_1 + c_2 \leq 35\}$

P? Any;

34: $\{c_1 \leq 14 \wedge c_2 \leq 24 \wedge c_1 + c_2 \leq 37 \wedge 2c_1 + c_2 \geq 56 \wedge 2c_1 + c_2 \geq 46\}$

V? Any;

33: $\{11 \leq c_1 \leq 13 \wedge 21 \leq c_2 \leq 23 \wedge c_1 + c_2 \geq 33\}$

od;

35: {false}

$]\!]$;

1: {false}

Ces invariants sont suffisants pour démontrer que les points 1 et 21 sont en exclusion mutuelle (puisque si le contrôle pouvait être simultanément en 14 et 24 on aurait $c_1 = 14 \wedge \{c_3 = 24 \wedge 21 \leq c_2 \leq 23\}$ et $c_2 = 24 \wedge \{c_3 = 34 \wedge 11 \leq c_1 \leq 13\}$ ce qui est faux !). Malheureusement le choix [illegible] des points du programme a une influence sur la [illegible] des résultats, comme le montre l'exemple suivant (le [illegible] l'expression de l'invariant suivant : à l'aide [illegible] d'inéquations linéaires) :

0: {true}

[[11: {21 ≤ c2 ≤ 24 ∧ 31 ≤ c3 ≤ 34 ∧ 2c2 - c3 ≥ 10 ∧ 2c3 - c2 ≥ 40}

while true do

12: {21 ≤ c2 ≤ 24 ∧ 31 ≤ c3 ≤ 34 ∧ 2c3 - c2 ≥ 40}

P! any;

13: {c2 ≤ 24 ∧ 32 ≤ c3 ≤ 34 ∧ c3 - c2 ≤ 12 ∧ c3 + c2 ≥ 54}

V! any;

14: {21 ≤ c2 ≤ 24 ∧ 32 ≤ c3 ≤ 24}

od;

15: {false}

||

21: {11 ≤ c1 ≤ 14 ∧ 31 ≤ c3 ≤ 34 ∧ c3 - 2c1 ≤ 10 ∧ 2c3 - c1 ≥ 50}

while true do

22: {11 ≤ c1 ≤ 14 ∧ 31 ≤ c3 ≤ 34 ∧ 2c3 - c1 ≥ 50}

P! any;

23: {c1 ≤ 14 ∧ 32 ≤ c3 ≤ 34 ∧ c3 - c1 ≤ 22 ∧ c3 + c1 ≥ 44}

V! any;

24: {11 ≤ c1 ≤ 14 ∧ 32 ≤ c3 ≤ 24}

od;

25: {false}

||

31: {11 ≤ c1 ≤ 12 ∧ 21 ≤ c2 ≤ 22}

while true do

32: {11 ≤ c1 ≤ 14 ∧ 21 ≤ c2 ≤ 24}

P? Amy;

33: {11 ≤ c1 ≤ 14 ∧ 21 ≤ c2 ≤ 24 ∧ c1 + c2 ≤ 33 ∧ c1 - c2 ≥ 33}

V? Amy;

34: {11 ≤ c1 ≤ 12 ∧ 21 ≤ c2 ≤ 24 ∧ c1 + c2 ≥ 33}

od;

35: {false}

]];

Ces résultats sont maintenant trop faibles pour démontrer l'exclusion mutuelle des points 13 et 23 du programme (puisque $c_1 = 13 \wedge \{c_2 \leq 24 \wedge 32 \leq c_3 \leq 34 \wedge c_3 - c_2 \leq 12 \wedge c_3 + c_2 \geq 54\} \wedge c_2 = 23 \wedge \{c_1 \leq 14 \wedge 32 \leq c_3 \leq 34 \wedge c_3 - c_1 \leq 22 \wedge c_3 + c_1 \geq 44\}$ n'est pas identiquement faux).

Il est évidemment possible d'éviter cette perte d'information, par exemple en choisissant (α_p, γ_p) comme en 4.3.2.4.2, ce qui revient à associer une relation linéaire entre l'état mémoire initial et courant à toute valeur de l'état de contrôle. Dans ce cas l'analyse donne comme résultat l'assertion true associée aux états de contrôle (11,21,31), (12,21,31), (11,22,31), (11,21,32), (12,22,31), (12,21,32), (11,22,32), (13,21,33), (12,22,32), (11,23,33), (14,21,34), (13,22,33), (12,23,33), (11,24,34), (12,21,34), (14,22,34), (14,21,32), (12,24,34), (11,22,34), (11,24,32), (12,22,34), (14,22,32), (12,24,32), (14,22,33), (13,24,33), (14,24,34), (14,24,32) et faux associé aux autres états de contrôle. (on remarque que la réunion de ces points n'est pas convexe d'où la perte d'information avec la décomposition moins fine ci-dessus). Comme pour l'exemple 4.3.2.5.1-1, le coût de ce gain en précision est qu'il faut maintenant considérer un nombre d'invariants de l'ordre du produit et non plus de la somme des nombres de points de contrôle des processus du programme.
□

4.3.2.5.2 Analyse d'invariance "en arrière"

On peut aussi s'intéresser à une approximation de la relation [illegible] état $\vec{x}$ satisfaisant une condition ψ et leurs successeurs $\vec{x}'$:

$$\{\vec{x}' \mid \exists \vec{x}, \psi(\vec{x}) \wedge \vec{x} \xrightarrow{*} \vec{x}'\} = [\exists p \leq \Sigma, \lambda_i \leq |p|. \ \psi(\vec{x}) \wedge p_0 = \vec{x} \wedge \lambda \leq_{\ell} \wedge p_{|\lambda|} = \vec{x}']$$

ce qui se traite comme dans le cas précédent en appliquant la transformation -1 (cf. 4.2.1.2.3) sur $\underline{Pref}^{\omega}(\langle S,A,\Sigma\rangle)$ et consiste donc à trouver une approximation supérieure de la plus petite solution d'un système d'équations $X=\tilde{B}(X)$ tel que $\alpha\circ B\circ\delta \sqsubseteq \tilde{B}$ et

$$B(I)(\underline{\lambda},\bar{\lambda}) = [(\exists \lambda'\in S, a\in A.\ t_a(\underline{\lambda},\lambda') \wedge I(\lambda',\bar{\lambda})) \vee (\underline{\lambda}=\bar{\lambda} \wedge \psi(\bar{\lambda}))]$$

Exemple 4.3.2.5.2-1

Soit à chercher une condition suffisante $P(\underline{\lambda})$ sur les états d'entrée d'un programme parallèle :

$$Ps \,[\![\, Pr_0 \,\|\, \ldots \,\|\, \alpha L : p \,\|\, \ldots \,\|\, \alpha' L' : p' \,\|\, \ldots \,\|\, Pr_{m-1} \,]\!]\, ; Ps'$$

pour que les points L et L' dans deux processus différents Pr_i et Pr_j soient mutuellement exclusifs. Soit $\langle S,A,\Sigma\rangle$ la sémantique du programme. Posons $\mu\in(S\to\{tt,ff\})$ défini par $\mu(\langle L,M\rangle)=tt$ et $\mu(\langle\langle L_0,\ldots,L_{m-1}\rangle,M\rangle) = \neg(L_i=L \wedge L_j=L')$. Il s'agit de trouver P tel que

$$\forall p\in\Sigma.\ (P(p_0)\Rightarrow \forall i\in p.\ \mu(p_i))$$

Soit $\langle S,A,t,\varepsilon\rangle$ le système de transition engendré par $\langle S,A,\Sigma\rangle$. Il suffit de trouver P tel que :

$$\forall \underline{\lambda}\in S.\ [\varepsilon(\underline{\lambda}) \wedge P(\underline{\lambda})]\Rightarrow[\forall \bar{\lambda}\in S.\ t^*(\underline{\lambda},\bar{\lambda})\Rightarrow\mu(\bar{\lambda})]$$

On peut donc choisir $P(\underline{\lambda})$ tel que :

$$[\varepsilon(\underline{\lambda}) \wedge [\exists\bar{\lambda}\in S.\ t^*(\underline{\lambda},\bar{\lambda})\wedge\neg\mu(\bar{\lambda})]]\Rightarrow\neg P(\underline{\lambda})$$

c'est-à-dire comme la négation d'une condition nécessaire sur les états d'entrée pour qu'ils aient comme descendants un état où le contrôle est mutuellement en L et L'. Cette condition s'obtient comme approximation supérieure du plus petit point fixe de

$$b(I)(\lambda) = [(\exists\lambda'\in S, a\in A.\ t_a(\lambda,\lambda')\wedge I(\lambda'))\vee\neg\mu(\lambda)]$$

□

4.3.2.5.3 Analyse d'invariance "avant-arrière"

L'analyse d'invariance "avant-arrière" consiste à trouver une approximation supérieure de l'ensemble des descendants des états initiaux (satisfaisant une condition initiale ϕ portant par exemple sur les états d'entrée), qui satisfont une condition δ (dérivée par exemple des déclarations) et sont ascendants des états finaux (satisfont une condition finale ψ portant par exemple sur les états de sortie):

$$\underline{DA}\,(\phi,\delta,\psi,\langle S,A,\Sigma\rangle) \in (S^2\to\{tt,ff\})$$

$$\underline{DA}\,(\phi,\delta,\psi,\langle S,A,\Sigma\rangle)(\underline{\lambda},\lambda,\bar{\lambda}) =$$

$$[\exists p\in\Sigma,\ i,j\in|p|.\ \phi(\underline{\lambda})\wedge p_0=\underline{\lambda}\wedge\delta(\lambda)\wedge p_i=\lambda\wedge\psi(\bar{\lambda})\wedge p_j=\bar{\lambda}\wedge i\le j]$$

Une première approximation supérieure consiste à raisonner sur le système de transition $\langle S,A,t,\varepsilon\rangle$ engendré par la sémantique $\langle S,A,\Sigma\rangle$. On a:

$$\underline{DA}\,(\phi,\delta,\psi,\langle S,A,\Sigma\rangle)(\underline{\lambda},\lambda,\bar{\lambda}) \Rightarrow (\underline{lfp}(F)(\underline{\lambda},\lambda)\wedge\delta(\lambda)\wedge\underline{lfp}(B)(\lambda,\bar{\lambda}))$$

où comme précédemment :

$$F(I)(\underline{\lambda},\bar{\lambda}) = [(\underline{\lambda}=\bar{\lambda}\wedge\phi(\underline{\lambda}))\vee(\exists\lambda'\in S, a\in A.\ I(\underline{\lambda},\lambda)\wedge t_a(\lambda,\bar{\lambda}))]$$

$$B(I)(\underline{\lambda},\bar{\lambda}) = [(\exists\lambda'\in S, a\in A.\ t_a(\underline{\lambda},\lambda')\wedge I(\lambda',\bar{\lambda}))\vee(\bar{\lambda}=\underline{\lambda}\wedge\psi(\bar{\lambda}))]$$

Soit $\langle \tilde{A}_\lambda, \sqsubseteq\rangle$ une décomposition de $\langle A_\lambda,\to\rangle$ par la correspondance de Galois (α,γ) où $A_\lambda=(S^2\to\{tt,ff\})$. On pose $\tilde{F}=\alpha\circ F\circ\gamma$, $\tilde{B}=\alpha\circ B\circ\gamma$ et $\tilde{\delta}=\alpha(\delta)$. On définit l'opérateur Π sur $\langle\tilde{A}_\lambda,\sqsubseteq\rangle$ tel que $\forall P,Q\in\tilde{A}_\lambda.\ [\alpha(\gamma(P)\wedge\gamma(Q))\sqsubseteq(P\sqcap Q)]$, Π est un opérateur de rétrécissement comme en 4.2.2.5.1. A la suite de Cousot-P [78], on peut calculer une approximation supérieure de $\underline{DA}$ où $\underline{\tilde{DA}}\,(\underline{\lambda},\lambda,\bar{\lambda}) = [\underline{lfp}(\tilde{F})(\underline{\lambda},\lambda)\wedge\tilde{\delta}(\lambda)\wedge\underline{lfp}(\tilde{B})(\lambda,\bar{\lambda})]$

- Soient $X^0,\ldots,X^k$ une suite décroissante finie de $\tilde{A}_\lambda$ telle que :

$$\tilde{\delta} \sqsubseteq X^0$$

$X^0 \Delta Z^0 = X^1$ où $Z^0 \supseteq \mathrm{lfp}_{\sqsubseteq}(f^0)$ et $f^0(Y) = X^0 \sqcap F(Y)$

$X^1 \Delta Z^1 = X^2$ où $Z^1 \supseteq \mathrm{lfp}_{\sqsubseteq}(f^1)$ et $f^1(Y) = X^1 \sqcap B(Y)$

...

$X^{2k} \Delta Z^{2k} = X^{2k+1}$ où $Z^{2k} \supseteq \mathrm{lfp}_{\sqsubseteq}(f^{2k})$ et $f^{2k}(Y) = X^{2k} \sqcap F(Y)$

$X^{2k+1} \Delta Z^{2k+1} = X^{2k+2}$ où $Z^{2k+1} \supseteq \mathrm{lfp}_{\sqsubseteq}(f^{2k+1})$ et $f^{2k+1}(Y) = X^{2k+1} \sqcap B(Y)$

...

Pour calculer les Z^k, on utilise une itération chaotique croissante avec élargissement puis si la solution obtenue n'est pas un point fixe on l'améliore par une itération chaotique décroissante avec rétrécissement. Bien entendu si le treillis $\langle \mathcal{A}^\sharp, \sqsubseteq \rangle$ satisfait la condition de chaine ascendante (respectivement descendante) on peut choisir $\nabla = \sqcup$ (respectivement $\Delta = \sqcap$).

Exemple 4.3.2.5.3-1

Si nous poursuivons l'analyse en avant du programme 2.8.2.3 donnée en exemple 4.3.2.5.1-1, par une analyse avant-arrière combinée partant de $\delta(m, p1, p2, p) = (m \in [0, Ri] \wedge p1, p2, p \in [li, hi])$ (donnée par les déclarations du programme), nous obtenons les résultats suivants (pour le processus 1, en notant $\div$ la division entière et $\lfloor x \rfloor = (x \div 2) \times 2$)

	$k=1$			$k=2$			$k=3$		
	m	$p1$	$p2$	m	$p1$	$p2$	m	$p1$	$p2$
$D_{11}(2k)$	$\langle [0,Ri], [li,Ri], [li,Ri] \rangle$			$\langle [0,Ri], [li,Ri], [1,1] \rangle$			$\langle [2,Ri], [li,Ri], [li,Ri \div 2] \rangle$		
$D_{12}(2k)$	$\langle [0,Ri], [1,1], [li,Ri] \rangle$			$\langle [0,Ri], [1,1], [1,1] \rangle$			$\langle [2,Ri], [1,1], [li,Ri \div 2] \rangle$		
$D_{13}(2k)$	$\langle [2,Ri], [li,Ri \div 2], [li,Ri] \rangle$			$\langle [2,Ri], [1,Ri \div 2], [1,1] \rangle$			$\langle [2,Ri], [li,Ri \div 2], [li,Ri \div 2] \rangle$		
$D_{14}(2k)$	$\langle [1,Ri-1], [2,\lfloor Ri \rfloor], [li,Ri] \rangle$			$\langle [1,Ri-1], [2,\lfloor Ri \rfloor], [1,1] \rangle$			$\langle [1,Ri-1], [2,\lfloor Ri \rfloor], [li,Ri \div 2] \rangle$		
$D_{15}(2k)$	$\langle [0,1], [1,\lfloor Ri \rfloor], [2,\lfloor Ri \rfloor] \rangle$			$\langle [0,1], [1,\lfloor Ri \rfloor], [1,1] \rangle$			$\langle [0,1], [1,\lfloor Ri \rfloor], [li,Ri \div 2] \rangle$		

	$k=4$			$k=5$		
	m	$p1$	$p2$	m	$p1$	$p2$
$D_{11}(2k)$	$\langle [1,Ri-1], [li,Ri], [li,Ri] \rangle$			$\langle [0,1], [li,Ri], [1,\lfloor Ri \rfloor] \rangle$		
$D_{12}(2k)$	$\langle [1,Ri-1], [1,1], [li,Ri] \rangle$			$\langle [0,1], [1,1], [1,\lfloor Ri \rfloor] \rangle$		
$D_{13}(2k)$	$\langle [1,Ri-1], [1,Ri \div 2], [2,\lfloor Ri \rfloor] \rangle$			$\langle [1,1], [1,Ri \div 2], [2,\lfloor Ri \rfloor] \rangle$		
$D_{14}(2k)$	$\langle [0,Ri-2], [2,\lfloor Ri \rfloor], [2,\lfloor Ri \rfloor] \rangle$			$\langle [0,1], [2,\lfloor Ri \rfloor], [1,\lfloor Ri \rfloor] \rangle$		
$D_{15}(2k)$	$\langle [0,1], [1,\lfloor Ri \rfloor], [2,\lfloor Ri \rfloor] \rangle$			$\langle [0,1], [1,\lfloor Ri \rfloor], [1,\lfloor Ri \rfloor] \rangle$		

On peut associer à chaque point lj du programme l'invariant $\gamma(\bigcup_{k=1}^{n} D_{lj}(2k))$, ce qui donne :

```
0: {m ∈ [0,Ri]}
[[ 11: {m∈[0,Ri] ∧ p1∈[li,Ri] ∧ p2∈[li,Ri]}
       P1:=1;
   12: {m∈[0,Ri] ∧ p1∈[1,1] ∧ p2∈[li,Ri]}
       while N>1 do
   13:     {m∈[1,Ri] ∧ p1∈[1,Ri÷2] ∧ p2∈[li,Ri]}
           ¢ N1:=N-1 ; P1:=2×P1 ¢;
   14:     {m∈[0,Ri-1] ∧ p1∈[2,⌊Ri⌋] ∧ p2∈[li,Ri]}
       od;
   15: {m∈[0,1] ∧ p1∈[1,⌊Ri⌋] ∧ p2∈[li,Ri]}
||
   21: {m∈[0,Ri] ∧ p1∈[li,Ri] ∧ p2∈[li,Ri]}
       P2:=1;
   22: {m∈[0,Ri] ∧ p1∈[1,1] ∧ p2∈[li,Ri]}
       while N>1 do
   23:     {m∈[1,Ri] ∧ p1∈[li,Ri] ∧ p2∈[1,Ri÷2]}
           ¢ N1:=N-1 ; P2:=2×P2 ¢;
   24:     {m∈[0,Ri-1] ∧ p1∈[li,Ri] ∧ p2∈[2,⌊Ri⌋]}
       od;
   25: {m∈[0,1] ∧ p1∈[2,⌊Ri⌋] ∧ p2∈[1,⌊Ri⌋]}
]];
```

1: $\{m \in [0,1] \wedge p1 \in [1, \lfloor hi \rfloor] \wedge p2 \in [1, \lfloor hi \rfloor]\}$

if N=0 then P:= P1×P2 else P:= 2×P1×P2 fi ;

2: $\{m \in [0,1] \wedge p1 \in [1, \lfloor hi \rfloor] \wedge p2 \in [1, \lfloor hi \rfloor] \wedge p \in [1, hi]\}$

Ces résultats permettent de placer des tests qui doivent être vérifiés pour éviter les erreurs à l'exécution (mais en moins grand nombre que ne le ferait un compilateur n'utilisant que les informations données par les déclarations) ainsi que des tests qu'il est nécessaire (mais en général pas suffisant) de vérifier pour que l'exécution du programme se termine sans erreurs à l'exécution et en ne passant que par des états satisfaisant la condition δ (ces derniers ne sont pas placés par un compilateur classique, même pour les tests liés aux déclarations puisque les test à l'exécution sont généralement placés au moment de l'affectation aux variables et pas au moment de leur utilisation). Pour les tests de la première sorte, on trouve :

- $p1 \leq hi \div 2$ au point 13
- $p2 \leq hi \div 2$ au point 23
- $p1 \leq hi \div p2$ au point 1 quand N=0
  $p1 \leq (hi \div 2) \div p2$ au point 1 quand N≠0

(Un compilateur classique placerait des tests inutiles comme $n-1 \geq 0$ aux points 13 et 23 ou des tests plus complexes comme pour tester que $li \leq p1 \times p2 \leq hi$ ou $li \leq 3 \times p1 \times p2 \leq hi$ au point 1 puisque le signe de p1 et p2 n'est pas connu). Pour les tests de la seconde sorte, on trouve :

- $m \geq 0$ au point 0

(ce test [illegible] un compilateur classique mais il serait inutile [illegible] l'entrée de la valeur de N). [illegible] une valeur en dehors [illegible] 1 [illegible] la [illegible] aux [illegible] est mieux


```
const li = ... ;   { plus petit entier, li<0}
      hi = ... ;   { plus grand entier, hi>1000}
type integer = li .. hi ;
var N, K, I, J : integer
    T : array [0..1000] of integer ;
1:  {m,k,i,j ∈ [li;hi]}
    read (N);
2:  {n∈[0,1000] ∧ k,i,j∈[li;hi]}
    K:=0;
3:  {n∈[0,1000] ∧ k∈[0,0] ∧ i,j∈[li;hi]}
    while K <= N do
4:      {n,k ∈[0,1000] ∧ i,j∈[li;hi]}
        read (T[K]);
5:      {n,k ∈[0,1000] ∧ i,j∈[li;hi]}
        K:=K+1;
6:      {n∈[0,1000] ∧ k∈[1,1001] ∧ i,j∈[li;hi]}
    od;
7:  {n∈[0,1000] ∧ k∈[0,1001] ∧ i,j∈[li;hi]}
    I:= N;
8:  {i,j∈[0,1000] ∧ k∈[0,1001] ∧ j∈[li;hi]}
    while I<>0 do
9:      {n∈[0,1000] ∧ i∈[1,1000] ∧ k,j∈[li;hi]}
        J:=0;
10:     {n∈[0,1000] ∧ i∈[1,1000] ∧ j∈[0,0] ∧ k∈[li;hi]}
        while J<>I do
11:         {n∈[0,1000] ∧ i∈[1,1000] ∧ j∈[0,999] ∧ k∈[li;hi]}
            if T[J] <= T[J+1] then
12:             {n∈[0,1000] ∧ i∈[1,1000] ∧ j∈[0,999] ∧ k∈[li;hi]}
                K:= T[J]; T[J]:= T[J+1]; T[J+1]:= K;
13:             {n∈[0,1000] ∧ i∈[1,1000] ∧ j∈[1,999] ∧ k∈[li;hi]}
            fi;
```

14: {m∈[0,1000] ∧ i∈[1,1000] ∧ i∈[0,999] ∧ k∈[li,fi]}

J := J+1;

15: {m∈[0,1000] ∧ i,j∈[1,1000] ∧ k∈[li,fi]}

od;

16: {m∈[0,1000] ∧ i,j∈[1,1000] ∧ k∈[li,fi]}

I := I-1;

17: {m∈[0,1000] ∧ i,j∈[1,1000] ∧ k∈[li,fi]}

od;

18: {m∈[0,1000] ∧ i∈[0,0] ∧ j∈[1,1000] ∧ k∈[li,fi]}

Cette analyse conduit à introduire le test $0 \leq m \leq 1000$ au point 2 (pour garantir que l'exécution se termine sans erreurs à l'exécution, ce test ne serait évidemment pas introduit par un compilateur classique). Dans ces conditions, il faut également introduire le test $j \leq 999$ au point 11 (ce test étant en fait inutile).

□

4.4 REFERENCES

APT K.R., FRANCEZ N., DE ROEVER W.P. [80], "A proof system for communicating séquential processes", TOPLAS 2, 3 (1980), 359-385.

ASHCROFT E.A. [75], "Proving assertions about parallel programs", J. of Comp. and System Science, 10 (1975), 110-135.

ASHCROFT E.A., MANNA Z. [70], "Formalization of properties of parallel programs, Machine Intelligence, 6 (1970), 17-41.

COUSOT P. [77], "Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice", Rapport de Recherche N°88, IMAG, Université de Grenoble, (Mars 1978).

COUSOT P. [78], "Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique des programmes, Thèse d'Etat, Université de Grenoble, (Mars 1978).

COUSOT P. [79], "Analysis of the behavior of dynamic discrete systems", Rapport de Recherche n°161, IMAG, Université de Grenoble, (Jan. 1979).

COUSOT P. [81], "Semantic foundations of program analysis", dans "Program flow analysis, theory and applications", S.S. Muchnick & N.J. Jones (eds.), Prentice-Hall, (1981), 303-342.

COUSOT R. [81], "Proving invariance properties of parallel programs by backward induction", Rapport de Recherche CRIN-81-P026, (1981).

COUSOT P. COUSOT R. [76], "Static determination of dynamic properties of programs", Proc. 2nd Int. Symp. on Programming, Paris, Dunod, (Avril 1976), 106-130.

COUSOT P., COUSOT R. [77a], "Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints", Conf. Rec. of the 4th ACM Symp. on Principles of Programming Languages, Los Angeles, (Jan. 1977), 238-252.

COUSOT P., COUSOT R. [77b], "Static determination of dynamic properties of generalized type unions", ACM Conf. on Language Design for Reliable Software, Raleigh, SIGPLAN Notices 12, 3(1977), 77-94.

COUSOT P., COUSOT R. [77c], "Static determination of dynamic properties of recursive procedures", Conf. on Formal Description of Programming Concepts, St-Andrews, Canada, North-Holland Pub Co, (1977), 237-277.

COUSOT P., COUSOT R. [77d], "Automatic synthesis of optimal invariant assertions : mathematical foundations", Proc. ACM Symp. on Artificial Intelligence & Programming Languages, Rochester, SIGPLAN Notices 12, 8(1977), 1-12.

COUSOT P., COUSOT R. [79a], "Systematic design of program analysis frameworks", Conf. Rec. of the 6th ACM Symp. on Principles of Programming Languages, San Antonio, Texas, (1979), 269-282.

COUSOT P., COUSOT R. [79b], "Constructive versions of Tarski's fixed point theorems", Pacific Journal of Math., vol. 82, n°1, (1979), 43-57.

COUSOT P., COUSOT R. [80a], "Semantic analysis of communicating sequential processes", Automata, Languages and Programming, 7th Colloq. Lecture Notes in Computer Sci. 85, Springer-Verlag, (1980), 119-133.

COUSOT P., COUSOT R. [80b], "Constructive [illegible]", Proc. Int. Workshop on Program Construction, INRIA Ed., Tome 1, (1979),

COUSOT P., COUSOT R. [80c], "Reasoning about program invariance proof methods", Rapport de Recherche CRIN-80-P050, (1980).

COUSOT P., COUSOT R. [82a], "Induction principles for proving invariance properties of programs", dans "Tools and Notions for Program Construction", (D. Neel Ed.), Cambridge University Press, (1982), 75-119.

COUSOT P., COUSOT R. [82b], "'A la Floyd' induction principles for proving inevitability properties of programs", Rapport de recherche LRIM-82-04, à paraître dans "Algebraic Methods in Programming", (M. Nivat & J. Reynolds, eds.), Cambridge University Press.

COUSOT P., COUSOT R. [84], "Invariance proof methods and analysis techniques for parallel programs", dans "Automatic program construction techniques", (A. Biermann et al., eds.), MAC MILLAN, (1984), 243-271.

COUSOT P., HALBWACHS N. [78], "Automatic discovery of linear restraints among variables of a program", Conf. Rec. of the 5th ACM Symp. on Principles of Programming Languages, Tucson, Arizona, (1978), 84-97.

DIJKSTRA E.W. [82], "Selected writings on computing : a personal perspective", Springer-Verlag, (1982).

FLOYD R.W. [67], "Assigning meaning to programs", Proc. Symp. in Applied Math., AMS, Providence, RI, (1967), 19-32.

HOARE C.A.R. [69], "An axiomatic basis for computer programming", CACM 12, 10(1969), 576-580, 583.

HOARE C.A.R. [72], "Toward a theory of parallel programming", dans Operating System Techniques (Hoare & Perrott eds.), Academic Press, (1972).

HOARE C.A.R. [75], "Parallel programming: an axiomatic approach", Computer Languages, 1 (1975), 151-160.

HOARE C.A.R. [78], "Communicating sequential processes", CACM 21, 8 (1978), 666-677.

HOWARD J.H. [76], "Proving monitors", CACM 19, 5 (1976), 273-279.

KELLER R.M. [76], "Formal verification of parallel programs", CACM 19, 7 (1976), 371-384.

LAMPORT L. [77], "Proving the correctness of multiprocess programs", IEEE Trans. on Soft. Eng., SE3, 2 (1977), 125-143.

LAMPORT L. [80], "The 'Hoare Logic' of concurrent programs", Acta Informatica 14, (1980), 21-37.

LEVIN G.M. [79], "A proof technique for communicating sequential processes (with an example)", TR79-401, Comp. Sci. Dept, Cornell U., N.Y., (1979).

MANNA Z. [70], "Mathematical theory of partial correctness", JCSS 5, 3 (1970), 238-253.

MAZURKIEWICZ A [71], "Concurrent program schemes and their interpretation", Dept. Comp. Sci, Aarhus U., Danemark, DAIMI-PB-78, (1971).

MISRA J. [78], "Some aspects of the verification of loop computations", IEEE Trans. on Soft. Eng., SE-4, 6 (1978), 478-486.

MORRIS J.H., WEGBREIT B. [77], "Subgoal induction", CACM 20, 4 (1977), 209-222.

NAUR P. [66], "Proof of algorithms by general snapshots", BIT 6 (1966), 310-316.

OWICKI S., GRIES D. [76a], "An axiomatic proof technique for parallel programs I", Acta Informatica, 6 (1976), 319-340

OWICKI S., GRIES D. [76b], "Verifying properties of parallel programs: an axiomatic approach", CACM 19, 5 (1976), 279-285.

RICART G., AGRAWALA A.K. [81], "An optimal algorithm for mutual exclusion in computer networks", CACM 24, 1 (1981), 9-17.

TARSKI A. [55], "A lattice theoretical fixpoint theorem and its applications", Pacific Journal of Math., 5 (1955), 285-310.

5. PREUVES DE FATALITE

5. PREUVES DE FATALITE

5.1 RELATIONS ENTRE SEMANTIQUES CONSERVANT LA FATALITE

5.1.1 CONSERVATION DE PROPRIETES DE FATALITE PAR INCLUSION DE SEMANTIQUES

5.1.2 CONSERVATION DE PROPRIETES DE FATALITE POUR DES SEMANTIQUES CONCORDANTES A DES RELATIONS ENTRE ETATS ET ACTIONS PRES

5.2 PRINCIPES D'INDUCTION "A LA FLOYD"

5.2.1 RAPPEL DE LA METHODE DE FLOYD (DITE DES ASSERTIONS INVARIANTES ET DE L'ORDRE BIEN-FONDE) POUR DEMONTRER LA CORRECTION TOTALE DE PROGRAMMES SEQUENTIELS

5.2.1.1 Correction partielle

5.2.1.2 Absence d'erreurs à l'exécution (ou absence d'états de blocage)

5.2.1.3 Terminaison

5.2.2 LE PRINCIPE D'INDUCTION DE BASE POUR LES SEMANTIQUES CLOSES

5.2.3 PRINCIPES D'INDUCTION EQUIVALENTS POUR LES SEMANTIQUES CLOSES

5. PREUVES DE FATALITE

Nous étudions dans ce chapitre les méthodes de preuve de propriétés de fatalité des programmes (cf. 3.3) en utilisant la même approche que dans le chapitre précédent. Ceci consiste à partir des méthodes de preuves classiques pour démontrer la correction totale des programmes séquentiels (Floyd [67], Burstall [74] principalement) en les formalisant à l'aide de principes d'induction pour démontrer des propriétés de fatalité de systèmes de transition ce qui permet de considérer une classe de propriétés plutôt qu'une propriété particulière et de faire abstraction d'un langage de programmation particulier ou d'une méthode particulière de présentation de la preuve. Il est ensuite plus facile de faire des généralisations au cas des systèmes de transition non déterministes (et donc aux programmes parallèles, en particulier asynchrones) puis aux sémantiques non closes (et donc en particulier, aux programmes parallèles équitables). Cette formalisation facilite la compréhension des nombreuses variantes de ces méthodes de preuve ainsi que les preuves de correction, de complétude sémantique et d'équivalence de ces variantes.

La méthode la plus connue pour démontrer la correction totale des programmes séquentiels est la méthode de Floyd [67] dite des "assertions invariantes et de l'ordre bien-fondé" qui consiste à décomposer la preuve en une preuve de correction partielle, une preuve d'absence d'erreurs à l'exécution (ou de blocages) et une preuve de terminaison à l'aide d'une quantité qui décroit à chaque pas du programme ou à chaque cycle dans une boucle mais qui ne peut pas décroitre indéfiniment). Le chapitre précédent nous a déjà permis de formaliser

la partie preuves d'invariance (correction partielle, absence de blocages) et de manière générale la preuve de terminaison se formalise à l'aide de la notion d'ensemble bien-fondé.

Lorsque le non-déterminisme n'est pas fini, Dijkstra [76, 82] a montré qu'il n'est pas toujours possible d'utiliser des relations dont le rang (on dit aussi ordre) est strictement inférieur à ω et nous caractérisons les relations bien fondées qui sont nécessaires et suffisantes pour la complétude sémantique dans divers cas de non-déterminisme borné généralisant le non-déterminisme fini de Dijkstra.

Lorsque la sémantique du programme n'est pas close, la méthode d'induction sur les calculs de Floyd n'est pas applicable sans recourir à des variables auxiliaires. Une première approche consiste à appliquer la méthode de Floyd à une relation de transition auxiliaire (portant sur les états et les variables d'histoire) qui engendre exactement l'ensemble de traces original. Une seconde approche qui généralise l'utilisation de points de coupure des boucles dans la méthode de Floyd pour les programmes séquentiels, consiste à utiliser des points de coupure (où une fonction de terminaison décroit strictement) choisis dynamiquement c'est-à-dire en fonction de l'histoire des calculs cumulée dans les variables auxiliaires. Nous montrerons ensuite que ces deux approches sont en fait fortement équivalentes.

La méthode de Burstall [74] dite des "assertions intermittentes" pour démontrer la correction totale des programmes séquentiels est moins connue, souvent ignorée des ouvrages de base (nous n'avons pas trouvé de références dans Lifschitz [78] ou Berg et al [82] par exemple) et est sujette à polémiques (Manna-Waldinger [78], Gries [82]). Nous nous sommes intéressés [illegible] toujours présentée de manière très différente

de la méthode de Floyd nous ne comprenions pas bien le rapport entre ces deux méthodes.

A partir de la description donnée par Burstall et Manna-Waldinger de la méthode des assertions intermittentes (principalement à l'aide d'exemples) nous dérivons un principe d'induction dont nous montrons la correction. Il est sémantiquement complet mais sous une condition qui porte sur les traces d'exécution et la propriété de fatalité considérée. En particulier cette condition est remplie quand on s'intéresse à la correction totale ou bien à des propriétés de fatalité qui s'expriment à l'aide d'assertions unaires sur des états. Cependant nous faisons la conjecture que le principe d'induction de base n'est pas complet quand on considère des propriétés de fatalité arbitraires qui sont binaires (c'est-à-dire reliant les valeurs des états à des instants différents dans le temps) et lorsqu'on ne s'autorise pas l'emploi de variables auxiliaires.

Cette conjecture nous conduit à une généralisation de la méthode de Burstall en utilisant une induction transfinie (de manière à prendre en compte le non-déterminisme non borné) et en introduisant l'utilisation de variables auxiliaires sous la forme très limitée d'assertions ternaires (qui permettent de relier l'état des variables au début du programme ainsi qu'à deux autres instants différents au cours du calcul).

A partir de ce principe d'induction généralisé nous dérivons une série de principes d'induction de manière à élargir le champ des formes de preuves permises. Ceci nous permet également de formuler la méthode de Burstall sous des formes de plus en plus abstraites pour n'en retenir finalement que l'essence.

Tous les principes d'induction considérés sont corrects et sémantiquement complets (comme le montre l'argument de Manna-Waldinger [78]

qui consiste à dire qu'une preuve à la Floyd peut s'exprimer par la méthode de Burstall). Toutefois nous démontrons un résultat de complétude sémantique plus fort en ce sens que les propositions et les lemmes qui interviennent dans une preuve par la méthode des assertions intermittentes telle que nous l'avons généralisée peuvent être choisis librement (du moins sous une condition nécessaire et suffisante que nous stipulons avec précision).

La formalisation identique des méthodes de Floyd et Burstall nous permet de comprendre simplement leur rapport. Non seulement (comme l'ont montré Manna-Waldinger) toute preuve à la Floyd peut s'exprimer par la méthode de Burstall mais toute preuve à la Floyd est une preuve à la Burstall ! En effet, le principe d'induction qui exprime l'essence de la méthode de Burstall (convenablement généralisée comme nous le proposons) montre clairement que la méthode de Floyd est un simple cas particulier de la méthode de Burstall. Par analogie avec la programmation; ce rapport de la méthode de Burstall à celle de Floyd est similaire au rapport qu'a une programmation itérative avec un seul programme principal avec une programmation utilisant des sous-programmes éventuellement récursifs.

Pour tirer des conclusions pratiques de cette remarque, il faut à élaborer une méthode de présentation des preuves qui soit aussi bien utilisable pour des preuves à la Floyd qu'à la Burstall. C'est pourquoi nous proposons une présentation graphique des preuves de structure sous la forme de chartes de preuves qui nous est certaine des aspects de manière structurée. Nous montrons que cette présentation des preuves est correcte, sémantiquement complète et convient pour démontrer

Pour conclure sur la comparaison de ces méthodes, nous montrons qu'elles sont fortement équivalentes, en ce sens qu'une preuve par la méthode de Burstall peut se réécrire en une preuve par la méthode de Floyd (de la même façon qu'il est toujours possible d'éliminer la récursivité et les sous-programmes d'un programme).

Nous avons choisi de présenter ce chapitre en allant du particulier (méthode de Floyd) au général (méthode de Burstall) ce qui correspond à l'historique de la découverte de ces méthodes et à une complexité croissante des principes d'induction. Bien que moins satisfaisante pour l'esprit qu'une démarche descendante du général au particulier, cette présentation nous a semblé préférable parce qu'elle gradue les difficultés.

5.1 RELATIONS ENTRE SEMANTIQUES CONSERVANT LA FATALITE

Nous étudions dans ce paragraphe les relations entre sémantiques (définies en 2.5 et 2.6) qui conservent les propriétés de fatalité. Ceci permet en particulier de justifier très simplement les méthodes de preuve de programmes basées sur une transformation du programme.

5.1.1 CONSERVATION DE PROPRIETES DE FATALITE PAR INCLUSION DE SEMANTIQUES

De manière évidente d'après la définition 3.3.1, nous avons :

Théorème 5.1.1-1

Si $\langle S,A,\Sigma\rangle \subseteq \langle S',A',\Sigma'\rangle$, $\phi,\psi \in (S'\times S' \to \{tt,ff\})$ alors

$\Rightarrow$ [ψ est fatale sous invariance de ϕ pour $\langle S',A',\Sigma'\rangle$]
$\not\Leftarrow$ [ψ est fatale sous invariance de ϕ pour $\langle S,A,\Sigma\rangle$]

Pour voir que la réciproque ($\not\Leftarrow$) n'est pas vraie il suffit de considérer le contre-exemple suivant :

Exemple 5.1.1-1

La sémantique $\langle S,A,\Sigma\rangle$ définie dans l'exemple 2.1.2-3 ($S=\{s_0,s_1,\dots\}$, $A=\{a,b\}$, $\Sigma=\{s_0 \xrightarrow{b} s_1,\ s_0 \xrightarrow{a} s_2 \xrightarrow{a} \dots \xrightarrow{a} s_i \xrightarrow{a} \dots \xrightarrow{a} s_\omega \xrightarrow{b} s_f\}$) est inclue dans la sémantique $\langle S',A',\Sigma'\rangle$ définie dans l'exemple 2.1.2-3 ($S'=\{s_0,s_1,\dots\}$, $A=\{a,b\}$, $\Sigma'=\{s_0 \xrightarrow{b} s_1,\dots,s_0 \xrightarrow{a} s_0 \dots s_0 \xrightarrow{a} s_0 \xrightarrow{b} s_1,\dots,s_0 \xrightarrow{a} s_0 \dots\}$) (et telle que $\langle S,A,\Sigma\rangle \subseteq \langle S',A',\Sigma'\rangle$). ψ définie par $\psi(s_i,s_f)$ et $\psi(s_i,s_j)$ est fatale pour $\langle S,A,\Sigma\rangle$ mais pas pour $\langle S',A',\Sigma'\rangle$

Corollaire 5.1.1-2

Si $\langle S,A,\Sigma\rangle = \underline{Efus}(\langle S,A,\Sigma\rangle)$ et $\langle S,A,\Sigma\rangle = \underline{Reltps}(\langle S,A,\Sigma\rangle)$ alors

$\Rightarrow$ [ψ est fatale sous invariance de ϕ pour $\underline{Rtran}(\langle S,A,\Sigma\rangle)$]
[ψ est fatale sous invariance de ϕ pour $\langle S,A,\Sigma\rangle$]

5.1.2 CONSERVATION DE PROPRIETES DE FATALITE POUR DES SEMANTIQUES CONCORDANTES A DES RELATIONS ENTRE ETATS ET ACTIONS PRES

Théorème 5.1.2∾1

Si $\simeq\langle rs, ra\rangle(\langle S,A,\Sigma\rangle, \langle S',A',\Sigma'\rangle)$
$\wedge\ rs^{-1} \subseteq \phi \circ rs \Rightarrow \phi'$
$\wedge\ rs^{-1} \subseteq \psi \circ rs \rightarrow \psi'$

alors

$\Rightarrow$ [ψ est fatale sous invariance de ϕ pour $\langle S,A,\Sigma\rangle$]
$\Leftarrow$ [ψ' est fatale sous invariance de ϕ pour $\langle S',A',\Sigma'\rangle$]

Démonstration

($\rightarrow$) Si ψ est fatale sous invariance de ϕ pour $\langle S,A,\Sigma\rangle$ et $p\in\Sigma'$ alors il existe $p\in\Sigma$ tel que $|p|=|p'|$ et $\forall R\in|p|.\ rs(p_R, p'_R)$. Comme $\exists i\in|p|.[\forall j\in i.\ \phi(p_0,p_j) \wedge \psi(p_0,p_i)]$ il vient $\exists i\in|p'|.[\forall j\in i.\ rs^{-1}\subseteq\phi\subseteq rs(p'_0,p'_j)\wedge rs^{-1}\subseteq\psi\subseteq rs(p'_0,p'_i)]$ et donc $\exists i\in|p'|.[\forall j\in i.\ \phi'(p'_0,p'_j)\wedge\psi'(p'_0,p'_i)]$.

($\leftarrow$) Choisir $S=\{0,1\}$, $A=\phi$, $\Sigma=\{0,1\}$, $S'=\{0'\}$, $A'=\phi$, $\Sigma'=\{0'\}$, $rs(0,0')$, $rs(1,0')$, $\psi(0,0)$, $\neg\psi(1,1)$, $\phi=\psi$. Nous avons $\phi'=\psi'=rs^{-1}\subseteq\psi\subseteq rs(0',0')$ et donc ψ' est fatale (sous invariance de ϕ') pour Σ' mais ψ n'est pas fatale (sous invariance de ϕ) pour Σ.

□

La réciproque est vraie si nous ajoutons des conditions supplémentaires

Théorème 5.1.2∾2

Si $\simeq\langle rs, ra\rangle(\langle S,A,\Sigma\rangle, \langle S',A',\Sigma'\rangle)$
$\wedge\ rs^{-1}\subseteq\phi\subseteq rs\rightarrow\phi' \wedge rs^{-1}\subseteq\psi\subseteq rs\Rightarrow\psi'$
$\wedge\ ra\subseteq\phi'\circ ra^{-1}\rightarrow\phi \wedge ra\subseteq\psi'\subseteq ra^{-1}\rightarrow\psi$

alors

$\Leftrightarrow$ [ψ est fatale sous invariance de ϕ pour $\langle S,A,\Sigma\rangle$]
[ψ' est fatale sous invariance de ϕ' pour $\langle S',A',\Sigma'\rangle$]

Démonstration

($\Rightarrow$) cf. 5.1.2∾1

($\Leftarrow$) Si ψ' est fatale sous invariance de ϕ' pour $\langle S',A',\Sigma'\rangle$ et $p\in\Sigma$ alors $\Sigma'=\simeq\langle rs,ra\rangle[\Sigma]$ implique l'existence de $p'\in\Sigma$ tel que $\simeq\langle rs,ra\rangle(p,p')$ et $\exists i\in|p'|=|p|.(\forall j\in i.\ \phi'(p'_0,p'_j)\wedge\psi(p_0,p_i))$. Par conséquent nous avons $\forall j\in i.(rs(p_0,p'_0)\wedge\phi'(p'_0,p'_j)\wedge rs^{-1}(p'_j,p_j))\wedge(rs(p_0,p'_0)\wedge\psi'(p'_0,p'_i)\wedge rs^{-1}(p'_i,p_i))$ ce qui implique $\forall j\in i.\ \phi(p_0,p_j)\wedge\psi(p_0,p_i)$.

□

Dans le cas particulier d'une concordance entre sémantiques à une fonction entre états près, nous obtenons le corollaire suivant.

Corollaire 5.1.2∾3

Si $\langle S',A',\Sigma'\rangle = \simeq\langle fs\rangle(\langle S,A,\Sigma\rangle)$
$\wedge\ \phi(A_1,A_2) = \phi'(fa(A_1), fa(A_2))$
$\wedge\ \psi(A_1,A_2) = \psi'(fa(A_1), fa(A_2))$

alors

$\Leftrightarrow$ [ψ est fatale sous invariance de ϕ pour $\langle S,A,\Sigma\rangle$]
[ψ' est fatale sous invariance de ϕ' pour $\langle S',A',\Sigma'\rangle$]

5.2 PRINCIPES D'INDUCTION "A LA FLOYD"

Par abstraction à partir de la méthode de Floyd[67] (méthode des assertions invariantes et de l'ordre bien-fondé pour montrer la correction totale de programmes séquentiels), nous proposons des principes d'induction pour démontrer les propriétés de fatalité de systèmes de transition. Nous démontrons que ces principes d'induction sont corrects, sémantiquement complets et équivalents. Ceci formalise la méthode de Floyd indépendemment d'un langage de programmation particulier et d'un langage d'assertions particulier et la généralise au cas de systèmes de transition non-déterministes et donc aux programmes parallèles. Considérant différentes classes de non-déterminisme borné, nous caractérisons les relations bien-fondées correspondantes qui sont nécessaires et suffisantes pour la complétude sémantique.

Quand la sémantique n'est pas close (par exemple dans le cas d'une exécution parallèle équitable), la méthode de Floyd ne peut pas s'appliquer sans utiliser des variables auxiliaires. Une première approche consiste à appliquer la méthode de Floyd à une relation de transition auxiliaire (portant sur les états et des variables d'histoire) qui engendre exactement la sémantique originale. Une seconde approche qui généralise l'utilisation de points de coupure des boucles dans la méthode de Floyd pour des programmes séquentiels consiste à utiliser des points de coupure (si une fonction de terminaison décroit strictement) choisis dynamiquement c'est-à-dire en fonction de l'histoire des calculs mémorisée dans des variables auxiliaires. Nous montrons que ces deux approches sont fondamentalement équivalentes.

5.2.1 RAPPEL DE LA METHODE DE FLOYD (DITE DES ASSERTIONS INVARIANTES ET DE L'ORDRE BIEN-FONDE) POUR DEMONTRER LA CORRECTION TOTALE DE PROGRAMMES SEQUENTIELS

Floyd[67] considère des programmes séquentiels P_s (comme en 2.8.1) avec des états de la forme $\langle c, m\rangle \in S[\![P_s]\!]$ où $c \in C[\![P_s]\!]$ est un point de contrôle et $m \in \mathcal{M}$ est un état mémoire (affectant des valeurs aux variables). Soient $\psi \in (\mathcal{M}\times\mathcal{M} \to \{tt, ff\})$ une spécification de sortie et $\sigma \in (S[\![P_s]\!] \to \{tt, ff\})$ une fonction caractérisant les états de sortie soit $\bar{\psi} \in (S[\![P_s]\!] \times S[\![P_s]\!] \to \{tt, ff\})$ telle que $\bar{\psi}(\langle \underline{c}, \underline{m}\rangle, \langle \bar{c}, \bar{m}\rangle) = \psi(\underline{m}, \bar{m})$.

La correction totale est une propriété de fatalité de la forme :

$$\forall p \in \Sigma\langle S, A, t, \varepsilon\rangle.\ \exists i \in |p|.\ [(\forall j \in i.\ \neg\sigma(p_j)) \wedge \sigma(p_i) \wedge \bar{\psi}(p_0, p_i)]$$

D'après la méthode de Floyd, on démontre la correction totale en démontrant d'abord la correction partielle, puis l'absence d'erreurs à l'exécution (c'est-à-dire d'après 2.8.1.2.4, l'absence d'états de blocage) et finalement la terminaison.

5.2.1.1 Correction partielle

La méthode de Floyd[67]-Naur[66] de preuve de correction partielle consiste à d'abord associer une assertion $P_c \in (\mathcal{M}\times\mathcal{M} \to \{tt, ff\})$ à chaque point de contrôle c du programme ($P_c(\underline{m}, m)$ reliant l'état mémoire courant m à l'état mémoire initial $\underline{m}$) puis montrer que ces assertions sont invariantes et finalement montrer que l'assertion associée aux états finaux implique la spécification d'entrée-sortie.

Pour montrer que ces assertions sont invariantes, on doit d'abord montrer que l'assertion d'entrée est vraie :

$\forall \underline{c} \in C[\![Ps]\!],\ \underline{m} \in \mathcal{M}.\ [\epsilon(\langle \underline{c}, \underline{m} \rangle) \Rightarrow P_{\underline{c}}(\langle \underline{m}, \underline{m} \rangle)]$

Puis pour chaque commande du programme, on doit montrer que si le contrôle était au point c avec P_c vraie avant l'exécution de la commande alors après exécution (si elle se termine correctement), le contrôle doit être en c' avec $P_{c'}$ vraie :

$\forall c, c' \in C[\![Ps]\!],\ \underline{m}, m, m' \in \mathcal{M}_\perp.$

$([P_c(\underline{m}, m) \wedge \neg\delta(\langle c, m \rangle) \wedge t_{\underline{a}}(\langle c, m \rangle, \langle c', m' \rangle)] \Rightarrow P_{c'}(\underline{m}, m'))$

Finalement, on doit montrer que si l'exécution atteint un point de sortie du programme alors l'assertion associée à ce point doit impliquer la spécification :

$\forall \bar{c} \in C[\![Ps]\!],\ \underline{m}, \bar{m} \in \mathcal{M}_\perp.\ ([P_{\bar{c}}(\underline{m}, \bar{m}) \wedge \delta(\langle \bar{c}, \bar{m} \rangle)] \Rightarrow \psi(\underline{m}, \bar{m}))$

5.2.1.2 Absence d'erreurs à l'exécution (ou absence d'états de blocage)

Si des opérations partielles sont utilisées dans un programme, alors une preuve d'absence d'erreurs à l'exécution doit montrer qu'elles ne donnent pas de résultats indéfinis. Si par convention, la sémantique opérationnelle est définie de sorte que les états conduisant à des résultats indéfinis soient des états de blocage alors une preuve d'absence d'erreurs à l'exécution consiste à montrer que les états accessibles qui ne sont pas des états finaux doivent avoir au moins un successeur :

$\forall c \in C[\![Ps]\!],\ \underline{m}, m \in \mathcal{M}_\perp.$

$([P_c(\underline{m}, m) \wedge \neg\delta(\langle c, m \rangle)] \Rightarrow [\exists c' \in C[\![Ps]\!],\ m' \in \mathcal{M}_\perp.\ t_{\underline{a}}(\langle c, m \rangle, \langle c', m' \rangle)])$

5.2.1.3 Terminaison

La méthode de preuve de terminaison de Floyd consiste d'abord à associer à chaque point de contrôle $c \in C[\![Ps]\!]$ du programme, une fonction de terminaison $f_c \in (\mathcal{M}_\perp \times \mathcal{M} \to Rng(f_c))$.

Puis on montre que les valeurs de cette fonction appartiennent à un bon-ordre $\langle W, \prec \rangle$, $W \in Rng(f_c)$:

$\forall c \in C[\![Ps]\!],\ \underline{m}, m \in \mathcal{M}_\perp.$

$([P_c(\underline{m}, m) \wedge \neg\delta(\langle c, m \rangle)] \Rightarrow f_c(\underline{m}, m) \in W)$

Finalement, on montre qu'après chaque exécution d'une commande, la valeur courante de la fonction de terminaison associée à son point de sortie est strictement plus petite que la valeur de la fonction de terminaison associée à son point d'entrée :

$\forall c, c' \in C[\![Ps]\!],\ \underline{m}, m, m' \in \mathcal{M}_\perp.$

$([P_c(\underline{m}, m) \wedge \neg\delta(\langle c, m \rangle) \wedge t_{\underline{a}}(\langle c, m \rangle, \langle c', m' \rangle)] \Rightarrow [f_{c'}(\underline{m}, m') \prec f_c(\underline{m}, m)])$

5.2.2 LE PRINCIPE D'INDUCTION DE BASE POUR LES SEMANTIQUES CLOSES

Au lieu d'utiliser des assertions locales P_c associées aux points de contrôle $c \in C[\![Ps]\!]$, nous pouvons utiliser un invariant global J tel que (cf. 4.2.1.1) :

$$J \in (S[\![Ps]\!]^2 \to \{tt, ff\})$$

$$J(\langle c, m\rangle, \langle c, m\rangle) = P_c(m, m)$$

et une fonction de terminaison globale telle que :

$$f \in (S[\![Ps]\!]^2 \to \cup\{Rng(f_c) : c \in C[\![Ps]\!]\})$$

$$f(\langle c, m\rangle, \langle c, m\rangle) = f_c(m, m)$$

Il est alors trivial de vérifier que les conditions de vérification de Floyd (définies comme en 5.2.1) sont équivalentes aux suivantes :

- Correction partielle :

$$\begin{aligned}[& (\varepsilon(\underline{\Delta}) \Rightarrow J(\underline{\Delta},\underline{\Delta}))\\ & \wedge([J(\underline{\Delta},\Delta) \wedge \neg\sigma(\Delta) \wedge t_a(\Delta,\Delta')] \Rightarrow J(\underline{\Delta},\underline{\Delta}'))\\ & \wedge([J(\underline{\Delta},\bar{\Delta}) \wedge \sigma(\bar{\Delta})] \Rightarrow \bar{\Psi}(\underline{\Delta},\bar{\Delta}))]\end{aligned}$$

- Absence d'états de blocage :

$$([J(\underline{\Delta},\Delta) \wedge \neg\sigma(\Delta)] \Rightarrow [\exists \Delta' \in S[\![Ps]\!] . t_a(\Delta,\Delta')])$$

- Terminaison :

$$\begin{aligned}[& ([J(\underline{\Delta},\Delta) \wedge \neg\sigma(\Delta)] \Rightarrow f(\underline{\Delta},\Delta) \in W)\\ & \wedge([J(\underline{\Delta},\Delta) \wedge \neg\sigma(\Delta) \wedge t_a(\Delta,\Delta')] \Rightarrow [f(\underline{\Delta},\Delta') \prec f(\underline{\Delta},\Delta)])]\end{aligned}$$

Si maintenant nous prenons :

$J, \Psi \in (S[\![Ps]\!]^2 \to \{tt, ff\})$,

$\Phi(\underline{\Delta},\Delta) = [\neg\sigma(\Delta)]$

$\Psi(\Delta,\bar{\Delta}) = [\sigma(\bar{\Delta}) \wedge \bar{\Psi}(\underline{\Delta},\bar{\Delta})]$

alors nous pourrons vérifier aisément que la méthode de Floyd repose sur le principe d'induction suivant :

$$\begin{aligned}&(\exists J \in (S^2 \to \{tt, ff\}),\ f \in (S^2 \to Rng(f)),\ W \in Rng(f),\ \prec \in (Rng(f) \times Rng(f) \to \{tt, ff\}).\\ &\quad \underline{wo}(W, \prec)\\ &\quad \wedge [\forall \underline{\Delta}, \Delta \in S.\\ &\qquad (\varepsilon(\underline{\Delta}) \Rightarrow J(\underline{\Delta},\underline{\Delta}))\\ &\qquad \wedge (J(\underline{\Delta},\Delta) \Rightarrow \Psi(\underline{\Delta},\Delta)\\ &\qquad\qquad \vee [\Phi(\underline{\Delta},\Delta) \wedge f(\underline{\Delta},\Delta) \in W \wedge \exists \Delta' \in S, a \in A . t_a(\Delta,\Delta') \wedge\\ &\qquad\qquad\quad \forall \Delta' \in S, a \in A . (t_a(\Delta,\Delta') \Rightarrow [J(\underline{\Delta},\Delta') \wedge f(\underline{\Delta},\Delta') \prec f(\underline{\Delta},\Delta)])])])\end{aligned} \qquad (\mathcal{F}_1^{\#})$$

où $\underline{wo}(W,\prec)$ caractérise les bons-ordres sur W.

5.2.3 PRINCIPES D'INDUCTION EQUIVALENTS POUR LES SEMANTIQUES CLOSES

Des variantes du principe d'induction de base sont souvent utilisées. Nous introduisons maintenant des transformations successives qui conduisent à différents principes d'induction. Nous montrons que tous ces principes d'induction sont équivalents au principe d'induction de base $(\mathcal{F}_1)$.

Le co-domaine de la fonction de terminaison f peut toujours être choisi de sorte qu'il coïncide avec le sous-ensemble W de l'ordre $\prec$:

$$
\begin{array}{l}
(\exists J \in (S^2 \to \{tt,ff\}),\ f \in (S^2 \to \underline{Rng}(f)),\ \prec \in (\underline{Rng}(f) \times \underline{Rng}(f) \to \{tt,ff\}). \\
\quad \underline{wo}(\underline{Rng}(f), \prec) \\
\quad \wedge\ [\forall \underline{a}, a \in S. \\
\qquad (\varepsilon(\underline{a}) \Rightarrow J(\underline{a},\underline{a})) \\
\qquad \wedge\ (J(\underline{a},a) \Rightarrow \Psi(\underline{a},a) \\
\qquad\quad \vee\ [\Phi(\underline{a},a) \wedge \exists a' \in S, a \in A.\, t_a(a,a') \wedge \\
\qquad\qquad \forall a' \in S, a \in A.\, (t_a(a,a') \Rightarrow [J(\underline{a},a') \wedge f(\underline{a},a') \prec f(\underline{a},a)])])])
\end{array}
\qquad (\mathcal{F}_1)
$$

Quand, dans les démonstrations d'équivalences, il sera nécessaire d'utiliser en même temps des objets qui sont différents dans les deux principes d'induction $(\mathcal{F}_1)$ et $(\mathcal{F}_2)$, mais auxquels, pour simplifier, nous avons donné le même nom (comme $\tau, \underline{a}, \prec, \ldots$) nous utiliserons des indices (comme $J_1, J_2, f_1, f_2, \prec_1, \prec_2, \ldots$).

Théorème 5.2.3.1

Démonstration

Choisir $J_2(\underline{a},a) = [(f_1(\underline{a},a) \in W_1 \wedge J_1(\underline{a},a)) \vee \Psi(\underline{a},a)]$, $\underline{Rng}(f_2) = (W_1 \cup \{\perp\})$ avec $\perp \notin W_1$, $f_2(\underline{a},a) = \langle (f_1(\underline{a},a) \in W_1) \to f_1(\underline{a},a) \mid \perp \rangle$, $w' \prec_2 w$ si et seulement si $[(w' = \perp \wedge w \in W_1) \vee (w' \in W_1 \wedge w \in W_1 \wedge w' \prec_1 w)]$

□

Il n'est pas nécessaire d'associer une fonction de terminaison à tous les points de contrôle du programme mais seulement aux points de coupure des boucles :

$$
\begin{array}{l}
(\exists K \subseteq S,\ J \in (K \times K \times S \to \{tt,ff\}),\ f \in (K^2 \to \underline{Rng}(f)),\ \prec \in (\underline{Rng}(f) \times \underline{Rng}(f) \to \{tt,ff\}). \\
\quad \underline{Cutset}\ \langle S,A,t,\varepsilon \rangle (K) \\
\quad \wedge\ \underline{wo}(\underline{Rng}(f), \prec) \\
\quad \wedge\ [\forall \underline{a}, a \in K, a' \in S. \\
\qquad (\varepsilon(\underline{a}) \Rightarrow J(\underline{a},\underline{a},\underline{a})) \\
\qquad \wedge\ (J(\underline{a},a,a') \Rightarrow \Psi(\underline{a},a') \\
\qquad\quad \vee\ [\Phi(\underline{a},a') \wedge \exists a'' \in S, a \in A.\, t_a(a',a'') \wedge \forall a'' \in S, a \in A. \\
\qquad\qquad t_a(a',a'') \Rightarrow [(a'' \in K \wedge f(\underline{a},a'') \prec f(\underline{a},a) \wedge J(\underline{a},a'',a'')) \\
\qquad\qquad\quad \vee\ (a'' \notin K \wedge J(\underline{a},a,a''))])])])
\end{array}
\qquad (\mathcal{F}_3)
$$

où $\underline{Cutset}\ \langle S,A,t,\varepsilon \rangle (K) = [(K \subseteq S) \wedge (\forall \underline{a} \in S.(\varepsilon(\underline{a}) \Rightarrow \underline{a} \in K)) \wedge \forall p \in T^{\omega}\langle S,A,t,tt \rangle.\, \exists i \in \omega.\, f_i \in K]$

Un ensemble de points de coupure ("cutset") est une classe d'états (qui pour un programme bien défini, inclut les états d'entrée et) telle que s'il y avait une exécution infinie du programme, celle-ci passerait infiniment souvent par des états appartenant à l'ensemble de points de coupure.

Théorème 5.2.3 ~ 2

$(\mathcal{F}_2) \Rightarrow (\mathcal{F}_3)$

Démonstration

Choisir $J_3(\underline{s},A,A') = [J_2(\underline{s},A') \wedge A = A']$, $\underline{Rng}(f_3) = \underline{Rng}(f_2)$, $f_3 = f_2$, $\prec_3 = \prec_2$ et $K = S$.

□

L'utilisation de bons ordres n'est pas obligatoire. Des relations bien-fondées sont suffisantes (et quelquefois plus commodes) :

$(\exists J \in (S^2 \to \{tt,ff\}),\ f \in (S^2 \to \underline{Rng}(f)),\ \prec \in (\underline{Rng}(f) \times \underline{Rng}(f) \to \{tt,ff\}).$

$\underline{wf}(\underline{Rng}(f), \prec)$

$\wedge\ [\forall \underline{s}, A \in S.$

$(\epsilon(A) \Rightarrow J(\underline{s},A))$

$\wedge\ (J(\underline{s},A) \Rightarrow \Psi(\underline{s},A)$

$\vee\ [\Phi(\underline{s},A) \wedge \exists A' \in S, a \in A.\ t_a(A,A') \wedge$

$\forall A' \in S, a \in A.\ (t_a(A,A') \Rightarrow [J(\underline{s},A') \wedge f(\underline{s},A') \prec f(\underline{s},A)]]]]]])$ $(\mathcal{F}_4)$

où $\underline{wf}(W,\prec)$ caractérise les relations bien fondées sur W.

Théorème 5.2.3 ~ 3

$(\mathcal{F}_3) \Rightarrow (\mathcal{F}_4)$

Démonstration

Puisque $\underline{wo}(\underline{Rng}(f_3), \prec_3)$ implique $\underline{wf}(\underline{Rng}(f_3), \prec_3)$ nous pouvons définir :

- $\mu \in (S \times S \to \underline{Ord})$

$\mu(\underline{s},A') = \big((\forall A \in S.\ \neg J_3(\underline{s},A,A') \vee \Psi(\underline{s},A')) \to 0 \mid$

$\cap \{\underline{rk}(\underline{Rng}(f_3), \prec_3)(f_3(\underline{s},A)) + 1 :\ A \in S \wedge J_3(\underline{s},A,A') \wedge \neg\Psi(\underline{s},A')\}\big)$

- $f_4 \in (S \times S \to \underline{Ord} \times S)$

$f_4(\underline{s},A') = \langle \mu(\underline{s},A'), A' \rangle$

- $\ll \in (S \times S \to \{tt,ff\})$ telle que $A' \ll A$ si et seulement si $[\exists a \in A.\ t_a(A,A') \wedge A' \notin K]$

- $\prec_4 \in (\underline{Rng}(f_4) \times \underline{Rng}(f_4) \to \{tt,ff\})$ telle que $\langle w',A' \rangle \prec_4 \langle w,A \rangle$ si et seulement si $((w' < w) \vee (w' = w \wedge A' \ll A))$

Puisque $\underline{wf}(\underline{Ord},<)$ et $\underline{Cutset}\langle S,A,t,\epsilon \rangle(K)$ impliquent $\underline{wf}(S,\ll)$ nous avons $\underline{wf}(\underline{Rng}(f_4), \prec_4)$. Si nous choisissons $J_4(\underline{s},A') = [\exists A \in S.\ J_3(\underline{s},A,A')]$ la démonstration consiste essentiellement à montrer que nous avons :

$[(\exists A \in S.\ J_3(\underline{s},A,A')) \wedge \neg\Psi(\underline{s},A') \wedge t_a(A',A'') \wedge A'' \in K] \Rightarrow (\mu(\underline{s},A') > \mu(\underline{s},A''))$

et

$[(\exists A \in S.\ J_3(\underline{s},A,A')) \wedge \neg\Psi(\underline{s},A') \wedge t_a(A',A'') \wedge A'' \notin K] \Rightarrow (\mu(\underline{s},A') \geq \mu(\underline{s},A''))$

□

La fonction de terminaison peut être remplacée par une variable auxiliaire (w, n'apparaissant pas comme une variable du programme) à valeurs dans le domaine d'une relation bien-fondée et qui "décroit strictement" à chaque pas du programme.

$$(\exists W,\ \prec \in (W^2 \to \{tt, ff\}),\ J \in (W \times S \times S \to \{tt, ff\}).$$

$$\begin{array}{l} \underset{\sim}{wf}(W, \prec) \\ \wedge\ [\forall \underline{a}, a \in S,\ w \in W. \\ \quad (\epsilon(\underline{a}) \Rightarrow [\exists \underline{w} \in W.\ J(\underline{w}, \underline{a}, \underline{a})]) \\ \quad \wedge (J(w, \underline{a}, a) \Rightarrow \Psi(\underline{a}, a) \\ \qquad \vee [\Phi(\underline{a}, a) \wedge \exists a' \in S, q \in A.\ t_q(a, a') \wedge \\ \qquad\quad \forall a' \in S, q \in A.\ (t_q(a, a') \Rightarrow [\exists w' \prec w.\ J(w', \underline{a}, a')])])] \end{array} \qquad (\mathcal{F}_5)$$

Théorème 5.2.3 v 4

$$(\mathcal{F}_4) \Rightarrow (\mathcal{F}_5)$$

Démonstration

Choisir $W_5 = \underset{\sim}{Rng}(f_4)$, $\prec_5 = \prec_4$, $J_5(w, \underline{a}, a) = [w = f_4(\underline{a}, a) \wedge J_4(\underline{a}, a)]$

□

Puisque les relations bien-fondées peuvent être plongées dans des bons-ordres, l'isomorphisme de bons-ordres est une relation d'équivalence et les ordinaux sont des représentants de chaque classe d'équivalence, nous pouvons toujours utiliser le bon-ordre $<$ sur la classe $\underset{\sim}{Ord}$ des ordinaux pour les preuves de fatalité :

$$(\exists \Gamma \in \underset{\sim}{Ord},\ J \in (\Gamma \times S \times S \to \{tt, ff\}).$$

$$\begin{array}{ll} (\mathcal{F}'_6.1) & (\forall a \in S.\ \exists \gamma \in \Gamma.\ J(\gamma, a, a)) \\ (\mathcal{F}'_6.2) & (\forall a, a' \in S,\ \gamma' \in \Gamma. \\ & \quad J(\gamma', a, a') \Longrightarrow \\ (\mathcal{F}'_6.2.a) & \quad [\Phi(a, \kappa) \wedge \exists a'' \in S, q \in A.\ t_q(a', a'') \wedge \forall a'' \in S, q \in A.\ [t_q(a', a'') \Rightarrow \exists \gamma'' < \gamma'.\ J(\gamma'', a, a'')]] \\ (\mathcal{F}'_6.2.b) & \quad \vee [\epsilon(a) \Rightarrow \Psi(a, \kappa')]]) \end{array} \qquad (\mathcal{F}'_5)$$

Théorème 5.2.3 v 5

$$(\mathcal{F}'_5) \Rightarrow (\mathcal{F}'_6)$$

Démonstration

Définir une fonction rang $\rho \in (W_5 \to \underset{\sim}{Ord})$ comme suit :

$$\rho(w) = \cap \{\alpha \in \underset{\sim}{Ord} : \forall w' \in W_5.\ [w' \prec_5 w \Rightarrow \rho(w') < \alpha]\}$$

(cette définition se justifie aisément par induction transfinie sur $\prec_5$, puisque $\prec_5$ est une relation bien-fondée sur W_5).

Observer que $\forall w', w \in W_5.\ [(w' \prec_5 w) \Rightarrow (\rho(w') < \rho(w))]$. Définir $\delta = \underset{\sim}{sup}^+ \{\rho(w)+1 : w \in W_5\}$.

Choisir :

$$\begin{array}{l} J_6(\gamma, \underline{a}, a) = [\epsilon(\underline{a}) \Rightarrow ([\gamma = 0 \wedge \Psi(\underline{a}, a)] \\ \qquad\qquad \vee [\exists w \in W_5.\ (J_5(w, \underline{a}, a) \wedge \gamma = \rho(w)+1)])] \end{array}$$

□

La classe bien-fondée auxiliaire $(W_1, \underset{\sim}{Rng}(f_2), \underset{\sim}{Rng}(f_3), \underset{\sim}{Rng}(f_4)$ ou $W_5)$ peut toujours être choisie comme $\langle \underset{\sim}{Ord}, < \rangle$ mais aussi comme $\langle S \times S, \prec \rangle$ où $\prec$ est une relation binaire bien-fondée sur S établie convenablement choisie :

$$(\exists J \in (S^2 \to \{tt, ff\}),\ \prec \in (S^2 \times S^2 \to \{tt, ff\}).$$

$$\underline{wf}(S^2, \prec)$$
$$\wedge\ [\forall \underline{s}, s \in S.$$
$$(\varepsilon(\underline{s}) \Rightarrow J(\underline{s}, \underline{s}))$$
$$\wedge\ (J(\underline{s}, s) \Rightarrow \Psi(\underline{s}, s)$$
$$\vee\ [\phi(\underline{s}, s) \wedge \exists s' \in S, a \in A.\ t_a(s, s') \wedge$$
$$\forall s' \in S, a \in A.\ (t_a(s, s') \Rightarrow [J(\underline{s}, s') \wedge \langle \underline{s}, s' \rangle \prec \langle \underline{s}, s \rangle])])])$$

$(\mathcal{F}_7)$

Théorème 5.2.3-6

$(\mathcal{F}_6) \Rightarrow (\mathcal{F}_7)$

Démonstration

Choisir $J_7(\underline{s}, s) = [\exists \gamma \in \Gamma.\ (\varepsilon(\underline{s}) \wedge J_6(\gamma, \underline{s}, s))]$, $\langle \underline{s}', s' \rangle \prec_7 \langle \underline{s}, s \rangle$ si et seulement si

$[\varepsilon(\underline{s}) \wedge \underline{s}' = \underline{s} \wedge \exists \gamma \in \Gamma. J_6(\gamma, \underline{s}, s) \wedge \neg\Psi(\underline{s}, s) \wedge \forall \gamma \in \Gamma.\ ([J_6(\gamma, \underline{s}, s) \wedge \gamma > 0] \Rightarrow \exists \gamma' < \gamma.\ J_6(\gamma', \underline{s}, s'))]$.

□

Les principes d'induction $(\mathcal{F}_1)$ à $(\mathcal{F}_7)$ sont tous équivalents dans le sens que si une preuve a été faite au moyen d'un certain principe d'induction $(\mathcal{F}_i)$ comportant $J_i, \prec_i, \ldots$ la preuve peut être reexprimée pour tout autre principe d'induction $(\mathcal{F}_j)$ puisque $J_j, \prec_j, \ldots$ peuvent être dérivés à partir de $J_i, \prec_i, \ldots$ en utilisant les règles de raisonnement dans les démonstrations $(\mathcal{F}_i) \to (\mathcal{F}_{i \bmod 7 + 1}) \to \ldots \to (\mathcal{F}_i)$.

Un dernier résultat est nécessaire :

Théorème 5.2.3-7

Démonstration

Choisir $W_1 = \underline{Rang}(\underline{F}) = \underline{Ord}$, $\prec_1 = <$, $J_1 = J_2$ et $f_{T+1}(\underline{s}, s) = \mu\{\alpha_{T_1}(\underline{s}, s') + 1 : \langle \underline{s}, s' \rangle \prec_7 \langle \underline{s}, s \rangle\}$

□

En utilisant les versions contrapositives de ces principes d'induction, nous pourrons démontrer les propriétés de fatalité des programmes par l'absurde. Par exemple $(\mathcal{F}_6)$ peut également s'écrire :

$$(\exists \Gamma \in \underline{Ord},\ J \in (\Gamma \times S \times S \to \{tt, ff\}).$$

$$[\forall \underline{s}, s, s', \bar{s} \in S,\ a \in A,\ \gamma \in \Gamma.$$
$$(\varepsilon(\underline{s}) \Rightarrow [\exists \gamma \in \Gamma.\ J(\gamma, \underline{s}, s)])$$
$$\wedge\ ([J(\gamma, \underline{s}, s) \wedge \gamma > 0] \to [\phi(\underline{s}, s) \wedge \exists s' \in S, a \in A.\ t_a(s, s')])$$
$$\wedge\ ([J(\gamma, \underline{s}, s) \wedge \gamma > 0 \wedge t_a(s, s')] \Rightarrow [\exists \gamma' < \gamma.\ J(\gamma', \underline{s}, s')])$$
$$\wedge\ (J(0, \underline{s}, \bar{s}) \Rightarrow \Psi(\underline{s}, \bar{s}))])$$

$(\mathcal{F}_6')$

dont la version contrapositive est :

$$(\exists \Gamma \in \underline{Ord},\ J \in (\Gamma \times S \times S \to \{tt, ff\}).$$

$$[\forall \underline{s}, s, s', \bar{s} \in S,\ a \in A,\ \gamma \in \Gamma.$$
$$(\neg\Psi(\underline{s}, \bar{s}) \Rightarrow J(0, \underline{s}, \bar{s}))$$
$$\wedge\ ([\gamma > 0 \wedge (\neg\phi(\underline{s}, s) \vee \forall s' \in S, a \in A.\ \neg t_a(s, s'))] \Rightarrow J(\gamma, \underline{s}, s))$$
$$\wedge\ ([\gamma > 0 \wedge t_a(s, s') \wedge \forall \gamma' < \gamma.\ \neg J(\gamma', \underline{s}, s')] \Rightarrow J(\gamma, \underline{s}, s))$$
$$\wedge\ (\varepsilon(\underline{s}) \Rightarrow [\exists \gamma \in \Gamma.\ \neg J(\gamma, \underline{s}, \underline{s})])])$$

$(\overline{\mathcal{F}_6'})$

Les versions positives et contrapositives des principes d'induction sont évidemment équivalentes :

Théorème 5.2.3-8

$(\mathcal{F}_i) \Longleftrightarrow (\overline{\mathcal{F}_i})$, $i = 1, \ldots, 7$

Démonstration

($\Rightarrow$) Choisir $\overline{J}_i = \neg J_i$. ($\Leftarrow$) Choisir $J_i = \neg \overline{J}_i$.

□

Exemple 5.2.3-1 (Preuve d'un programme de parcours d'arbre utilisant le principe d'induction $(\mathcal{F}_6)$)

Dans la suite (pour illustrer la méthode de Burstall [74]) nous utiliserons le programme séquentiel suivant (tiré littéralement de Burstall [74]) qui parcourt un arbre binaire à l'aide d'une pile en comptant ses feuilles externes :

La valeur de la variable Tr de type "arbre" est soit "mil" soit $(ff(Tr) . rg(Tr))$ où $ff(Tr)$ et $rg(Tr)$ sont des "arbres", la valeur de Co de type "nat" est un nombre naturel et la valeur de la variable St de type "pile" est soit () soit $(hd(St) . tl(St))$ où $hd(St)$ est un "arbre" et $tl(St)$ est une "pile".


```
Start:   St := ( ) ;  Co := 0 ;
Loop:    if Tr ≠ mil
             then begin  Push Tr onto St ;
                         Tr := ff(Tr) ;  goto Loop
                  end
             else begin  Co := Co + 1 ;
                         if St = ( ) then goto Finish ;
                         Pop Tr from St ;
                         Tr := rg(Tr) ;  goto Loop
                  end ;
Finish:
```

Dans la suite nous utiliserons cet exemple pour illustrer notre formalisation de la méthode de Burstall. Pour permettre sa comparaison avec la méthode de Floyd, nous allons montrer à l'aide de cet exemple qu'une preuve de correction totale par la méthode de Floyd consiste exactement à appliquer le principe d'induction $(\mathcal{F}_6)$.

Le système de transition $\langle S, A, t, \varepsilon \rangle$ correspondant à ce programme est défini par :

- $S = \{start, Loop, Finish\} \times arbre \times nat \times pile$
- $A = \{a\}$
- $t_a(\langle \ell, tr, co, st \rangle, \langle \ell', tr', co', st' \rangle) =$

$$\begin{aligned}[& (\ell = start \wedge \ell' = Loop \wedge tr' = tr \wedge co' = 0 \wedge st' = ( )) \\ \vee & (\ell = Loop \wedge tr \neq mil \wedge \ell' = loop \wedge tr' = ff(Tr) \wedge co' = co \wedge st' = (tr . st)) \\ \vee & (\ell = Loop \wedge tr = mil \wedge st \neq ( ) \wedge \ell' = Loop \wedge tr' = rg(hd(st)) \wedge co' = co + 1 \wedge st' = tl(st)) \\ \vee & (\ell = Loop \wedge tr = mil \wedge st = ( ) \wedge \ell' = Finish \wedge tr' = tr \wedge co' = co + 1 \wedge st' = st)]\end{aligned}$$

- $\varepsilon(\langle \ell, tr, co, st \rangle) = [\ell = start]$

La correction totale de ce programme peut être spécifiée par la fatalité de ψ pour $\langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle$, telle que :

$$\psi(\langle \ell, tr, co, st \rangle, \langle \ell', tr', co', st' \rangle) = [\ell' = Finish \wedge co' = tips(tr)]$$

où

$$tips(tr) = (tr = nil \rightarrow 1 \mid (tips(lf(tr)) + tips(rg(tr))))$$

- La correction partielle de ce programme consiste d'abord à découvrir des assertions intermédiaires associées aux points de contrôle du programme. Ces assertions expriment les relations qu'on s'attend à trouver entre les valeurs initiales $\underline{tr}, \underline{co}, \underline{st}$ des variables Tr, Co, St et leurs valeurs tr, co, st quand le contrôle est en ces points :

$$I_{start}(\underline{tr}, \underline{co}, \underline{st}, tr, co, st) = [tr = \underline{tr} \wedge co = \underline{co} \wedge st = \underline{st}]$$

$$I_{Loop}(\underline{tr}, \underline{co}, \underline{st}, tr, co, st) = [(tips(tr) + co + Sum(tips \circ rg, st)) = tips(\underline{tr})]$$

$$I_{Finish}(\underline{tr}, \underline{co}, \underline{st}, tr, co, st) = [co = tips(\underline{tr})]$$

où

$fog(x) = f(g(x))$ et si $f \in (arbre \rightarrow \omega)$ et st est une pile alors

$$Sum(f, st) = (st = () \rightarrow 0 \mid (f(hd(st)) + Sum(f, tl(st))))$$

Puis on montre que ces assertions sont invariantes, c'est-à-dire :

- I_{start} est initialement vraie avec $tr = \underline{tr}$, $co = \underline{co}$ et $st = \underline{st}$
- si l'assertion intermédiaire associée au point de contrôle ℓ du programme est vraie et que le contrôle passe du point ℓ au point ℓ', alors l'assertion associée à ℓ' doit être vraie :

$$I_{start}(\underline{tr}, \underline{co}, \underline{st}, tr, co, st) \Rightarrow I_{Loop}(\underline{tr}, \underline{co}, \underline{st}, tr, 0, ())$$

$$[I_{Loop}(\underline{tr}, \underline{co}, \underline{st}, tr, co, st) \wedge tr \neq nil] \Rightarrow I_{Loop}(\underline{tr}, \underline{co}, \underline{st}, \gamma_f(tr), co, (tr, st))$$

$$[I_{Loop}(\underline{tr}, \underline{co}, \underline{st}, tr, co, st) \wedge tr = nil \wedge st \neq ()] \Rightarrow I_{Loop}(\underline{tr}, \underline{co}, \underline{st}, rg(hd(st)), co+1, tl(st))$$

$$[I_{Loop}(\underline{tr}, \underline{co}, \underline{st}, tr, co, st) \wedge tr = nil \wedge st = ()] \Rightarrow I_{Finish}(\underline{tr}, \underline{co}, \underline{st}, tr, co+1, st)$$

Finalement, l'assertion associée au point de sortie doit impliquer

- D'après Floyd [67], "proofs of termination are dealt with by showing that each step of the program decreases some entity which cannot decrease infinitely". Nous associons donc à tout point de contrôle ℓ du programme, une fonction W_ℓ des valeurs tr, co, st des variables Tr, Co, St du programme à résultat dans le bon-ordre $\langle \omega, < \rangle$:

$$W_{start}(tr, co, st) = size(tr) + 1$$

$$W_{Loop}(tr, co, st) = size(tr) + Sum(size \circ rg, st)$$

$$W_{Finish}(tr, co, st) = 0$$

où

$$size(tr) = (tr = nil \rightarrow 1 \mid 1 + size(lf(tr)) + size(rg(tr)))$$

et montrons qu'après chaque exécution d'une commande, la valeur courante de la fonction $W_{\ell'}$ associée à son point de "sortie" ℓ' est inférieure à la valeur antérieure de la fonction W_ℓ associée à son point d'entrée ℓ :

$$[tr' = tr \wedge co' = 0 \wedge st' = ()] \Rightarrow (W_{start}(tr, co, st) > W_{Loop}(tr', co', st'))$$

$$[tr \neq nil \wedge tr' = lf(tr) \wedge co' = co \wedge st' = (tr, st)] \Rightarrow (W_{Loop}(tr, co, st) > W_{Loop}(tr', co', st'))$$

$$[tr = nil \wedge st \neq () \wedge tr' = rg(hd(st)) \wedge co' = co+1 \wedge st' = tl(st)] \Rightarrow (W_{Loop}(tr, co, st) > W_{Loop}(tr', co', st'))$$

$$[tr = nil \wedge st = () \wedge tr' = tr \wedge co' = co+1 \wedge st' = st] \Rightarrow (W_{Loop}(tr, co, st) > W_{Finish}(tr', co', st'))$$

- En choisissant :

$\Gamma = \omega$

$$I_\ell(\gamma, \langle \ell, \underline{tr}, \underline{co}, \underline{st} \rangle, \langle \ell, tr, co, st \rangle) = [(\ell = start) \ (\gamma = W_\ell(tr, co, st) \wedge I_\ell(\underline{tr}, \underline{co}, \underline{st}, tr, co, st))]$$

les conditions de vérification de Floyd sont équivalentes à $(\mathcal{P}_c)$ (puisque ce programme est total), les vérifications d'absence d'erreurs à l'exécution :

$$\forall \ell = \{start, Loop\} . [I_\ell(\underline{tr}, \underline{co}, \underline{st}, tr, co, st) \Rightarrow \exists \ell', tr', co', st' . t(\langle \ell, tr, co, st \rangle, \langle \ell', tr', co', st' \rangle)]$$

n'ont pas lieu d'être).

□

5.2.4 CORRECTION ET COMPLETUDE SEMANTIQUE DES PRINCIPES D'INDUCTION "A LA FLOYD" POUR LES SEMANTIQUES CLOSES

Ψ est fatale sous invariance de Φ pour $\langle S, A, \Sigma\langle S,A,t,\varepsilon\rangle\rangle$ si et seulement si un des principes d'induction est applicable.

Théorème 5.2.4 N1 (Correction)

$(\mathcal{F}_1) \rightarrow$ (Ψ est fatale sous invariance de Φ pour $\langle S, A, \Sigma\langle S,A,t,\varepsilon\rangle\rangle$)

Démonstration

Supposons par l'absurde qu'il existe $p \in \Sigma\langle S,A,t,\varepsilon\rangle$ tel que $\forall i \in |p|.[(\forall j \in i.\Phi(p_0,p_j)) \Rightarrow \neg\Psi(p_0,p_i)]$. Alors par induction sur i, $(\mathcal{F}_2')$ implique $\forall i \in |p|.[(\forall j \in (i+1).\Phi(p_0,p_j)) \wedge J_1(p_0,p_i)]$. Si $\exists m \in (\omega \setminus 0).\ p \in \Sigma^m\langle S,A,t,\varepsilon\rangle$ alors $J_1(p_0,p_{m-1})$, $\neg\Psi(p_0,p_{m-1})$ et $(\mathcal{F}_3')$ impliquent $\exists A' \in S, a \in A.\ t_a(p_{m-1},A')$, une contradiction. Sinon $p \in \Sigma^\omega\langle S,A,t,\varepsilon\rangle$ de sorte que pour tout $i \in \omega$ nous avons $J_1(p_0,p_i)$ et $t_{\#_i}(p_i,p_{i+1})$ et donc d'après $(\mathcal{F}_4')$ que $f_1(p_0,p_i) \in W_1$, $f_1(p_0,p_{i+1}) \in W_1$ et $f_1(p_0,p_{i+1}) \prec_1 f_1(p_0,p_i)$ en contradiction avec $\underset{\sim}{\omega o}(W_1,\prec_1)$.

□

Théorème 5.2.4 N2 (Complétude sémantique)

(Ψ est fatale sous invariance Φ pour $\langle S, A, \Sigma\langle S,A,t,\varepsilon\rangle\rangle$) $\Rightarrow (\mathcal{F}_4)$

Démonstration :

Définissons W et $\prec \in \wp(W^2 \rightarrow \{t,f\})$ telle que :

- $W = \{\langle \underline{a}, A\rangle \in S^2 : \exists p \in \Sigma\langle S,A,t,\varepsilon\rangle.\ \underline{a} \in |p|.\ \exists i \in |p|.$
$[\forall j \in i.(\Phi(p_0,p_j) \wedge \neg\Psi(p_0,p_j)) \wedge \Psi(p_0,p_i) \wedge \underline{a} = p_0 \wedge A = p_i]\}$
- $\langle \underline{a}', A'\rangle \prec \langle \underline{a}, A\rangle \Leftrightarrow (\exists p \in \Sigma\langle S,A,t,\varepsilon\rangle, i \in |p|.\ k \in \omega.[\forall j \in i.(\Phi(p_0,p_j) \wedge \neg\Psi(p_0,p_j)) \wedge$

Puisque $\langle \underline{a}', A'\rangle \prec \langle \underline{a}, A\rangle$ implique $(\underline{a}' = A \wedge \Phi(\underline{a}, A) \wedge \neg\Psi(\underline{a}, A) \wedge \exists a \in A.\ t_a(A, A')$ $\wedge\ \Phi(\underline{a}, A') \wedge \neg\Psi(\underline{a}, A')$ nous avons $\underset{\sim}{\omega f}(W, \prec)$. (sinon il y aurait eu une chaîne $\langle \underline{a}, A_0\rangle \prec \langle \underline{a}, A_1\rangle \prec \ldots$ et donc une trace infinie $\underline{a}, p_1, \ldots, p_{k+1}, A_0, A_1, \ldots$ dont tous les états A satisfont $\Phi(\underline{a}, A) \wedge \neg\Psi(\underline{a}, A)$ en contradiction avec le fait que Ψ est fatale sous invariance de Φ pour $\langle S, A, \Sigma\langle S,A,t,\varepsilon\rangle\rangle$).

Définissons maintenant $\underset{\sim}{Rng}(f_4) = (W \cup \{\langle \underline{a}, A\rangle : \Psi(\underline{a}, A)\})$, $f_4(\underline{a}, A) = \langle \underline{a}, A\rangle$ et $\langle \underline{a}', A'\rangle \prec_4 \langle \underline{a}, A\rangle$ $\Leftrightarrow [(\Psi(\underline{a}', A') \wedge \underline{a}' = \underline{a} \wedge \langle \underline{a}, A\rangle \in W) \vee (\langle \underline{a}', A'\rangle \in W \wedge \langle \underline{a}, A\rangle \in W \wedge \langle \underline{a}', A'\rangle \prec \langle \underline{a}, A\rangle)]$. Nous avons $\underset{\sim}{\omega f}(\underset{\sim}{Rng}(f_4), \prec_4)$ de sorte qu'en choisissant $\tau_4(\underline{a}, A) = [\langle \underline{a}, A\rangle \in \underset{\sim}{Rng}(f_4)]$, les conditions de vérification de $(\mathcal{F}_4)$ sont vérifiées.

□

5.2.5 SUR L'UTILISATION D'HYPOTHESES d'INDUCTION ASSERTIONNELLES OU RELATIONNELLES

Si nous voulons démontrer que l'assertion $\Psi(\underline{s},\underline{A}) = \Psi(\underline{A})$ est fatale vis à vis de l'invariance de $\Phi(\underline{s},\underline{A}) = \phi(\underline{A})$ pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$, les hypothèses d'induction J_i dans les principes d'induction $(\mathcal{F}_2^i)$, $i=1,\dots,7$, peuvent ne pas dépendre des états initiaux. Dans ce cas, le choix des f_i dans $(\mathcal{F}_2^i)$, $i=1,\dots,4$ comme une fonction ne dépendant pas des états initiaux est également sémantiquement complet (si en plus $\phi(\underline{A})=\Bbbk$, $\Psi(\underline{A}) = [\forall \underline{s}' \in S, a \in A, \neg t_a(\underline{A},\underline{s}')]$, $\langle \underset{\sim}{\mathrm{Rng}}(f_2), \prec_2\rangle = \langle \underset{\sim}{\mathrm{Ord}}, <\rangle$ alors le principe d'induction $(\mathcal{F}_2^4)$ donne la méthode de preuve de terminaison de Lehmann-Pnueli-Stavi [81].

Théorème 5.2.5 v1

En général, Ψ est une relation entre les états initiaux et finaux, et dans ce cas l'hypothèse d'induction doit être une relation (reliant les états initiaux et courants) pour garantir la complétude sémantique.

Démonstration

Considérons $S=\{0,1,2\}$, $A=\{a\}$, $\varepsilon(\underline{s}) = [0 \le \underline{s} \le 1]$, $t_a(\underline{s},\underline{s}') = [(\underline{s}+1 \in S) \wedge (\underline{s}'=\underline{s}+1)]$, $\Phi(\underline{s},\underline{A}) = \Bbbk$. Alors $\Psi = t_a$ est fatale de manière évidente. Supposons qu'on puisse trouver J_1 de la forme $J_1(\underline{s},\underline{A}) = I(\underline{A})$ dans $(\mathcal{F}_2^1)$. Alors $\varepsilon(1) \Rightarrow J_1(1,1)$. Puisque $\neg\Psi(1,1)$ et $t_a(1,2)$ alors nous aurons $J_1(1,2) = I(2) = J_1(0,2)$. Mais $\neg\Psi(0,2)$ et $\forall \underline{s} \in S, a \in A. \neg t_a(\underline{s},\underline{s}')$, d'où contradiction

□

Théorème 5.2.5 v2

De la même manière, dans le principe d'induction $(\mathcal{F}_2^i)$, $i=1,\dots,4$ on choisit comme une fonction unaire ne dépendant pas

Démonstration

Considérons $S=\omega$, $A=\{a\}$, $\varepsilon(\underline{s})=\Bbbk$, $t_a(\underline{s},\underline{s}') = [\underline{s}'=\underline{s}+1]$ et $\Phi(\underline{s},\underline{A}) = \Bbbk$. Alors de manière évidente, $\Psi(\underline{s},\underline{s}) = [\underline{s}=\underline{s}+1]$ est fatale. Mais $(\mathcal{F}_2^1)$ ne peut être appliqué avec f_1 de la forme $f_1(\underline{s},\underline{s}) = f(\underline{s})$. Par l'absurde, nous aurions $\forall s \in \omega. (\varepsilon(s) \Rightarrow J_1(s,s))$ de sorte que $J_1(s,s) \wedge \neg\Psi(s,s) \wedge t_a(s,s+1)$ implique $J_1(s,s) \in W_1$ et $f_1(s,s+1) \prec_1 f_1(s,s)$ c'est-à-dire $f(s) \in W_1$ et $f(s+1) \prec_1 f(s)$. en contradiction avec $\underset{\sim}{\mathrm{wo}}(W_1,\prec_1)$.

□

Par conséquent, la généralisation de la méthode de preuve de la correction partielle de Hoare[69] à la correction totale proposée par Manna-Pnueli[74] (pour laquelle la fonction de terminaison dans les boucles ne porte que sur l'état courant des variables dans la boucle sans liaison possible avec l'état initial des variables à l'entrée de la boucle) n'est pas (sémantiquement) complète. Ceci peut se corriger de la manière suivante:

Si on tient à utiliser des principes d'induction assertionnels et non relationnels, on pourra utiliser l'artifice bien connu qui consiste à utiliser des variables auxiliaires dans le programme.

Les propriétés fatales pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ peuvent toujours aisément être démontrées en raisonnant sur $\langle S',A',\Sigma\langle S',A',t',\varepsilon'\rangle\rangle$ tel que:

$S' = S \times S$

$A' = A$

$t'_a(\langle\underline{s},\underline{s}'\rangle,\langle\underline{s}'',\underline{s}'''\rangle) = [\underline{s}=\underline{s}'' \wedge t_a(\underline{s}',\underline{s}''')]$

$\varepsilon'(\langle\underline{s},\underline{s}'\rangle) = [\varepsilon(\underline{s}) \wedge \underline{s}=\underline{s}']$

$\forall p \in \Sigma\langle S,A,t,\varepsilon\rangle. \exists i \in |p|. [(\forall j \in i. \bar{\Phi}(p_0 \dots p_j)) \wedge \Psi(p_0,p_i)]$

$\Leftrightarrow$

$\forall p' \in \Sigma\langle S',A',t',\varepsilon'\rangle. \exists i \in |p'|. [(\forall j \in i. \bar{\Phi}(p'_j)) \wedge \Psi(p'_i)]$

Dans ce cas, il nous semble que l'emploi de variables auxiliaires dans les preuves ne doit pas être présenté comme une transformation de programme mais plutôt comme l'utilisation d'un principe d'induction différent. Ceci parce que les variables auxiliaires sont plus simples à introduire dans les preuves que dans les programmes (qui ont une syntaxe rigide). En outre, ceci permet de raisonner sur les méthodes de preuve de programmes qui sont indépendantes des langages de programmation.

5.2.6 SUR LE NON-DETERMINISME BORNE

Floyd [67] a noté qu'il peut être nécessaire d'utiliser d'autres bons-ordres que l'ensemble $\langle \omega, < \rangle$ des nombres naturels pour les preuves de terminaison. Dijkstra [76, p.77] a donné le contre-exemple $S = \mathbb{Z}$, $A = \{a\}$, $t_a(x,x') = [(x<0 \wedge x'>0) \vee (x>0 \wedge x'=x-1)]$, $\varepsilon(x) = [x<0]$, $\phi(x,x) = tt$ et $\psi(x,\bar{x}) = [\bar{x}=0]$ pour lequel le nombre de transitions requises pour la terminaison n'a pas de borne supérieure finie. Dijkstra a aussi montré que quand le non-déterminisme est fini, on peut toujours faire les preuves de terminaison en utilisant $\langle \omega, < \rangle$.

5.2.6.1 Non-déterminisme m-borné

Un système de transition $\langle S, A, t, \varepsilon \rangle$ est dit <u>déterministe</u> si $\forall s \in S. [\underline{card}(\{s' \in S : \exists a \in A. t_a(s,s')\}) \leq 1]$ et <u>non déterministe</u> sinon.

Le non déterminisme est dit <u>fini</u> (Dijkstra dit borné) si $\forall s \in S. \exists m \in \omega. [\underline{card}(\{s' \in S : \exists a \in A. t_a(s,s')\}) \leq m]$ ou, et ceci est équivalent, si $\forall s \in S. [\underline{card}(\{s' \in S : \exists a \in A. t_a(s,s')\}) < \omega]$ et <u>infini</u> sinon.

Le non déterminisme est dit <u>dénombrable</u> si $\forall s \in S. [\underline{card}(\{s' \in S : \exists a \in A. t_a(s,s')\}) \leq \omega]$ et <u>non dénombrable</u> sinon.

Plus généralement, si $m \in \underline{ord}$ est un cardinal, alors nous disons que le non déterminisme d'un système de transition $\langle S, A, t, \varepsilon \rangle$ est <u>m-borné</u> si $\forall s \in S. [\underline{card}(\{s' \in S : \exists a \in A. t_a(s,s')\}) < m]$.

(En particulier, le non déterminisme de $\langle S, A, t, \varepsilon \rangle$ est toujours <u>S^+-borné</u>. En outre, le non déterminisme est dénombrable si et seulement si il est ω_1-borné où ω_1 est le plus petit cardinal strictement plus grand que ω).

5.2.6.2 La fatalité peut être démontrée à l'aide du bon-ordre $<m^+,<>$ quand le non-déterminisme est m-borné

Le principe d'induction suivant, pour démontrer les propriétés de fatalité de sémantiques $<S,A,\Sigma<S,A,t,\varepsilon>>$ avec non-déterminisme m-borné, est correct (puisque $(\mathcal{P}_0') \Rightarrow (\mathcal{P}_0'')$) et sémantiquement complet :

$$
\begin{array}{l}
(\exists J \in (m^+ \times S \times S \to \{tt,ff\}). \\
\quad [\forall \underline{s},\underline{s}',\underline{s}'' \in S,\ \gamma < m^+. \\
\quad\quad (a) \quad (\varepsilon(\underline{s}) \Rightarrow [\exists \underline{\gamma} < m^+.\ J(\underline{\gamma},\underline{s},\underline{s})]) \\
\quad\quad (b) \quad \wedge\ ([J(\gamma,\underline{s},\underline{s}) \wedge \gamma > 0] \Rightarrow [\Phi(\underline{s},\underline{s}) \wedge \exists \underline{s}' \in S, a \in A.\ t_a(\underline{s},\underline{s}')]) \\
\quad\quad (c) \quad \wedge\ ([J(\gamma,\underline{s},\underline{s}) \wedge \gamma > 0 \wedge t_a(\underline{s},\underline{s}')] \Rightarrow [\exists \gamma' < \gamma.\ J(\gamma',\underline{s},\underline{s}')]) \\
\quad\quad (d) \quad \wedge\ (J(0,\underline{s},\underline{s}) \Rightarrow \Psi(\underline{s},\underline{s}))\ ])
\end{array}
\qquad (\mathcal{P}_0'')
$$

où $m^+ = \omega$ si $m < \omega$, $m^+ = m$ si m est un cardinal infini régulier et $m^+ = m^+$ si m est un cardinal infini singulier.

Pour démontrer la complétude sémantique, nous introduisons quelques définitions que nous utiliserons par la suite.

Nous dirons que :

- un état $\underline{s}$ est intermédiaire pour $<S,A,t,\varepsilon>$ lorsqu'il existe une trace d'exécution passant par $\underline{s}$ pour laquelle Ψ n'est pas vraie jusqu'à $\underline{s}$ compris
- un état $\underline{s}$ est un but ("goal") pour $<S,A,t,\varepsilon>$ lorsqu'il existe une trace d'exécution passant par $\underline{s}$ pour laquelle Ψ est vraie pour la première fois sur un état
- un état $\underline{s}$ est accessible pour $<S,A,t,\varepsilon>$ lorsque $\underline{s}$ est un état intermédiaire ou un but.

Définition 5.2.6.2-1 (Etats intermédiaires, buts et accessibles)

- $\underline{\text{Inter}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s}) = \{\underline{s} \in S :\ \exists p \in \Sigma<S,A,t,\varepsilon>, i \in |p|.$
  $(p_0 = \underline{s} \wedge \forall j < i.\ (\Phi(p_0,p_j) \wedge \neg\Psi(p_0,p_j)) \wedge p_i = \underline{s})\}$
- $\underline{\text{Inter}}<S,A,t,\varepsilon,\Phi,\Psi> = \cup\{\underline{\text{Inter}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s}) :\ \underline{s} \in S\}$
- $\underline{\text{Goal}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s}) = \{\underline{s} \in S :\ \exists p \in \Sigma<S,A,t,\varepsilon>, i \in |p|.$
  $(p_0 = \underline{s} \wedge \forall j < i.\ (\Phi(p_0,p_j) \wedge \neg\Psi(p_0,p_j)) \wedge p_i = \underline{s} \wedge \Psi(p_0,p_i))\}$
- $\underline{\text{Goal}}<S,A,t,\varepsilon,\Phi,\Psi> = \cup\{\underline{\text{Goal}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s}) :\ \underline{s} \in S\}$
- $\underline{\text{Acc}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s}) = \underline{\text{Inter}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s}) \cup \underline{\text{Goal}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s})$
- $\underline{\text{Acc}}<S,A,t,\varepsilon,\Phi,\Psi> = \underline{\text{Inter}}<S,A,t,\varepsilon,\Phi,\Psi> \cup \underline{\text{Goal}}<S,A,t,\varepsilon,\Phi,\Psi>$

Théorème 5.2.6.2-1 (Complétude sémantique)

(Ψ est fatale sous invariance de Φ pour $<S,A,\Sigma<S,A,t,\varepsilon>>$) $\to (\mathcal{P}_0'')$

Démonstration

Pour tous $\underline{s} \in S$ nous définissons :

- $I(\underline{s}) = \{\underline{s} \in S :\ \underline{\text{Inter}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s})\}$
- $G(\underline{s}) = \{\underline{s} \in S :\ \underline{\text{Goal}}<S,A,t,\varepsilon,\Phi,\Psi>(\underline{s})\}$
- $A(\underline{s}) = I(\underline{s}) \cup G(\underline{s})$
- $\prec_{\underline{s}} \in (S \times S \to \{tt,ff\})$, telle que $\underline{s}' \prec_{\underline{s}} \underline{s}$ si et seulement si $[\underline{s} \in I(\underline{s}) \wedge \exists a \in A.\ t_a(\underline{s},\underline{s}')]$

Nous démontrons d'abord que $\forall \underline{s} \in S.\ \text{wf}(A(\underline{s}), \prec_{\underline{s}})$.

Si $\neg\varepsilon(\underline{s})$ alors $A(\underline{s}) = \emptyset$ et toute relation est bien fondée sur l'ensemble vide $\emptyset$.

Sinon, par l'absurde, supposons qu'il existe une séquence infinie q d'états dans $A(\underline{s})$ telle que $\forall i \in \omega.\ (q_{i+1} \prec_{\underline{s}} q_i)$. Si $\underline{s} \in G(\underline{s})$ alors $\underline{s} \notin I(\underline{s})$ de sorte que $\forall \underline{s}' \in S.\ \neg(\underline{s}' \prec_{\underline{s}} \underline{s})$. Donc q_1 ne peut appartenir à $G(\underline{s})$. En particulier, puisque $q_0 \in I(\underline{s})$ nous pouvons supposer que $q_0 = \underline{s}$ (sinon il existe un préfixe $\ell = p_0, \ldots p_k = q_0$ d'une certaine trace $p \in \Sigma<S,A,t,\varepsilon>$ avec $p_{j+1} \prec_{\underline{s}} p_j$ pour $j \in k$, qui peut être adjoint à la gauche de q, puisque $\varepsilon(q_0)$). En outre $\forall i \in \omega.\ (q_{i+1} \prec_{\underline{s}} q_i)$, d'où $t_{a_i}(q_i, q_{i+1})$. Il s'ensuit que $a \in \Sigma<S,A,t,\varepsilon>$.

Mais Vieu nous avons $q_i \in I(q_0)$ et donc $\Phi(q_0,q_i) \wedge \neg\Psi(q_0,q_i)$ en contradiction avec l'hypothèse de fatalité.

Nous démontrons maintenant que $\forall \underline{a}, a \in S. (\underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a)) < m^+)$

La preuve est par induction transfinie sur la relation $\prec_{\underline{a}}$ bien fondée sur $A(\underline{a})$.

Si $\{a' \in S : a' \prec_{\underline{a}} a\}$ est vide alors $rk(A(\underline{a}), \prec_{\underline{a}})(a) = 0$ donc $\underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a)) = 0 < \omega \leq m^+$.

Autrement $a' \prec_{\underline{a}} a$ implique $\exists a \in A. t_{\underline{a}}(a,a')$ donc $\underline{card}(\{a' \in S : a' \prec_{\underline{a}} a\}) \leq \underline{card}(\{a' \in S : \exists a \in A. t_{\underline{a}}(a,a')\})$

$< m \leq m^+$ et $\underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a')) < m^+$ par hypothèse d'induction. Donc ou bien

$\underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a')) < \omega$ et $\underline{card}(\mathcal{P}(rk(A(\underline{a}), \prec_{\underline{a}})(a'))) = \mathcal{P}(\underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a'))) < \omega \leq m^+$ ou

$\underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a')) > \omega$ auquel cas $\underline{card}(\mathcal{P}(rk(A(\underline{a}), \prec_{\underline{a}})(a'))) = \underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a')) < m^+$.

Si m est un cardinal infini régulier, alors pour tout système $\langle \mu_i : i \in I \rangle$ de cardinaux avec $\mu_i < m$ pour tout $i \in I$ et $\underline{card}(I) < m$, nous avons $\bigcup_{i \in I} \mu_i < m$. D'où nous concluons que $\underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a)) = \underline{card}(\bigcup\{rk(A(\underline{a}), \prec_{\underline{a}})(a')+1 : a' \prec_{\underline{a}} a\}) < m^+$.

$\underline{card}(rk(A(\underline{a}), \prec_{\underline{a}})(a)) = \underline{card}(\sup^+\{rk(A(\underline{a}), \prec_{\underline{a}})(a') : a' \prec_{\underline{a}} a\}) = \underline{card}(\sup\{\mathcal{P}(rk(A(\underline{a}), \prec_{\underline{a}})(a')) : a' \prec_{\underline{a}} a\})$

$< m^+$.

Finalement nous définissons J_δ tel que $J_\delta(\delta, \underline{a}, a) = [a \in A(\underline{a}) \wedge \delta = rk(A(\underline{a}), \prec_{\underline{a}})(a)]$

Nous avons $J_\delta \in (m^+ \times S \times S \to \{tt, ff\})$ et les conditions de vérification de $(\mathcal{F}'_\delta)$ sont satisfaites

(a) $\varepsilon(\underline{a}) \Rightarrow [a \in A(\underline{a})] \Rightarrow [\exists \delta < m^+. J_\delta(\delta, \underline{a}, a)]$.

(b) Si $[J_\delta(\delta, \underline{a}, a) \wedge \delta > 0]$ alors $rk(A(\underline{a}), \prec_{\underline{a}})(a) > 0$ de sorte qu'il existe a' tel que $a' \prec_{\underline{a}} a$. Ceci implique $\exists a \in A. t_a(a,a')$ et $a \in I(\underline{a})$ et donc $\Phi(\underline{a}, a)$.

(c) Si $[J_\delta(\delta, \underline{a}, a) \wedge \delta > 0 \wedge t_a(a,a')]$ alors $a \in I(\underline{a})$ de sorte que d'après l'hypothèse de fatalité, nous devons avoir $a' \in A(\underline{a})$ et $a' \prec_{\underline{a}} a$ et donc $rk(A(\underline{a}), \prec_{\underline{a}})(a') < rk(A(\underline{a}), \prec_{\underline{a}})(a)$ et $J_\delta(\delta', \underline{a}, a')$

(d) $J_\delta(0, \underline{a}, a) \Rightarrow (rk(A(\underline{a}), \prec_{\underline{a}})(a) = 0) \Rightarrow a \in G(\underline{a}) \Rightarrow \Psi(\underline{a}, a)$.

□

En particulier, pour démontrer les propriétés de fatalité de sémantiques $\langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle$, on sait choisir des bons-ordres isomorphes à $\langle \omega, < \rangle$ quand le non-déterminisme de $\langle S, A, t, \varepsilon \rangle$ est fini et des bons-ordres isomorphes à $\langle \omega_1, < \rangle$ quand le non-déterminisme de $\langle S, A, t, \varepsilon \rangle$ est dénombrable, (ω et $\omega_1 = \omega^+$ sont réguliers de sorte que $\omega^+ = \omega$ et $\omega_1^+ = \omega_1$).

5.2.6.3 Quels ordinaux sont nécessaires ?

Théorème 5.8.6.3 v1

Soient m un cardinal régulier fini ou infini et $\langle S, A, t, \varepsilon \rangle$ un système de transition dont le non-déterminisme est m-borné. Supposons que nous voulions démontrer que ψ est fatale sous invariance de Φ pour $\langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle$ en utilisant le principe d'induction $(\mathcal{F}'_\delta)$ avec $\Gamma < m^+ = m$. Ceci n'est pas complet.

Démonstration

C'est évident quand $m = \omega$ et Γ est un nombre naturel, aussi nous pouvons supposer $\Gamma \geq \omega$.

Définissons $S = (\{\perp\} \cup \{\Gamma + 1\})$ où $\perp \notin (\Gamma + 1)$, $A = \{\underline{a}\}$. Nous avons $\underline{card}(S) = (1 + \underline{card}(\Gamma + 1)) = \underline{card}(\Gamma + 1) = \underline{card}(\Gamma) \leq \Gamma < m^+ = m$. Ainsi le non-déterminisme de $\langle S, A, t, \varepsilon \rangle$ est m-borné. Définissons $t_{\underline{a}}(x, x') = [(x = \perp \wedge x' < \Gamma) \vee (0 < x' < x < \Gamma)]$, $\varepsilon(\underline{x}) = [x = \perp]$, $\Phi(x, \tilde{x}) = tt$ et $\psi(x, \tilde{x}) = [\tilde{x} = 0]$.

Une trace d'exécution $p \in \Sigma \langle S, A, t, \varepsilon \rangle$ est telle que $p_0 = \perp$, $p_i \in \underline{Ord}$, $i \in (\omega \cup 0)$ et $p_{i+1} < p_i$, de sorte que nous devons avoir fatalement un $p_i = 0$ puisque $\omega_0(\underline{ord}, <)$. Ainsi ψ est fatale sous invariance de Φ pour $\langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle \rangle$.

Cependant, ceci ne peut être démontré au moyen de $(\mathcal{F}'_\delta)$. Si nous trouvons J_δ satisfaisant les conditions de vérification de $(\mathcal{F}'_\delta)$, nous pouvons construire une séquence infinie strictement décroissante $x_0 > x_1 > \ldots$ d'ordinaux comme suit : Posons $\alpha_0 = \Gamma$. Puisque $\varepsilon(\perp)$, nous devons avoir un δ_0 tel que $J_{\delta_0}(\delta_0, \perp, \perp)$. Posons $x_1 = \delta_0$. Puisque $\neg\Psi(\perp, \perp)$, nous avons $\neg J_0(0, \perp, \perp)$

et donc $\alpha_1 > 0$. Mais $[\alpha_1 > 0 \wedge J_6(\alpha_1, \perp, \perp) \wedge t_{\underline{2}}(\perp, \alpha_1)] \Rightarrow [\exists \delta < \alpha_1 . J_6(\delta, \perp, \alpha_0)]$. Posons $\alpha_2 = \delta$. Nous avons $\Gamma = \alpha_0 > \alpha_1 > \alpha_2$ et $J_6(\alpha_2, \perp, \alpha_0)$. Puisque $\alpha_0 > 0$, nous avons $\neg \Psi(\perp, \alpha_0)$ et donc $\neg J_6(0, \perp, \alpha_0)$ et $\alpha_2 \neq 0$. Supposons que nous ayons construit la séquence jusqu'en α_{j+2} avec $\beta \geq \alpha_j > \alpha_{j+1} > \alpha_{j+2} > 0$ et $J_6(\alpha_{j+2}, \perp, \alpha_j)$. D'après $(\mathcal{F}_6')$ nous avons $[\alpha_{j+2} > 0 \wedge J_6(\alpha_{j+2}, \perp, \alpha_j) \wedge t_{\underline{2}}(\alpha_j, \alpha_{j+1})] \Rightarrow [\exists \delta < \alpha_{j+2} . J_6(\delta, \perp, \alpha_{j+1})]$. Posons $\alpha_{j+3} = \delta$. Puisque $\alpha_{j+1} > 0$ nous avons $\neg \Psi(\perp, \alpha_{j+1})$ de sorte que $\beta \geq \alpha_{j+1} > \alpha_{j+2} > \alpha_{j+3} > 0$ et $J_6(\alpha_{j+3}, \perp, \alpha_{j+1})$. Et ainsi la séquence peut être prolongée indéfiniment.

□

Bienque ceci soit restreint aux cardinaux réguliers, le résultat est très général puisque le premier cardinal singulier infini est $\omega_\omega = \sup_{i \in \omega} \omega_i$ (qui est trop grand pour avoir un intérêt quelconque en informatique).

Un cas particulier intéressant est celui de la méthode de Knuth [68], Luckham-Suzuki [75] pour montrer la terminaison qui consiste à utiliser un compteur par boucle du programme, qui est strictement incrémenté à chaque itération dans la boucle et dont la valeur est bornée (nous utiliserons, ce qui revient au même, un seul compteur δ incrémenté à chaque pas du programme et borné par $\beta \in \omega$) :

$(\mathcal{F}_{\hat{k}})$

$$
\begin{aligned}
&(\exists \beta \in \omega,\ J \in (\omega \times S \times S \to \{tt, ff\}). \\
&\quad (\forall \perp \in S.\ \exists \delta \in \omega.\ J(\delta, \perp, \perp)) \\
&\quad \wedge (\forall \perp, \perp' \in S,\ \delta' \in \omega. \\
&\qquad J(\delta', \perp, \perp') \Rightarrow [\ (\delta' < \beta \ \wedge\ \Phi(\perp, \perp') \ \wedge \\
&\qquad\quad \exists \perp'' \in S, \alpha \in \beta.\ t_0(\perp, \perp'', \alpha) \\
&\qquad\quad \wedge \forall \perp'' \in S, \gamma \in \beta.\ [t_0(\perp, \perp'', \gamma) \Rightarrow \exists \delta'' < \delta'.\ J(\delta'', \perp'', \perp')] \\
&\qquad \vee \Psi(\perp, \perp')]))
\end{aligned}
$$

La méthode est correcte car on retrouve $(\mathcal{F}_6)$ avec $\Gamma_6 = (\beta \to \omega)$ et $J_6(\delta, \perp, \perp') = J_{\hat{k}}(\beta - \delta, \perp, \perp')$. Elle est évidemment sémantiquement complète quand le non-déterminisme est fini (car on retrouve alors $(\mathcal{F}_\delta')$). Le résultat ci-dessus montre que cette méthode n'est pas sémantiquement complète quand le non-déterminisme n'est pas fini, ce qui explique qu'elle n'ait pas pu être généralisée (par exemple au cas des programmes parallèles équitables).

5.2.7 DECOMPOSITION DES CONDITIONS DE VERIFICATION

La méthode de construction d'une méthode de preuve d'invariance à partir d'une sémantique opérationnelle et d'un principe d'induction par décomposition de l'hypothèse d'induction globale en hypothèses d'induction locales proposée en 4.3 est évidemment directement applicable aux preuves de fatalité et nous n'y reviendrons pas. La plupart du temps, cette décomposition des conditions de vérification s'obtient par décomposition de l'ensemble des états ou des actions du système de transition (Cousot: P[81]) nous en donnerons quelques exemples.

5.2.7.1 Décomposition des conditions de vérification au moyen d'un recouvrement de l'ensemble des états du système de transition

Un recouvrement de l'ensemble des états d'un système de transition $\langle S, A, t, \varepsilon\rangle$ est une paire $\langle r, \pi\rangle$ telle que :

- r est un ensemble fini non vide de noms de blocs
- $\pi \in (r \to (S \to \{tt, ff\}))$ caractérise un recouvrement de la classe s des états par :

$\forall s \in S.\ \exists k \in r.\ \pi_k(s)$

(La classe s des états peut être décomposée en donnant des noms aux blocs d'états jouant des rôles similaires. Par exemple un π_k sera vrai exactement pour les états ayant une composante entière donnée)

Etant donné un recouvrement $\langle r, \pi\rangle$ de l'état de $\langle S, A, t, \varepsilon\rangle$, une preuve de fatalité de Ψ sous invariance de Φ pour $\langle S, A, t, \varepsilon\rangle$ peut être décomposée pour chaque bloc du recouvrement de l'ensemble des états :

$$(\exists \Gamma \in \mathrm{Ord},\ J \in (r \to (\Gamma \times S \times S \to \{tt, ff\}))).$$

$$\begin{aligned}
&[\forall k, \ell \in r,\ s, s_1, s_2, \bar{s} \in S,\ a \in A,\ \gamma \in \Gamma.\\
&\quad ([s \in (s_1) \wedge \pi_k(s_1)] \Rightarrow [\exists \bar{\gamma} \in \Gamma.\ J_k(\bar{\gamma}, s_1, s_2)])\\
&\quad \wedge ([J_k(\gamma, s_2, s_1) \wedge \pi_k(s_1) \wedge \gamma > 0] \Rightarrow [\Phi(s_2, s_1) \wedge \exists s_1' \in S, a \in A.\ t_a(s_1, s_1')])\\
&\quad \wedge ([J_k(\alpha, s_2, s_1) \wedge \pi_k(s_1) \wedge \gamma > 0 \wedge t_a(s_1, s_1') \wedge \pi_\ell(s_1')]\\
&\qquad\qquad \Rightarrow [\exists \gamma' < \gamma.\ J_\ell(\gamma', s_2, s_1')])\\
&\quad \wedge ([J_k(0, s_2, \bar{s}) \wedge \pi_k(\bar{s})] \Rightarrow \Psi(s_2, \bar{s}))]
\end{aligned} \qquad (\mathcal{F}'_{9.1})$$

Théorème 5.2.7.1.v1

$(\mathcal{F}'_{9.1}) \Longleftrightarrow$ (Ψ est fatale sous invariance de Φ pour $\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$)

Démonstration

- Correction, $(\mathcal{F}_{9.1}) \Rightarrow (\mathcal{F}'_c)$

  Choisir $\Gamma_c = \Gamma_9$, $J_c(\gamma, s_2, s_1) = [\exists k \in r.\ (\pi_k(s_1) \wedge J_{9k}(\gamma, s_2, s_1))]$.

- Complétude sémantique, $(\mathcal{F}'_c) \Rightarrow (\mathcal{F}_{9.1})$

  Choisir $\Gamma_9 = \Gamma_c$, $J_{9k}(\gamma, s_2, s_1) = [\pi_k(s_1) \wedge J_c(\gamma, s_2, s_1)]$.

□

On observera que de manière équivalente $(\mathcal{F}_{9.1})$ s'obtient à partir de $(\mathcal{F})$ par la décomposition 5.2.2.4.1.

Une version de la méthode de Floyd pour les programmes P_2, avec des points de coupure $\langle c, m\rangle$ où $c \in C[\![P_2]\!]$ est un point de contrôle et $m \in \mathcal{M}$ est un état mémoire, peut être dérivée à partir de $(\mathcal{F}_{9.1})$ en choisissant $r = C[\![P_2]\!]$, $\pi_k(\langle c, m\rangle) = [c = k]$, $\bar{P}(\langle c_2, m_2\rangle, \langle c_1, m_1\rangle) = \neg s(\langle c_1, m_1\rangle)$ et $\Psi(\langle c_2, m_2\rangle, \langle c_1, m_1\rangle) = [s(\langle c_1, m_1\rangle) \wedge \psi(m_2, m_1)]$. Comme on le verra avec le paragraphe 5.2.1, nous pourrons poser $P_c(m_2, m_1) = [\exists c_2 \in C[\![P_2]\!].\ J_{9c}(\langle c_2, m_2\rangle, \langle c_1, m_1\rangle)]$.

5.2.7.2 Décomposition des conditions de vérification au moyen d'un recouvrement de l'ensemble des actions du système de transition

En décomposant l'hypothèse d'induction globale J dans $(\mathcal{F}_4)$ à $(\mathcal{F}_8)$ en une disjonction d'hypothèses d'induction locales correspondant à chaque action $a \in A$ du système de transition $\langle S,A,t,\varepsilon\rangle$, une preuve de fatalité de Ψ sous invariance de Φ pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$ peut être décomposée en :

- card(A) preuves indépendantes d'invariance et de terminaison pour chaque bloc, $((\mathcal{F}_{g.t})$-a-b-c-d)
- card(A) × (card(A)-1) vérifications d'absence d'interférences entre preuves de blocs distincts, $((\mathcal{F}_{g.t})$-e)
- une preuve d'absence d'états de blocage (par exemple par l'absurde) $((\mathcal{F}_{g.t})$-f)

$$
\begin{aligned}
&(\exists \Gamma \in \mathrm{Ord},\ J \in (A \to (\Gamma \times S \times S \to \{tt, ff\}))).\\
&\quad [\forall a \in A.\\
&\quad\ (\forall \underline{s}, s, s' \in S,\ \gamma \in \Gamma.\\
&(a)\quad\ (\varepsilon(\underline{s}) \Rightarrow [\exists \underline{\gamma} \in \Gamma.\ J_a(\underline{\gamma}, \underline{s}, s)])\\
&(b)\quad \wedge\ ([J_a(\gamma, \underline{s}, s) \wedge \gamma > 0 \wedge t_a(s, s')] \Rightarrow [\exists \gamma' < \gamma.\ J_a(\gamma', \underline{s}, s')])\\
&(c)\quad \wedge\ ([J_a(\gamma, \underline{s}, s) \wedge \gamma > 0] \Rightarrow \Phi(\underline{s}, s))\\
&(d)\quad \wedge\ (J_a(0, \underline{s}, s) \Rightarrow [\Psi(\underline{s}, s) \vee \mathrm{fa}(A)]))]\\
&\quad \wedge\ [\forall a \in A,\ b \in (A \setminus a).\\
&\qquad (\forall \underline{s}, s, s' \in S,\ \gamma \in \Gamma,\ \gamma \in \Gamma.\\
&\qquad\quad [J_a(\gamma, \underline{s}, s) \wedge J_b(\gamma, \underline{s}, s') \wedge \gamma > 0 \wedge t_b(s, s')] \Rightarrow\\
&\qquad\qquad [\exists \gamma' \in \Gamma.\ J_a(\gamma', \underline{s}, s')])]\\
&\quad \wedge\ [\forall \underline{s}, s \in S.\\
&(f)\qquad ([\forall a \in A.\ \mathrm{fa}(s) \wedge \neg\Psi(\underline{s}, s)] \Rightarrow [\exists a \in A.\ \forall \gamma \in \Gamma.\ \neg J_a(\gamma, \underline{s}, s)]))]
\end{aligned}
\qquad (\mathcal{F}_{g.t})
$$

où

$\mathrm{fa} \in (A \to (S \to \{tt, ff\}))$ caractérise les états de blocage de l'action a

$\mathrm{fa}_a(s) = [\forall s' \in S.\ \neg t_a(s, s')]$

Théorème 5.2.7.2 v1

$(\mathcal{F}_{g.t}) \Leftrightarrow$ (Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle$)

Démonstration

- Correction, $(\mathcal{F}_{g.t}) \to (\mathcal{F}_8)$

  Choisir $W_8 = (A \to \delta)$, $\delta' <_8 \delta$ si et seulement si $(\exists a \in A.\ [(\delta'_a < \delta_a) \wedge (\forall b \in (A \setminus a).\ \delta'_b \leq \delta_b)])$

  et $J_8(\delta, \underline{s}, A) = [\forall a \in A.\ J_{g_a}(\delta, \underline{s}, A)]$.

- Complétude sémantique, $(\mathcal{F}_c) \Rightarrow (\mathcal{F}_{g.t})$

  Choisir $J_{g.t}(\delta, \underline{s}, A) = [(J_c(\delta, \underline{s}, A) \wedge \neg\Psi(\underline{s}, s) \wedge \delta > 0) \vee (\Psi(\underline{s}, s) \wedge \delta = 0)]$

□

Exemple

Une généralisation de la méthode de preuve de Lamport [80] à la preuve de correction totale de programmes parallèles asynchrones $[\![\mathrm{Pra}_0 \| \ldots \| \mathrm{Pra}_{m-1}]\!]$ peut être dérivé à partir de $(\mathcal{F}_{g.t})$. La relation de transition associée au programme est décomposée en card(A) blocs t_a correspondant à chaque processus Pra_a ainsi $\overline{t}_{g_a}$ est un invariant global pour le processus Pra_a. Nous pouvons aussi [illegible] d'un des principes d'induction $(\mathcal{F}_4)$-$(\mathcal{F}_8)$ en utilisant la décomposition [illegible].

5.2.7.3 Combinaison des décompositions selon les états et les actions du système de transition

Les décompositions selon l'ensemble des états et selon l'ensemble des actions peuvent être combinées et appliquées récursivement de sorte à induire des décompositions plus fines des conditions de vérification. Par exemple, nous pouvons considérer des systèmes de transitions $\langle S, A, t, \varepsilon\rangle$ et des paires $\langle \tau, \pi\rangle$ où $\tau \in (A \to (S\times S \to \{tt, ff\}))$ et $\forall a\in A.\ (\tau_a \in (\tau_a \to (S \to \{tt, ff\})))$ avec $\forall a\in S, a\in A.\ \exists i\in\tau_a.\ \pi_a^i(A)$. Le principe d'induction correspondant est

$$
\begin{aligned}
&(\exists \Gamma \in \mathcal{O}\mathcal{N},\ J.\ [\forall a\in A,\ i\in\tau_a.\ J_a^i \in (\Gamma\times S\times S\to\{tt,ff\})]\ \wedge\\
&\quad [\forall a\in A.\\
&\qquad (\forall i,i'\in\tau_a.\\
&\qquad\quad (\forall A, A', A''\in S,\ \gamma\in\Gamma.\\
&\text{(a)}\qquad\quad ([\varepsilon(A)\wedge\pi_a^i(A)]\to[\exists\bar\gamma\in\Gamma.\ J_a^i(\bar\gamma,A,A)])\\
&\text{(b)}\qquad\quad \wedge([J_a^i(\gamma,A,A)\wedge\pi_a^i(A)\wedge\gamma>0\wedge t_a(A,A')\wedge\pi_a^{i'}(A')]\\
&\qquad\qquad\qquad\to[\exists\gamma'<\gamma.\ J_a^{i'}(\gamma',A,A')])\\
&\text{(c)}\qquad\quad \wedge([J_a^i(\gamma,A,A)\wedge\pi_a^i(A)\wedge\alpha>0]\to\Phi(A,A))\\
&\text{(d)}\qquad\quad \wedge([J_a^i(0,A,A)\wedge\pi_a^i(A)]\to[\psi(A,A)\vee\beta_a(A)]))]\\
&\quad \wedge[\forall a\in A,\ i,i'\in\tau_a,\ b\in(A\setminus a),\ j\in\tau_b.\\
&\qquad (\forall A,A',A''\in S,\ \gamma,\alpha\in\Gamma.\\
&\text{(e)}\qquad\quad ([J_a^i(\gamma,A,A)\wedge\pi_a^i(A)\wedge J_b^j(\alpha,A,A)\wedge\pi_b^j(A)\wedge t_b(A,A')\wedge\pi_a^{i'}(A')]\\
&\qquad\qquad\qquad\to[\exists\gamma'<\gamma.\ J_a^{i'}(\gamma',A,A')])]\\
&\quad \wedge[\forall A,A\in S.\\
&\text{(f)}\qquad ([\forall a\in A.\ \pi_a^i(A)\wedge\neg\psi(A,A)]\to[\exists a\in A.\ t_a(\pi_a^i(A),\ldots)\wedge\forall i\in\tau_a.\ldots])]
\end{aligned}
\tag{$\mathcal{F}_{1,2}$}
$$

Théorème 5.2.7.3-1

[illegible] pour $\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$)

Démonstration

- Correction, $(\mathcal{F}_{3.b}) \Rightarrow (\mathcal{F}_{3.t})$

  Choisir $J_{3.t\,a}^{\ i}(\gamma,A,A) = [\exists i\in\tau_a.\ (\pi_a^i(A)\wedge J_{3.b\,a}^{\ i}(\gamma,A,A))]$

- Complétude sémantique, $(\mathcal{F}_{3.t}) \Rightarrow (\mathcal{F}_{3.b})$

  Choisir $J_{3.b\,a}^{\ i}(\gamma,A,A) = [\pi_a^i(A)\wedge J_{3.t\,a}(\gamma,A,A)]$

□

Exemple

Une généralisation de la méthode de preuve de Lamport [77] (et Owicki-Gries [76]) à la preuve de correction totale de programmes parallèles asynchrones $[Pra_0 \| \ldots \| Pra_{m-1}]$ peut être dérivée à partir de $(\mathcal{F}_{3.b})$. La relation de transition associée au programme est décomposée en blocs t_a correspondant à chaque processus Pra_a, τ_a est l'ensemble des points de contrôle du processus Pra_a et π_a^i est vrai pour les états dont la composante contrôle pour le processus Pra_a est égale à i. Par suite, nous pouvons associer $J_{3.a}^{\ i}$ au point i du processus Pra_a. En outre, l'exécution atomique de commandes des processus Pra_b ne peut pas modifier le contrôle dans le processus Pra_a de sorte que le seul cas à considérer dans $(\mathcal{F}_{3.b})$-(e) est ici. Nous pouvons ainsi appliquer la décomposition 4.3.2.4.4 (ou 4.3.2.4.5) à l'un des principes d'induction $(\mathcal{F}_i)$ à $(\mathcal{F}_3)$.

□

5.2.8 PRINCIPES D'INDUCTION "A LA FLOYD" POUR DEMONTRER DES PROPRIETES DE FATALITE DE SEMANTIQUES NON CLOSES

Si nous voulons démontrer des propriétés de fatalité pour des programmes ayant une sémantique close, nous pourrons utiliser n'importe quel principe d'induction $(\mathcal{F}_i)$ à $(\mathcal{F}_9)$ pour le système de transition engendré par cette sémantique.

Comme ces principes d'induction induisent une preuve d'invariance, nous retrouverons les difficultés (et les solutions) du chapitre 4 concernant les sémantiques non closes. Il s'y ajoute la difficulté que les preuves de terminaison par la méthode de l'ordre bien fondé ne sont pas applicables aux sémantiques non closes comme le montre le contre-exemple suivant

Exemple 5.2.8-1

Pour continuer l'exemple 5.1.1-1, si nous voulions démontrer que Ψ définie par $\Psi(0)=ff$, $\Psi(1)=tt$ est fatale pour la sémantique $\langle S,A,\Sigma\rangle$ définie dans l'exemple 2.1.2-2 ($S=\{0,1\}$, $A=\{a,b\}$, $\Sigma=\{0 \xrightarrow{b} 1, 0 \xrightarrow{a} 0 \xrightarrow{b} 1, 0 \xrightarrow{a} 0 \dots 0 \xrightarrow{a} 0 \xrightarrow{b} 1, \dots\}$) par le principe d'induction $(\mathcal{F}_2')$, nous aurons $f_2(0,0) \prec_2 f_2(0,0)$ contrairement à $\omega\omega(\mathrm{Rang}(f_2), \prec_2)$.

□

5.2.8.1 Principes d'induction "à la Floyd" pour une sémantique non close définie par concordance avec une sémantique close

Il est toujours possible de définir une sémantique non close $\langle S,A,\Sigma\rangle$ par concordance avec une sémantique close (engendrée par un système de transition $\langle S^\#, A^\#, t^\#, \varepsilon^\#\rangle$) à une fonction $f_0^\# \in (S^\# \to S)$ entre états près (cf. 2.7.2.8 v1) :

$$\langle S,A,\Sigma\rangle = \simeq\langle f_0^\#\rangle(\langle S^\#, A^\#, \Sigma\langle S^\#, A^\#, t^\#, \varepsilon^\#\rangle\rangle)$$

Par conséquent, pour démontrer que $\Psi \in (S\times S \to \{tt, ff\})$ est fatale pour $\langle S,A,\Sigma\rangle$, il est toujours correct et possible (d'après 5.1.2 v3) d'utiliser l'un quelconque des principes d'induction des paragraphes 5.2.2, 5.2.3, 5.2.6.2 et 5.2.7 pour $\langle S^\#, A^\#, t^\#, \varepsilon^\#\rangle$ et $\Psi^\#$ définie par $\Psi^\#(A_0,A_1) = \Psi(f_0^\#(A_0), f_0^\#(A_1))$.

Par exemple, en utilisant le principe d'induction $(\mathcal{F}_6')$, nous obtenons :

$$(\exists S^\#, A^\#,\ t^\# \in (S^\# \times S^\# \to \{tt,ff\}),\ \varepsilon^\# \in (S^\# \to \{tt,ff\}),\ f_0^\# \in (\varepsilon^\# \to S)\ :$$

$$\langle S,A,\Sigma\rangle = \simeq\langle f_0^\#\rangle(\langle S^\#, A^\#, \Sigma\langle S^\#, A^\#, t^\#, \varepsilon^\#\rangle\rangle$$

$$\wedge\ [\exists \Gamma \in \mathcal{O}_{\omega d},\ J \in (\Gamma \times S^\# \times S^\# \to \{tt,ff\}).$$

$$(\forall s_1 \in S^\#.\ \exists \gamma \in \Gamma.\ J(\gamma, s_1, s_1))$$

$$\wedge\ (\forall s_1, s_2 \in S^\#,\ \gamma' \in \Gamma.$$

$$J(\gamma', s_1, s_2) \Rightarrow$$

$$[\ (\Phi(f_0^\#(s_1), f_0^\#(s_2)) \wedge \exists s'' \in S^\#, a \in A^\#.\ t_a^\#(s_1, s'')\ \wedge$$

$$\forall \dots \in S^\#.\ \dots \Rightarrow \exists \dots)$$

$$\vee\ (\neg\Phi(f_0^\#(s_1), f_0^\#(s_2)) \wedge \Psi(\dots, \dots))]]]] \qquad (\mathcal{F}_6'^\#)$$

Quand $S^\sharp$, $A^\sharp$, $t^\sharp$ et $\varepsilon^\sharp$ sont définis en termes de S, A, t, ε (et des domaines auxiliaires), nous pouvons leur substituer leurs définitions dans les conditions de vérifications relatives au système de transition $\langle S^\sharp A^\sharp, t^\sharp, \varepsilon^\sharp\rangle$ de façon à obtenir des conditions de vérification équivalentes relatives au système de transition original $\langle S, A, t, \varepsilon\rangle$ (et des variables auxiliaires). Alors, par construction, le principe d'induction est correct et sémantiquement complet.

Exemple 5.2.8.1-1

Reprenons l'exemple 2.7.3.2-2 où la réduction aux traces faiblement équitables $\underset{\sim}{\text{Wfair}}\langle A\rangle(\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle)$ d'une sémantique $\langle S, A, \Sigma\langle S, A, t, \varepsilon\rangle\rangle$ engendrée par un système de transition $\langle S, A, t, \varepsilon\rangle$ est définie par concordance avec la sémantique engendrée par le système de transition $\langle S^\sharp, A, t^\sharp, \varepsilon^\sharp\rangle$ défini par :

$S^\sharp = (A\to\omega)\times S$

$t^\sharp(\langle m, A\rangle, \langle m', A'\rangle) = [\exists a\in A.\ t_a(A, A')\wedge([m_a>0 \wedge m'_a<m_a \wedge \forall b\in(A\vee a).(m'_b = m_b)]\vee [\forall b\in A.((\beta_b(A)\vee m_b>0)\wedge m'_b>0)])]$

$\varepsilon^\sharp(\langle m, A\rangle) = \varepsilon(A)$

à une fonction $f^\sharp_S$ entre états près telle que :

$f^\sharp_S(\langle m, A\rangle) = A$

En utilisant le principe d'induction $(\mathcal{F}'_c)$ pour $\langle S^\sharp, A, t^\sharp, \varepsilon^\sharp\rangle$ et en posant $J_c(\gamma, \langle \underline{m}, A\rangle, \langle m, A\rangle) = J_{12}(\gamma, m, \underline{A}, A)$, nous obtenons un principe d'induction pour démontrer les propriétés de fatalité de la sémantique $\underset{\sim}{\text{Wfair}}\langle A\rangle(\langle S, A\ \Sigma\langle S, A, t, \varepsilon\rangle\rangle$ quand $\underset{\sim}{\text{card}}(A)<\omega$:

$(\exists \Gamma\in 0_{\text{rd}},\ J\in(\Gamma\times(A\to\omega)\times S\times S\to\{ff, tt\}).$

$[\forall \underline{A}, A, A', \Sigma\in\Sigma,\ \gamma\in\Gamma,\ \underline{m}, m, m\in(A\to\omega),\ a\in A.$

$(\varepsilon(\underline{A})\Rightarrow[\exists \underline{\gamma}\in\Gamma.\ J(\underline{\gamma}, \underline{m}, \underline{A}, \underline{A})])$

$\wedge$

$([J(\gamma, m, \underline{A}, A)\ \wedge\ \gamma>0]$

$\Rightarrow[\Phi(\underline{A}, A)\wedge\exists A'\in S, a\in A.\ t_a(A, A')\ \wedge(m_a>0\vee B(m, A))])$ $(\mathcal{F}'_{12})$

$\wedge$

$([J(\gamma, m, \underline{A}, A)\ \wedge\ \gamma>0\ \wedge\ t_a(A, A')\wedge([m_a>0\vee\ m'\underset{\sim}{<}m]\vee[B(m, A)\wedge m'>0])]$

$\Rightarrow[\exists\gamma'<\gamma.\ J(\gamma', m', \underline{A}, A')])$

$\wedge$

$(J(0, m, \underline{A}, A)\Rightarrow\Psi(\underline{A}, A))])$

$B(m, A) = [\forall b\in A.(\beta_b(A)\vee m_b=0)]$

$m'\underset{\sim}{<}m$ si et seulement si $(m'_a<m_a\wedge\forall b\in(A\vee a).(m'_b=m_b))$

$m'>0$ si et seulement si $(\forall b\in A.\ m'_b>0)$

Pour l'exemple 5.1.1-1, $(\mathcal{F}'_{12})$ est satisfait par $J_{12}(\gamma, m, \Sigma, x) = [(\gamma=0\wedge x=1)\vee(\gamma=m_1+m_2>0)]$.

□

5.2.8.2 Principes d'induction "à la Floyd" pour une sémantique non close spécifiée par un système de transition et une condition sur les traces qu'il engendre

Nous pouvons également réutiliser l'idée du paragraphe 4.2.3.1 qui consiste à cumuler l'histoire des calculs dans une variable auxiliaire. En effet,

- quand la sémantique n'est pas fermée par fusion, la condition de vérification (c) de $(\mathcal{F}_8)$ doit être affaiblie pour que le principe d'induction soit sémantiquement complet. En effet l'invariant doit être vrai pour tous les états qui peuvent être atteints en suivant le préfixe d'une trace où Ψ n'est jamais satisfaite : mais pas forcément pour tous les préfixes obtenus par transitions successives $\bigvee_{a \in A} t_a$.

- quand la sémantique n'est pas réduite par élimination des traces préfixes stricts, la condition (b) de $(\mathcal{F}_8)$ doit être renforcée pour que le principe d'induction soit correct. Il faut s'assurer que pour les traces finies, le but Ψ est atteint avant la fin de la trace (qui peut ne pas être un état de blocage).

- quand la sémantique n'est pas fermée par limites, la condition de $(\mathcal{F}_8)$ doit être affaiblie (comme le montre le contre-exemple 5.2.8.1) pour que le principe d'induction soit sémantiquement complet. On ne peut pas reprendre l'idée d'une quantité prise dans un ensemble bien fondé à un ensemble strictement décroissant à chaque pas, pour un ensemble "statique" de points de coupure (statique étant compris comme signifiant que l'ensemble des points de coupure ne dépend choisi uniquement en fonction de l'ensemble Σ d'états).
Cependant, il est possible de choisir l'ensemble des points de coupure de manière où la quantité

strictement est choisi non plus en fonction de l'état courant du calcul, mais en fonction de l'histoire complète du calcul.

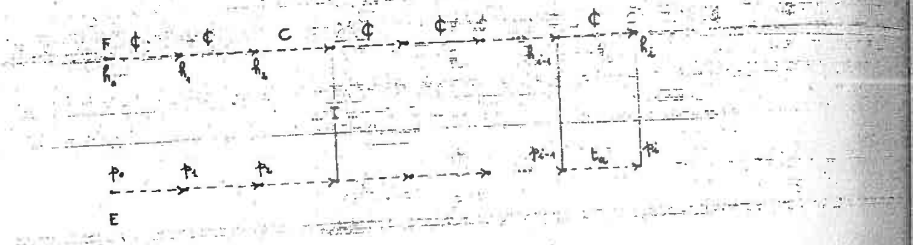
Nous commençons par le cas général, puis examinons chaque cas particulier où la sémantique n'est pas close parce que seulement une ou deux des conditions du théorème 2.6.8.4 ne sont pas satisfaites.

5.2.8.2.1 Sémantique non fermée par fusion, non réduite par élimination des traces préfixes stricts et non fermée par limites

En tenant compte des remarques précédentes, nous obtenons le principe d'induction suivant (où $\langle S,A,t,\varepsilon\rangle$ est engendré par $\langle S,A,\Sigma\rangle$) :

$$
\begin{array}{l}
(\exists H,\ F\in(S\times H\to\{tt,ff\}),\ I\in(S\times H\to\{tt,ff\}),\ L,R\in(S\times H\times A\times S\to\{tt,ff\}),\\
C,\Phi\in(H\times A\times S\times H\to\{tt,ff\}).\\
\quad \underline{\text{Live}}\langle S,A,\Sigma\rangle(A,F,I,C,\Phi)(L)\\
\quad \wedge\ \underline{\text{Next}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)(R)\\
\quad \wedge\ \underline{\text{O-cutset}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)\\
\quad \wedge\ (\exists \Gamma\in \underline{\text{Ord}},\ J\in(\Gamma\times S\times S\times H\to\{tt,ff\}).\\
\qquad (\forall \underline{s},\underline{s}'\in S,\ \gamma\in\Gamma,\ \underline{h}\in H.\\
\qquad\quad (\varepsilon(\underline{s})\Rightarrow[\exists \underline{\gamma}\in\Gamma,\ \underline{\underline{h}}\in H.(F(\underline{s},\underline{\underline{h}})\wedge J(\underline{\gamma},\underline{s},\underline{s}',\underline{\underline{h}}))])\\
\qquad\quad \wedge(J(\gamma,\underline{s},\underline{s}',\underline{h})\Rightarrow \Psi(\underline{s},\underline{s}')\\
\qquad\qquad \vee\ [\Phi(\underline{s},\underline{s}')\wedge I(\underline{s}',\underline{h})\\
\qquad\qquad\quad \wedge\ \exists a\in A,\ s''\in S.[t_a(s,s'')\wedge L(s,\underline{h},a,s'')]\\
\qquad\qquad\quad \wedge\ \forall a\in A,\ s''\in S.([t_a(s,s'')\wedge R(s,\underline{h},a,s'')]\Rightarrow\\
\qquad\qquad\qquad [(\exists \gamma'<\gamma,\ \underline{h}'\in H.(C(\underline{h},s,s',\underline{h}')\wedge J(\gamma',\underline{s},s',\underline{h}')))\\
\qquad\qquad\qquad \vee(\exists \underline{\underline{h}}'\in H.(\Phi(\underline{h},s,s',\underline{h}')\wedge J(\gamma,\underline{s},s',\underline{h}')))]]]))))
\end{array}
\qquad (\mathcal{F}_{13}')
$$

Avant de donner les définitions manquantes, remarquons que si p est le préfixe d'une trace de Σ ($\exists p' \in \Sigma.\ p \mapsto p'$) sur lequel le but n'est pas atteint ($\forall i \in |p|.\ \neg\Psi(p_0, p_i)$), alors pour tout $i \in |p|$, il existe t_i et h_i tels que $J_{13}(t_i, p_0, p_i, h_i)$ est vrai, de même que $F(p_0, h_0)$, $I(p_i, h_i)$ et $(C(h_{i-1}, t_{i-1}, p_i, h_i) \vee \Phi(h_{i-1}, t_{i-1}, p_i, h_i))$ quand $i > 0$. De la sorte F, I, C et Φ peuvent être convenablement choisis pour que h_i soit le cumul de l'histoire du calcul ayant conduit de p_0 à p_i selon le schéma suivant :



Dans $(\mathcal{F}'_{13})$ nous avons :

- $\underline{\text{Live}}\langle S, A, \Sigma\rangle(H, F, I, C, \Phi)(L) =$

$(\forall m \in (\omega \cup 0),\ p' \in \Sigma,\ p \in \Sigma^m\langle S, A\rangle,\ h \in (m \to H),\ a \in A,\ A' \in S.$

$[\ p \mapsto p' \wedge F(p_0, h_0) \wedge \forall j \in m.\ I(p_j, h_j) \wedge \forall j \in (m \cup 0).\ [\exists b \in A.\ t_b(p_{j-1}, p_j) \wedge$

$(\Phi(h_{j-1}, b, p_j, h_j) \vee C(h_{j-1}, b, p_j, h_j))] \wedge t_a(p_{m-1}, A') \wedge$

$L(p_{m-1}, h_{m-1}, a, A')] \Rightarrow [p \notin \Sigma])$

Une formule plus facile à comprendre à l'aide du schéma suivant

(de sorte que $L(A, h, a, A')$ empêche que s'il y a pas de trace finie p de Σ qui [illegible] sur traces de histoire h [illegible])

- $\underline{\text{Next}}\langle S, A, \Sigma\rangle(H, F, I, C, \Phi)(R) =$

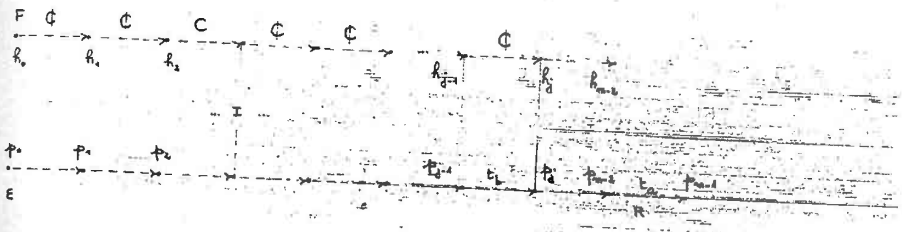
$(\forall p' \in \Sigma,\ m \in \omega,\ p \in \Sigma^m\langle S, A\rangle.\ [(p \mapsto p' \wedge m > 1) \Rightarrow$

$(\forall h \in ((m-1) \to H),\ a \in A.$

$[F(p_0, h_0) \wedge \forall j \in (m-1).\ I(p_j, h_j) \wedge \forall j \in ((m-1) \cup 0).\ [\exists b \in A.\ t_b(p_{j-1}, p_j) \wedge$

$(\Phi(h_{j-1}, b, p_j, h_j) \vee C(h_{j-1}, b, p_j, h_j))] \wedge t_a(p_{m-1}, p_{m-1})]$

$\Rightarrow R(p_{m-1}, h_{m-1}, a, p_{m-1})]))$



(de sorte que si $R(A, h, a, A')$ est vrai et p est le préfixe d'une trace de Σ se terminant dans l'état A et correspondant à l'histoire h alors $p \xrightarrow{a} A'$ est également préfixe d'une trace de Σ (et pas seulement préfixe d'une trace de la sémantique $\underline{\text{Fluo}}(\langle S, A, \Sigma\rangle)$).

- $\underline{\text{D-cutset}}\langle S, A, \Sigma\rangle(H, F, I, C, \Phi) =$

$[\forall p \in (\Sigma \cap \Sigma^\omega\langle S, A\rangle),\ h \in (\omega \to H).$

$\neg[\exists m \in (\omega \cup 0),\ \alpha \in (m \to \omega).$

$(\alpha_0 = 0 \wedge \forall i, j \in m.\ [(i < j) \Rightarrow (\alpha_i < \alpha_j)])$

$\wedge$

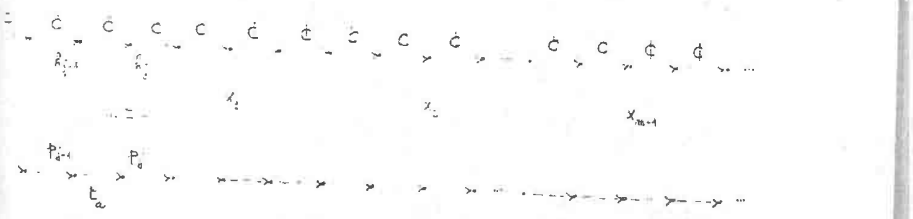
$(F(p_0, h_0) \wedge \forall j \in \omega.\ I(p_j, h_j))$

$\wedge$

$(\forall i \in (m \cup 0).\ \exists a \in A.\ [t_a(p_{\alpha_{i-1}}, p_{\alpha_i}) \wedge C(h_{\alpha_{i-1}}, a, p_{\alpha_i}, h_{\alpha_i})])$

$\wedge$

$(\forall j \in \omega.\ [(\forall i \in m.\ j \neq \alpha_i) \Rightarrow (\exists a \in A.\ (t_a(p_{j-1}, p_j) \wedge \Phi(a_{j-1}, a, p_j, h_j)))])]]$



(=intuitivement, il n'y a pas de trace infinie p et d'histoire h sur H (dont le premier élément est défini par F, les suivants par I) sauf pour un nombre fini de points de coupure qui sont définis tels pour lesquelles l'invariant I est toujours vrai. (I n'a pas été introduit dans F, C et Φ, uniquement par commodité)).

Exemple 5.2.8.2.1-1

Si $\Sigma = \Sigma\langle S,A,t,\xi\rangle$ et s'il existe $K \in \mathbb{N}$ tel que $\underline{\text{cutset}}\langle S,A,\Sigma\rangle(K)$ alors en choisissant $A=\{\perp\}$, $F(\underline{a},\underline{h}) = tt$, $I(a,h) = tt$, $C(h,a,\perp,h') = tt$, $\Phi(h,a,\perp,h',t,t') = t'$ et $J_3(\gamma,\underline{a},A) = J_{13}(\gamma,\underline{a},A,\perp)$ nous obtenons une version de $(\mathcal{F}_1^3)$.

□

Théorème 5.2.8.2.1-1 (Correction)

$(\mathcal{F}_{13}) \Rightarrow$ (Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle$)

Démonstration

Supposons $(\mathcal{F}_{13})$, $p \in \Sigma$ et $\forall i \in |h|.\neg\Psi(p_0,p_i)$. Posons $\mathrm{Inv}(d) \equiv$ $[\exists \delta \in (d \to \Gamma), h \in (d \to H).\ F(p_0,h_0) \wedge \forall j \in d.\ I(p_j,h_j) \wedge \forall j \in (d\dot{-}0).(\exists b \in A.\ t_b(p_{j-1},p_j) \wedge \Phi(h_{j-1},b,p_j,h_j)) \wedge \forall j \in (d\dot{-}0).\ C(h_{j-1},b,p_j,h_j)) \wedge \forall j \in d.\ J_{13}(\gamma_j,p_0 p_1 \ldots p_j,h_j) \wedge \forall j \in (d\dot{-}0).(\gamma_j \leq \gamma_{j-1})]$. Nous avons $\mathrm{Inv}(0)$ car $h_0 = \varepsilon$ et $F(p_0,\varepsilon)$ est vrai d'après $(\mathcal{F}_{13})$. Supposons, par induction, $\mathrm{Inv}(m-1)$ et $m \in |h|$. Nous avons $t_a(p_{m-1},p_m)$ pour une $a \in A$, donc par définition de $\underline{\text{Next}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)(R)$, $R(p_{m-1}\ h_{m-1}\ a,\ p_m\ h_m)$ est vrai et donc d'après $(\mathcal{F}_{13})$ il existe $\gamma_m < \gamma_{m-1}$, $n_m \in \mathbb{N}$ tels que $C(h_{m-1},a,p_m,h_m)$ ou bien $\gamma_m = \gamma_{m-1}$ et $h_m \neq \varepsilon$ tels que $\Phi(h_{m-1},a,p_m,h_m)$ et dans les deux cas $\exists \gamma_m \ldots$ les hypothèses de $(\mathcal{F}_{13})$ et $\forall i \in \ldots$ impliquent $I(p_m,h_m)$ et donc $\mathrm{Inv}(m)$.

Si $|p| = m \in \omega$ alors $\mathrm{Inv}(m)$ et $(\mathcal{F}_{13})$ impliquent $\exists a \in A, L \in \mathbb{S}.\ L t_a(p_{m-1},p_m) \wedge \ldots h\ a,h')]$ qui d'après la définition de $\underline{\text{Live}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)(L)$ est vrai.

Si $|p| = \omega$ alors $\mathrm{Inv}(\omega)$ et la séquence infinie d'ordinaux γ_i ne peut pas être strictement décroissante de sorte qu'il y a un rang fini d'exécuté $\gamma_i \in \omega$ tel que γ_i décroît strictement et où C est vrai. Partout ailleurs $(\mathcal{F}_{13})$ implique que Φ est vrai, en contradiction avec $\underline{\text{D-cutset}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)$.

□

Théorème 5.2.8.2.1-2 (Complétude sémantique faible)

(Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle$) $\Rightarrow (\mathcal{F}_{13})$

Démonstration

Supposons que Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle$ et définissons $H = \Sigma^{<\omega}\langle S,A\rangle$, $F(\underline{a},\underline{h}) = [\underline{h} = \langle\underline{a}\rangle]$, $I(a,h) = [\forall i \in |h|.\ \neg\Psi(h_0,h_i)]$ et $C(h,a,\perp,h') \equiv \Phi(h,a,\perp,h') \equiv [h' = h \xrightarrow{a} \perp]$. Observons que $\forall d \in ((\omega\dot{-}1)\dot{-}0)$, $\forall p' \in \Sigma$, $p \in \Sigma^d\langle S,A\rangle$. $p \to p' \Rightarrow h' \in (d \to H)$. $([F(p_0,h_0) \wedge \forall j \in d.\ I(p_j,h_j) \wedge \forall j \in (d\dot{-}0).[\exists b \in A.\ t_b(p_{j-1},p_j) \wedge (\Phi(h_{j-1},b,p_j,h_j) \vee C(h_{j-1},b,p_j,h_j))]] \Rightarrow (p = h))$. D'où $\underline{\text{D-cutset}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)$ parceque Ψ est fatale par hypothèse. De la même manière $\underline{\text{Live}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)(L)$ découle de $\underline{\text{I-cutset}}(\ldots)$ et $\underline{\text{Next}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)(R)$ de $R(a,h,a,a') \equiv [\exists p' \in \Sigma.(h \xrightarrow{a} \langle a's\rangle \to p']$. Alors $(\mathcal{F}_{13})$ est satisfait par $\Gamma = \mathbb{N}$ et $J_{13}(\delta,\underline{a},a,h) = [(\delta = 0 \wedge \Psi(\underline{a},a)) \vee (\delta = 1 \wedge \exists p' \in \Sigma.\ h \to p') \wedge \forall m \in (\omega\dot{-}0).(|h| = m \wedge h_0 = \underline{a} \wedge h_{m-1} = a \wedge [\forall i \in m.\ (\neg\Phi(h_0,h_i) \wedge \neg\Psi(h_0,h_i))])$.

□

Pour être complet, il nous faudrait examiner les 2 cas particuliers où la condition $\ldots$ n'est pas vide parceque seulement l'une ou deux des conditions du théorème $\ldots$ ne sont pas satisfaites. Pour ce faire,

- Si la sémantique est fermée par fusion, nous prenons $R(a,h,a,a') \equiv tt$ et $\underline{\text{Next}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)(R) = tt$ dans $(\mathcal{F}_{13})$.
- Si la sémantique est réduite par élimination des traces préfixes strictes, nous $\ldots$

- Si la sémantique est fermée par limites, nous prenons $\Phi(\hat{h},a,a',\hat{h}') \equiv \#$ et $\underline{\text{D-cutset}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi) = \#$ dans $(\mathfrak{F}_{13})$.

Nous ne donnerons les démonstrations que dans 2 de ces 4 cas particuliers, les 5 autres étant similaires.

5.2.8.2.2 Sémantique non close, fermée par fusion et réduite par élimination des traces préfixes stricts

$$
\begin{array}{l}
(\exists H, F\in(S\times H\to\{tt,ff\}),\ I\in(S\times H\to\{tt,ff\}),\ C\in(H\times A\times S\times H\to\{tt,ff\}),\\
\Phi\in(H\times A\times S\times H\to\{tt,ff\}).\\
\quad \underline{\text{Entry}}\langle S,A,\Sigma\rangle(E)\\
\quad \wedge\ \underline{\text{D-cutset}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)\\
\quad \wedge\ (\exists\Gamma\in\underline{\text{Ord}},\ J\in(\Gamma\times S\times S\times H\to\{tt,ff\}).\\
\qquad (\forall \underline{s},s\in S,\ \gamma\in\Gamma,\ h\in H.\\
\qquad\quad E(\underline{s})\Rightarrow[\exists\underline{\gamma}\in\Gamma,\ \underline{h}\in H.\ (F(\underline{s},\underline{h})\wedge J(\underline{\gamma},\underline{s},\underline{s},\underline{h}))]\\
\qquad\quad \wedge(J(\gamma,s,s,h)\Rightarrow\ \Psi(\underline{s},s)\\
\qquad\qquad \vee[\Phi(\underline{s},s)\wedge I(s,h)\wedge\exists a\in A,s'\in S.\ t_a(s,s')\wedge\\
\qquad\qquad\quad \forall a\in A,\ s'\in S.\ (t_a(s,s')\Rightarrow\\
\qquad\qquad\qquad [\ (\exists\gamma'<\gamma,\ h'\in H.\ C(h,a,s',h')\wedge J(\gamma',s,s',h')\\
\qquad\qquad\qquad\ \vee(\exists\underline{h}\in H.\ \Phi(\underline{h},a,s,h')\wedge\neg J(\underline{\gamma},s,s',\underline{h}))]
\end{array}
$$

Théorème 5.2.8.2.1 (Correction)

Si $\langle S,A,\Sigma\rangle = \underline{\text{Retps}}(\langle S,A,\Sigma\rangle)$ alors

Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle$

Démonstration

Supposons $(\mathfrak{F}_{14})$ et $p\in\Sigma$. S'il existe $i\in|p|$ tel que $\Psi(p_0,p_i)$ alors pour un tel plus petit i, il existe $\gamma\in(i\to\Gamma)$ et $h\in(i\to H)$ tels que $\forall j<i.\ J_{14}(\gamma_j,p_0,p_j,h_j)$ et donc $\forall j<i.\ \Phi(p_0,p_j)$. Il n'est pas possible d'avoir $\forall i\in|p|.\ \neg\Psi(p_0,p_i)$. Sinon il existerait $\gamma\in(|p|\to\Gamma)$ et $h\in(|p|\to H)$ tels que $E(p_0)$ donc $F(p_0,h_0)$ et $\forall j\in|p|.\ J_{14}(\gamma_j,p_0,p_j,h_j)$ donc $\forall j\in|p|.\ I(p_j,h_j)$ et en plus $\forall j\in(|p|\setminus 0).\ \gamma_{j-1}\geq\gamma_j$. Quand p est finie, nous avons $J_{14}(\gamma_{m-1},p_0,p_{m-1},h_{m-1})$ où $|p|=m$. Ceci implique $\exists a\in A,s'\in S.\ t_a(p_{m-1},s')$ en contradiction avec $\langle S,A,\Sigma\rangle = \underline{\text{Retps}}(\langle S,A,\Sigma\rangle)$. Quand p est infinie (auquel cas la séquence infinie d'ordinaux γ_j ne peut pas être strictement décroissante) il y a un nombre fini d'endroits $x_i\in\omega$, $i<m$, $m\in\omega$ où γ_j est strictement décroissante et c vrai. Partout ailleurs Φ serait vrai, en contradiction avec $\underline{\text{D-cutset}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)$.

□

Théorème 5.2.8.2.2

Si $\langle S,A,\Sigma\rangle = \underline{\text{Efus}}(\langle S,A,\Sigma\rangle)$ et $\langle S,A,\Sigma\rangle = \underline{\text{Retps}}(\langle S,A,\Sigma\rangle)$ alors

(Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle$) $\Rightarrow (\mathfrak{F}_{14})$

Démonstration

Supposons que Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle$ et définissons $H=\Sigma^{<\omega}\langle S,A\rangle$, $F(\underline{s},\underline{h})=[\underline{h}=\langle s\rangle]$, $I(s,h)=[\forall i<|h|.\ \neg\Psi(h_0,h_i)]$, $C(h,a,s',h')=\Phi(h,a,s',h')\wedge[h'=h\xrightarrow{a}\langle s'\rangle]$. Si nous pouvons trouver $p\in(\Sigma\wedge\Sigma^{\omega}\langle S,A\rangle)$, $h\in(\omega\to H)$, $m\in(\omega\setminus 0)$, $x\in(m\to\omega)$ ne satisfaisant pas la condition $\underline{\text{D-cutset}}\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)$ alors nous avons $F(p_0,h_0)$ et donc $h_0=p_0=p^{<1>}$. Supposons par induction que $h_j=p^{<j+1>}$. Soit $\langle x_0,\dots,x_{m-1}\rangle=x$ pour tout $i<m$ auquel nous savons $\exists a\in A.\ t_a(p_j,p_{j+1})\wedge\Phi(h_j,a,p_{j+1},h_{j+1})$ implique $h_{j+1}=p^{<j+1>}\xrightarrow{a}\langle p_{j+1}\rangle$ et $a=p_j$ d'où $h_{j+1}=p^{<j+2>}$. Si on avait $\exists i<m.\ x_i=(j+1)$ alors $x_i\neq j$ donc $i\neq 0$ et $(\exists a\in A.\ t_a(p_j,p_{j+1})\wedge C(h_j,a,p_{j+1},h_{j+1}))$ implique $h_{j+1}=p^{<j+2>}$. La contradiction avec le fait que Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle$ est maintenant que $\forall j\in\omega.\ I(p_j,h_j)$ donc $I(p_i,p^{<j+1>})$ d'où

$\neg\Psi(p_0,p_d)$. Donc D-cutset $\langle S,A,\Sigma\rangle(H,F,I,C,\Phi)$.

Choisissons maintenant $\Gamma=2$ et $J_{14}(\gamma,\underline{s},\underline{a},R) = [(\gamma=0 \wedge \Psi(\underline{s},\underline{a})) \vee$
$(\gamma=1 \wedge \exists p\in\Sigma.\ R\mapsto p) \wedge \exists m\in(\omega\setminus 0).(|R|=m) \wedge R_0=\underline{s} \wedge R_{m-1}=\underline{a} \wedge [\forall i\in m.(\overline{\Phi}(R_0,R_i)\wedge\neg\Psi(R_i,R_{i+1}))]]$
Alors, supposant $\langle S,A,\Sigma\rangle = \underline{Efus}(\langle S,A,\Sigma\rangle)$ et $\langle S,A,\Sigma\rangle = \underline{Relps}(\langle S,A,\Sigma\rangle)$, J_{14} satisfait $(\mathcal{F}'_{14})$.

□

Le résultat de complétude ci-dessus est faible dans le sens que l'ensemble de points de coupure choisi pour la preuve de complétude dépend de la propriété Ψ dont nous prouvons la fatalité.

Pour une classe donnée de sémantiques non closes, il est quelquefois possible de trouver des variables auxiliaires et un ensemble de points de coupure dynamique correspondant qui conviennent pour la preuve de toute propriété de fatalité.

Exemple 5.2.8.2.2-2

Pour $\langle S,A,\Sigma\rangle = \underline{Wfair}\langle A\rangle(\langle S,A,\Sigma\langle S,A,t,\epsilon\rangle\rangle)$ où $\underline{card}(A)<\omega$, nous pouvons choisir $H=A$, $F(A)=tt$, $I(A,R) = [\exists A'\in S.\ t_R(A,A')]$, $C(R,a,\underline{s}',R') = tt$ et $\Phi(R,a,\underline{s}',R') = [R\neq a \wedge R'=R]$.

Si il existe une trace p faiblement équitable et $R\in(|p|\to A)$ ne satisfaisant pas la condition D-cutset $\langle S,A,\Sigma\rangle(H,A,F,I,C,\Phi)$ alors ou bien $|p|=j\in(\omega\setminus 0)$ de sorte que p_{j-1} est un état de blocage, en contradiction avec $\Sigma(p_{j-1},R_{j-1})$, ou bien $|p|=\omega$ auquel cas il existe une $a=R_{x_{m-1}}$ qui est répétée $\Sigma(p_{x_{m-1}},R_{x_{m-1}})$, de sorte que $\Sigma(p_{x_i},R_{x_i})$ [illegible] $\exists b\in S.\ t_b(p_{x_i},A')$ et jamais arrivée $(\forall j>x_{m-1}.\ \exists b\in A.\ (t_b(p_{j-1},p_j)\wedge\Phi(R_{j-1},b,p_j,R_j))$ d'où $b\neq a$), en contradiction [illegible]

En remplaçant $H,F,\ldots$ par [illegible] respectives ci-dessus nous obtenons :

$(\exists\Gamma\in\underline{Ord},\ J\in(\Gamma\times S\times S\times A\to\{tt,ff\}))$.

$(\forall \underline{s},\underline{a}\in S,\ b\in\Gamma,\ b\in A.$

$(\mathcal{E}(\underline{s}) \Rightarrow [\exists\gamma\in\Gamma,\ b\in A.\ J(\gamma,\underline{s},\underline{s},b)]$ $\quad(\mathcal{F}'_{15})$

$\wedge$

$(J(\gamma,\underline{s},\underline{a},b) \Rightarrow \Psi(\underline{s},\underline{a})$

$\vee$

$[\overline{\Phi}(\underline{s},\underline{a}) \wedge \exists a'\in S.\ t_b(A,a') \wedge \forall q\in A, a\in S.(t_a(\underline{s},a') \to$

$[(\exists\gamma'<\gamma, b'\in A.\ J(\gamma',\underline{s},\underline{a}',b'))\vee(b\neq a \wedge J(\gamma,\underline{s},\underline{a}',b))]]))$

Une version de $(\mathcal{F}'_{15})$ a été proposée par Lehmann-Pnueli-Stavi [81] et leur preuve de complétude sémantique est facile à adapter. Cette preuve est indépendante de Φ et Ψ.

□

5.2.8.2.3 Sémantique (non close) fermée par limites, non fermée par fusion et non réduite par élimination des traces préfixes strictes

$$
\begin{array}{l}
(\exists H,\ F\in(S\times H\to\{tt,ff\}),\ I\in(S\times H\to\{tt,ff\}),\ L,R\in(S\times H\times A\times S\to\{tt,ff\}),\\
C\in(H\times A\times S\times H\to\{tt,ff\}).\\
\quad \underline{\mathrm{Live}}\langle S,A,\Sigma\rangle(H,F,I,C,ff)(L)\ .\\
\quad \wedge\ \underline{\mathrm{Next}}\langle S,A,\Sigma\rangle(H,F,I,C,ff)(R)\\
\quad \wedge\ (\exists\Gamma\in\underline{\mathrm{Ord}},\ J\in(\Gamma\times S\times S\times H\to\{tt,ff\}).\\
\qquad (\forall \underline{s},\underline{s}'\in S,\ \gamma\in\Gamma,\ h\in H.\\
\qquad\quad (E(\underline{s})\Rightarrow[\exists \underline{\gamma}\in\Gamma,\ \underline{h}\in H.(F(\underline{s},\underline{h})\wedge J(\underline{\gamma},\underline{s},\underline{s},\underline{h}))])\\
\qquad\quad \wedge\ (J(\gamma,\underline{s},s',h)\Rightarrow\ \Psi(\underline{s},s')\\
\qquad\qquad \vee\ [\ \Phi(\underline{s},s')\wedge I(s',h)\\
\qquad\qquad\quad \wedge\ \exists a\in A,\ s''\in S.[t_a(s',s'')\wedge L(s',h,a,s'')]\\
\qquad\qquad\quad \wedge\ \forall a\in A,\ s''\in S.([t_a(s',s'')\wedge R(s',h,a,s'')]\to\\
\qquad\qquad\qquad [\exists\gamma'<\gamma,\ h'\in H.(C(h,a,s'',h')\wedge J(\gamma',\underline{s},s'',h'))])])))
\end{array}
\qquad (\mathcal{F}_{16})
$$

Théorème 5.2.8.2.3-1 (Correction)

$(\mathcal{F}_{16})\Rightarrow(\Psi$ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle)$

Démonstration

Supposons que Ψ est fatale sous invariance de Φ pour $\langle S,A,\Sigma\rangle$. Fixons :

$\Gamma = \Sigma^{<\omega}\langle\in A\rangle$

$F(\underline{s},\underline{h}) = [\underline{h} = \langle\underline{s}\rangle]$

$I(s,h) = [\forall i\in|h|.\ \neg\Psi(h_0,h_i)]$

$C(h,a,s',h') = [h' = h\xrightarrow{a}\langle s'\rangle]$

$L(s,h,a,s') = [h\notin\Sigma]$

$R(s,h,a,s') = [\exists p'\in\Sigma.\ (h\xrightarrow{a}\langle s'\rangle)\mapsto p']$

de sorte que les conditions $\underline{\mathrm{Live}}\langle S,A,\Sigma\rangle(H,F,I,C,\phi)(L)$ et $\underline{\mathrm{Next}}\langle S,A,\Sigma\rangle(H,F,I,C,\phi)(R)$ (où $\phi(h,a,s',h') = ff$) sont satisfaites.

Définissons $h\prec h'$ si et seulement si $\exists p\in\Sigma.(h'\mapsto h\mapsto p\ \wedge\ h\neq h'\ \wedge\ \forall i\in|h|.\neg\Psi(p_0,p_i))$. Soit une suite infinie décroissante $h^0\succ h^1\succ h^2\succ\dots$. Comme la sémantique est fermée par limites, la trace limite infinie $p\in\Sigma^\omega\langle S,A\rangle$ telle que $\forall i.\ p_i=(h^{i+1})_i$ et $h^i\xrightarrow{p_i}h^{i+1}$ appartient à Σ avec $\forall i\in p.\ \neg\Psi(p_1,p_i)$ en contradiction avec l'hypothèse que Ψ est fatale. La relation $\prec$ étant bien-fondée, nous définissons :

$\rho(h) = \cap\{\alpha\in\underline{\mathrm{Ord}}:\ \forall h'\in H.[h'\prec h\Rightarrow\rho(h')<\alpha]\}$

et choisissons :

$\Gamma = \underline{\sup}^+\{\rho(h)+1 :\ h\in H\}$

$J_{16}(\gamma,\underline{s},s,h) = (\ [\gamma=0\ \wedge\ \Psi(\underline{s},s)]$

$\vee\ [\gamma>0\ \wedge\ \gamma=\rho(h)\ \wedge\ \exists p\in\Sigma.\ h\mapsto p\ \wedge\ h_0=\underline{s}\ \wedge\ \exists m\in\omega.|h|=m\ \wedge\ h_m=s\ \wedge$

$\forall i\le m.\ (\Phi(h_0,h_i)\wedge\neg\Psi(h_0,h_i))]$

Nota : J_{16} satisfait $(\mathcal{F}_{16})$ car $\forall h',h\in H.[(h'\prec h)\Rightarrow(\rho(h')<\rho(h))]$.

5.2.8.3 Equivalence forte des principes d'induction $(\mathcal{F}_6^{\#})$ et $(\mathcal{F}_{13})$

Théorème 5.2.8.3 ~1

Toute preuve de fatalité de Ψ pour une sémantique quelconque $\langle S, A, \Sigma\rangle$ utilisant le principe d'induction $(\mathcal{F}_6^{\#})$ peut se réécrire en une preuve de fatalité utilisant le principe d'induction $(\mathcal{F}_{13})$, et réciproquement.

Démonstration

- Ayant trouvé $S^{\#}, A^{\#}, t^{\#}, \varepsilon^{\#}, f^{\#}, \Gamma^{\#}$ et $J^{\#}$ satisfaisant les conditions de $(\mathcal{F}_6^{\#})$, nous pouvons toujours réécrire cette preuve de fatalité en une preuve de fatalité utilisant le principe d'induction $(\mathcal{F}_{13})$, en posant :

$H = (S^{\#} \times S^{\#})$

$F(\underline{a}, \langle \underline{a}^{\#}, a^{\#}\rangle) = [a^{\#} = \underline{a}^{\#} \wedge \underline{a} = \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}) \wedge \varepsilon^{\#}(\underline{a}^{\#})]$

$L(a, \langle \underline{a}^{\#}, a^{\#}\rangle, a, a') = [a = f_{a^{\#}}^{\#}(a^{\#}) \wedge \exists a^{\#\prime} \in S^{\#}. t_a^{\#}(a^{\#}, a^{\#\prime})]$

$R(a, \langle \underline{a}^{\#}, a^{\#}\rangle, a, a') = [a = f_{a^{\#}}^{\#}(a^{\#}) \wedge \exists a^{\#\prime} \in S^{\#}. (t_a^{\#}(a^{\#}, a^{\#\prime}) \wedge a' = \underline{f}_{a^{\#\prime}}^{\#}(a^{\#\prime}))]$

$I(a, \langle \underline{a}^{\#}, a^{\#}\rangle) = [a = f_{a^{\#}}^{\#}(a^{\#})]$

$C = t$

$\underline{t} = \underline{ff}$

$\Gamma = \Gamma^{\#}$

$J(\gamma, \underline{a}, a, \langle \underline{a}^{\#}, a^{\#}\rangle) = [\underline{a} = \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}) \wedge a = f_{a^{\#}}^{\#}(a^{\#}) \wedge \varepsilon(\underline{a}^{\#}) \wedge J^{\#}(\gamma, \underline{a}^{\#}, a^{\#})]$

En effet [illegible]

- Live $\langle \Sigma, \vdash, \Sigma\rangle(H, F, I, C, \underline{t})(L)$

car sinon on aurait $\exists m \in (\omega \setminus 0)$, $p' \in \Sigma$, $p \in \Sigma^m \langle S, A\rangle$, $h \in (m \to S^{\#} \times S^{\#})$. $[p \mapsto p' \wedge \varepsilon^{\#}(h_0(0)) \wedge h_0(0) = \underline{f}_{a_0}^{\#}(h_0(1)) \wedge p_0 = \underline{f}_{a_0}^{\#}(h_0(1)) \wedge \forall j \in m. (p_j = f_{a_j}^{\#}(h_j(1))) \wedge \forall j \in (m \setminus 0). \exists b \in A. t_b(p_{j-1}, p_j) \wedge t_a(p_{m-1}, a') \wedge p_{m-1} = f_{a_{m-1}}^{\#}(h_{m-1}(1)) \wedge \exists a^{\#\prime} \in S^{\#}. t_a^{\#}(h_{m-1}(1), a^{\#\prime}) \wedge p \in \Sigma]$, donc $h_0(1) \xrightarrow{t_0} h_1(1) \dots \xrightarrow{t_{m-2}} h_{m-1}(1)$ serait à la fois une trace de $\Sigma\langle S^{\#}, A^{\#}, t^{\#}, \varepsilon^{\#}\rangle$ car $f_{a^{\#}}(h_0(1) \xrightarrow{t_0} h_1(1) \dots \xrightarrow{t_{m-2}} h_{m-1}(1)) = p$ et préfixe strict d'une trace de $\Sigma\langle S^{\#}, A^{\#}, t^{\#}, \varepsilon^{\#}\rangle$ (car $\exists a^{\#\prime} \in S^{\#}. t_a^{\#}(h_{m-1}(1), a^{\#\prime})$) en contradiction avec le fait que $\langle S^{\#}, A^{\#}, \Sigma\langle S^{\#}, A^{\#}, t^{\#}, \varepsilon^{\#}\rangle\rangle$ est close.

- Next $\langle S, A, \Sigma\rangle(H, F, I, C, \underline{t})(R)$

car $\forall p \in \Sigma, m \in \omega, p \in \Sigma^m \langle S, A\rangle. [(p \mapsto p' \wedge m > 1) \wedge (\forall h \in ((m-1) \to S^{\#} \times S^{\#}), a \in A. (\varepsilon^{\#}(h_0(0)) \wedge h_0(0) = h_0(1) \wedge p_0 = f_{a_0}^{\#}(h_0(1)) \wedge \forall j \in (m-1). (p_j = f_{a_j}^{\#}(h_j(1))) \wedge \forall j \in ((m-1) \setminus 0). \exists b \in A. t_b(p_{j-1}, p_j) \wedge t_a(p_{m-2}, p_{m-1}))]$ implique que $h_0(1) \xrightarrow{t_0} h_1(1) \dots \xrightarrow{t_{m-2}} h_{m-1}(1)$ est préfixe d'une trace $p^{*\prime} \in \Sigma\langle S^{\#}, A^{\#}, t^{\#}, \varepsilon^{\#}\rangle$ telle que $p' = f_{a^{\#}}^{\#}(p^{*\prime})$. Comme $m-1 \leq |p| = |p^{*}|$ alors $\exists a^{\#} \in S^{\#}. t_a^{\#}(h_{m-2}(1), a^{\#}) \wedge p_{m-1} = f_{a^{\#}}^{\#}(a^{\#})$ donc $R(p_{m-2}, h_{m-2}, a, p_{m-1})$ est vrai.

- D-outset $\langle S, A, \Sigma\rangle(H, F, I, C, \underline{t})$

car sinon nous aurions une trace infinie $h_0(1) \xrightarrow{t_0} h_1(1) \dots \xrightarrow{t_{i-1}} h_i(1) \dots$ de $\Sigma\langle S^{\#}, A^{\#}, t^{\#}, \varepsilon^{\#}\rangle$ donc par $(\mathcal{F}_6^{\#})$, une chaîne infinie strictement décroissante d'ordinaux $\gamma_0, \gamma_1, \gamma_2, \dots$

$\forall \underline{a}, a \in S, \gamma \in \Gamma$, si $(\mathcal{F}_6^{\#})$,

- Si $\varepsilon(\underline{a})$ est vrai, $\exists \underline{a}^{\#} \in S^{\#}. (\underline{a} = \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}) \wedge \varepsilon^{\#}(\underline{a}^{\#}))$ donc $\exists \underline{a}^{\#} \in S^{\#}. (\underline{a} = \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}) \wedge \varepsilon^{\#}(\underline{a}^{\#}) \wedge \exists \underline{\gamma} \in \Gamma. J^{\#}(\underline{\gamma}, \underline{a}^{\#}, \underline{a}^{\#}))$ et donc $\exists \underline{a}^{\#} \in S^{\#}. (F(a, \langle \underline{a}^{\#}, a^{\#}\rangle) \wedge \exists \underline{\gamma} \in \Gamma. J(\gamma, \underline{a}, a, \langle \underline{a}^{\#}, a^{\#}\rangle) \wedge a = \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}))$. soit $\exists \underline{\gamma} \in \Gamma. \underline{h} = \langle \underline{a}^{\#}, a^{\#}\rangle \in H. (F(\underline{a}, \underline{h}) \wedge J(\underline{\gamma}, a, \underline{a}, \underline{h}))$.

- Si $J(\gamma, \underline{a}, a, \langle \underline{a}^{\#}, a^{\#}\rangle)$ alors par définition $[\underline{a} = \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}) \wedge a = f_{a^{\#}}^{\#}(a^{\#}) \wedge \varepsilon^{\#}(\underline{a}^{\#}) \wedge \varepsilon^{\#}(\underline{a}^{\#}) \wedge J^{\#}(\gamma, \underline{a}^{\#}, a^{\#})]$. Mais $J^{\#}(\gamma, \underline{a}^{\#}, a^{\#})$ implique soit $(\varepsilon^{\#}(\underline{a}^{\#}) \Rightarrow \forall \underline{b}(\underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}), \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}))$ donc $\forall \underline{b}(\underline{a}, a)$, soit $\Phi(\underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}), \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#})) \wedge \exists a^{\#\prime} \in S^{\#}. t_a^{\#}(a^{\#}, a^{\#\prime})$ et donc $[\Phi(a, a) \wedge I(a, \langle \underline{a}^{\#}, a^{\#}\rangle) \wedge \underline{t}(a, a') \wedge L(a, \langle \underline{a}^{\#}, a^{\#}\rangle, a, a')]$. Maintenant $[J(\gamma, \underline{a}, a, \langle \underline{a}^{\#}, a^{\#}\rangle, \underline{t}(a, a') \wedge R(a, \langle \underline{a}^{\#}, a^{\#}\rangle, a, a')]$ implique que $[\underline{a} = \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}) \wedge a = f_{a^{\#}}^{\#}(a^{\#}) \wedge a' = \underline{f}_{a^{\#}}^{\#}(a^{\#}) \wedge \varepsilon^{\#}(\underline{a}^{\#}) \wedge J^{\#}(\gamma, \underline{a}^{\#}, a^{\#}) \wedge \varepsilon^{\#}(\underline{a}^{\#}) \wedge J^{\#}(\gamma, \underline{a}^{\#}, a^{\#})]$ donc $[\underline{a} = \underline{f}_{a^{\#}}^{\#}(\underline{a}^{\#}) \wedge a' = \underline{f}_{a^{\#}}^{\#}(a^{\#}) \wedge \varepsilon^{\#}(\underline{a}^{\#}) \wedge \exists \gamma' < \gamma. J^{\#}(\gamma', \underline{a}^{\#}, a^{\#})]$ c'est-à-dire $\forall \underline{h}, \underline{h}' = \langle \underline{a}^{\#}, a^{\#}\rangle \in H. J(\gamma' \wedge \gamma' < \gamma)$

- Réciproquement, ayant trouvé $H, F, I, L, R, C, \Phi, \Gamma$ et J satisfaisant les conditions de $(\mathcal{F}_{13})$, nous pouvons toujours réécrire cette preuve de totalité en une preuve de fatalité utilisant le principe d'induction $(\mathcal{F}_6^\#)$ de la manière suivante :

Etant donnée $p \in \Sigma$, nous construisons $\gamma \in (|p| \to \omega)$ et $p^\# \in \Sigma^\# < (\omega \times \Gamma) \times S \times H, A >$ comme suit :

$\Gamma \times \omega$ bien-ordonné par l'ordre lexicographique gauche $<\gamma, m> \prec <\gamma', m'>$ si et seulement si $(\gamma < \gamma') \vee (\gamma = \gamma' \wedge m < m')$ est isomorphe à un ordinal $\Gamma^\# = (\omega \times \Gamma)$ par l'isomorphisme d'ordre $\underline{z}(<\gamma, m>) = (\omega \times \gamma) + m$ dont l'inverse $\underline{\gamma} \in (\Gamma^\# \to \Gamma)$, $\underline{m} \in (\Gamma^\# \to \omega)$ est tel que $\gamma = \underline{\gamma}(<\underline{\gamma}(\gamma), \underline{m}(\gamma)>)$ et $\underline{z}(<0,0>) = 0$.

Comme $\varepsilon(p_0) \Rightarrow [\exists \gamma_0 \in \Gamma, h_0 \in H. (F(p_0, h_0) \wedge J(\gamma_0, p_0, p_0, h_0))]$, nous choisissons $\alpha_0 = 0$.

Si $\neg \Psi(p_0, p_0)$ alors $I(p_0, h_0)$, ai $t_{\Phi_0}(p_0, p_0)$ alors $R(p_0, h_0, \Phi_0, p_1)$ est vrai. D'après $(\mathcal{F}_{13})$ $J(\gamma_0, p_0, p_0, h_0)$ implique, ou bien $\exists \gamma_1 < \gamma_0, h_1 \in H. (C(h_0, \Phi_0, p_0, h_1) \wedge J(\gamma_1, p_0, p_1, h_1))$ auquel cas nous choisissons $\alpha_1 = 1$, $p_0^\# = <\underline{z}(<\gamma_{\alpha_0}, 0>), p_{\alpha_0}, h_{\alpha_0}>$, $\Phi_0^\# = \Phi_0$, ou bien $\exists h_1 \in H. (\Phi(h_0, \Phi_0, p_1, h_1) \wedge J(\gamma_0, p_0, p_1, h_1))$. Alors à partir de $J(\gamma_0, p_0, p_1, h_1)$ nous appliquons $(\mathcal{F}_{13})$ jusqu'à atteindre le plus petit $0 < j \in |p|$ s'il existe tel que $[\exists \gamma_j < \gamma_0, h_j \in H. (C(h_{j-1}, \Phi_{j-1}, p_j, h_j) \wedge J(\gamma_j, p_0, p_j, h_j)) \wedge \forall k \in \omega. [0 < k < j \Rightarrow \exists h_k \in H. (I(p_k, h_k) \wedge \Phi(h_{k-1}, \Phi_{k-1}, p_k, h_k) \wedge J(\gamma_0, p_0, p_k, h_k))]]$ et nous choisissons $\alpha_1 = j \in (|p| \cup \omega)$, $\forall k \in \omega. [\alpha_0 \leq k < \alpha_1 \Rightarrow (\Phi_k^\# = \Phi_k \wedge p_k^\# = <\underline{z}(<\gamma_{\alpha_0}, (\alpha_1 - 1) - k>), p_k, h_k>)]$.

Ayant construit α_i pour $i = 0, \ldots, m-1$, $m > 0$ et $p_i^\#, \Phi_i^\#$ pour $i = 0, \ldots, (\alpha_{m-1} - 1)$ avec $\alpha_{m-1} \in (|p| \cup \omega)$ et tels que $[F(p_0, h_0) \wedge \forall i \in \alpha_{m-1}. I(p_i, h_i) \wedge \forall i \in (m \cup \omega). [(C(h_{\alpha_{i-1}}, \Phi_{\alpha_{i-1}}, p_{\alpha_i}, h_{\alpha_i}) \wedge t_{\Phi_{\alpha_{i-1}}}(p_{\alpha_{i-1}}, p_{\alpha_i})) \wedge \forall k \in \omega. [\alpha_{i-1} < k < \alpha_i \Rightarrow (\Phi(h_{\alpha_{i-1}}, \Phi_{\alpha_{i-1}}, p_k, h_k) \wedge t_{\Phi_{\alpha_{i-1}}}(p_{\alpha_{i-1}}, p_k))]] \wedge J(\gamma_{\alpha_{m-1}}, p_0, p_{\alpha_{m-1}}, h_{\alpha_{m-1}})]$. Nous avons supposé que $\Gamma(p_{\alpha_{m-1}}, h_{\alpha_{m-1}})$ ait $\neg \Psi(p_0, p_{\alpha_{m-1}})$. De plus si $t_{\Phi_{\alpha_{m-1}}}(p_{\alpha_{m-1}}, p_{\alpha_{m-1}})$ alors $\Phi_{\alpha_{m-1}}(p_{\alpha_{m-1}}, p_{\alpha_{m-1}}, h_{\alpha_{m-1}}, p_{\alpha_{m-1}+1})$. Nous avons supposé $(\mathcal{F}_{13})$ juste jusqu'à atteindre le plus petit $\alpha_{m-1} < j \in |p| \cup \omega$ s'il existe tel que $[\forall \gamma_j < \gamma_{\alpha_{m-1}}, h_j \in H. (C(h_{j-1}, \Phi_{j-1}, p_j, h_j) \wedge J(\gamma_j, p_0, p_j, h_j)) \wedge \forall k \in \omega. [\alpha_{m-1} < k < j \Rightarrow \exists h_k \in H. (I(p_k, h_k) \wedge \Phi(h_{k-1}, \Phi_{k-1}, p_k, h_k) \wedge J(\gamma_{\alpha_{m-1}}, p_0, p_k, h_k))]]$ et nous choisissons $\alpha_m = j \in (|p| \cup \omega)$, $\forall k \in \omega. [\alpha_{m-1} \leq k < \alpha_m \Rightarrow (\Phi_k^\# = \Phi_k \wedge p_k^\# = <\underline{z}(<\gamma_{\alpha_{m-1}}, (\alpha_m - 1) - k>), p_k, h_k>)]$. Nous avons ainsi α_i pour $i = 0, \ldots, m$, et $p_i^\#, \Phi_i^\#$ pour $i = 0, \ldots, (\alpha_m - 1)$ avec $\alpha_m \in (|p| \cup \omega)$.

Si un tel j n'existe pas, nous terminons la construction comme suit : il existe (c'est évident si p est finie sinon d'après Σ-cutset$<S, A, \Sigma>(H, F, I, C, \Phi)$) un plus petit $\underline{\imath} \in |p|$ tel que $\forall k \in \omega. [\alpha_{m-1} < k < \underline{\imath} \Rightarrow \exists h_k \in H. (\Gamma(p_k, h_k) \wedge \Phi(h_{k-1}, \Phi_{k-1}, p_k, h_k) \wedge J(\gamma_{\alpha_{m-1}}, p_0, p_k, h_k))] \wedge \neg I(p_{\underline{\imath}}, h_{\underline{\imath}})$, donc $\Psi(p_0, p_{\underline{\imath}})$. Nous faisons le même choix que ci-dessus en remplaçant j par $\underline{\imath}$ et $\forall k \in |p|. (k > \underline{\imath} \Rightarrow (p_k^\# = p_k \wedge p_k^\# = <\underline{z}(<0,0>), p_k, h_k>))$.

Si $\Psi(p_0, p_{\alpha_{m-1}})$ alors la construction se termine comme ci-dessus (en prenant $\underline{\imath} = \alpha_{m-1}$).

Posons maintenant :

$\Sigma^\# = \{p^\# : p \in \Sigma\}$

$S^\# = S \times 2^{\Sigma^\#} \times \omega$

$A^\# = A$

$\Gamma^\# = \omega \times \Gamma$

$\varepsilon^\#(<\underline{A}, \underline{I}, \underline{m}>) = [\underline{I} = \{p^\# \in \Sigma^\# : p_{\underline{m}}^\#(1) = \underline{A}\} \neq \phi \wedge \underline{m} = 0]$

$t_a^\#(<A, I, m>, <A', I', m'>) = [m' = m+1 \wedge I' = \{p^\# \in I : p_m^\#(1) = A \wedge \Phi_m^\# = a \wedge p_m^\#(1) = A'\} \neq \phi]$

$f_A^\#(<A, I, m>) = A$

$J^\#(\gamma^\#, <\underline{A}, \underline{I}, \underline{m}>, <A, I, m>) = [\varepsilon^\#(<\underline{A}, \underline{I}, \underline{m}>) \Rightarrow I = \{p^\# \in \Sigma^\# : m \in |p^\#| \wedge p_0^\#(1) = \underline{A} \wedge p_m^\#(1) = A \wedge p_m^\#(0) = \gamma^\#\} \neq \phi]$

alors

- Par construction de $S^\#, A^\#, \Sigma^\#$ et $f_A^\#$ nous avons $<S, A, \Sigma> = \Phi^\#(<S^\#, A^\#, \Sigma^\#>)$ et $\Sigma^\# = \Sigma < S^\#, A^\#, t^\#, \varepsilon^\#>$.

- $\forall <\underline{A}, \underline{I}, \underline{m}> \in S^\#. [\varepsilon^\#(<\underline{A}, \underline{I}, \underline{m}>) \Rightarrow (I = \{p^\# \in \Sigma^\# : p_{\underline{m}}^\#(1) = \underline{A}\} \neq \phi \wedge \underline{m} = 0)]$ donc $\forall <\underline{A}, \underline{I}, \underline{m}> \in S^\#. \exists \underline{\gamma}^\# = p_{\underline{m}}^\#(0) \in \Gamma^\#. [\varepsilon^\#(<\underline{A}, \underline{I}, \underline{m}>) \Rightarrow \underline{I} = \{p^\# \in \Sigma^\# : \underline{m} \in |p^\#| \wedge p_{\underline{m}}^\#(0) = \underline{\gamma}^\# \wedge p_{\underline{m}}^\#(1) = \underline{A}\} \neq \phi]$ c'est à dire $\forall <\underline{A}, \underline{I}, \underline{m}> \in S^\#. \exists \underline{\gamma}^\# \in \Gamma^\#. J^\#(\underline{\gamma}^\#, <\underline{A}, \underline{I}, \underline{m}>, <\underline{A}, \underline{I}, \underline{m}>)$.

- Si $J^\#(\gamma^\#, <\underline{A}, \underline{I}, \underline{m}>, <A, I, m>)$ alors $[\varepsilon^\#(<\underline{A}, \underline{I}, \underline{m}>) \Rightarrow I = \{p^\# \in \Sigma^\# : m \in |p^\#| \wedge p_0^\#(1) = \underline{A} \wedge p_m^\#(1) = A \wedge p_m^\#(0) = \gamma^\#\} \neq \phi]$. Si $\gamma^\# = 0$ alors par construction de $p^\#$, $\varepsilon^\#(<\underline{A}, \underline{I}, \underline{m}>) \Rightarrow \underline{I} = \underline{I}^\#(<\underline{A}, \underline{I}, \underline{m}>)$. Sinon $\gamma^\# \neq 0$ et $\exists \gamma' \in S$ $a \in A^\#$. $t_a(\gamma, \gamma')$ donc $\{p^\# \in I : m \in |p^\#| \wedge p_m^\#(1) = \gamma\} \neq \phi$ également $\Phi(p_0^\#(<\underline{A}, \underline{I}, \underline{m}>), f_0^\#(<A, I, m>)) \wedge \exists <A', I', m'> \in S^\#, a \in A^\#. t_a^\#(<A, I, m>, <A', I', m'>)$. Maintenant $J^\#(\gamma, <\underline{A}, \underline{I}, \underline{m}>, <A, I, m>) \wedge t_a^\#(<A, I, m>, <A', I', m'>)] \Rightarrow [\varepsilon^\#(<\underline{A}, \underline{I}, \underline{m}>) \Rightarrow$

$[\varepsilon^\#(<\underline{\lambda}, I, m>) \Rightarrow T' = \{p^\# \in \Sigma^\# : m' \in |p^\#| \wedge p^\#_0(1) = \underline{\lambda} \wedge p^\#_{m'}(1) = \lambda'\} \neq \phi]$ donc par construction des traces $p^\#$, $\exists \gamma^{\#\prime} < \gamma^\#. [\varepsilon^\#(<\underline{\lambda}, I, m>) \Rightarrow T' = \{p^\# \in \Sigma^\# : m' \in |p^\#| \wedge p^\#_0(1) = \underline{\lambda} \wedge p^\#_{m'}(1) = \lambda' \wedge \gamma^{*\prime} = p^\#_{m'}(0)\} \neq \phi]$ donc $\exists \gamma^{\#\prime} < \gamma^\#. J^\#(\gamma^{\#\prime}, <\underline{\lambda}, I, m>, <\lambda', T', m'>)$.

□

5.3 PRINCIPES D'INDUCTION "A LA BURSTALL"

Nous formalisons la méthode des assertions intermittentes de Burstall [74] initialement conçue pour montrer la correction totale de programmes séquentiels. Nous la généralisons pour démontrer les propriétés de fatalité de programmes non-déterministes et parallèles.

Dans 5.3.1, nous dérivons à partir des exemples de Burstall [74] et Manna-Waldinger [78], un principe d'induction de base qui est une formulation très concise de la méthode des assertions intermittentes de Burstall. Nous démontrons que la méthode est correcte. Utilisant l'induction transfinie (plutôt que finie) pour traiter le non-déterminisme infini, nous démontrons qu'elle est sémantiquement complète sous une condition suffisante (mais non nécessaire) sur les traces d'exécution et les propriétés de fatalité. Cette condition est vérifiée en particulier quand nous considérons la correction totale de programmes comme dans Burstall [74]. Elle est aussi vérifiée pour les propriétés de fatalité unaires qui ne dépendent que des états finaux (une restriction considérée par Pnueli [77], Apt-Delporte [83], Manna-Pnueli [83]).

Lorsqu'on considère des propriétés de fatalité unaires, les relations entre les valeurs initiales et finales des variables des programmes ne peuvent être exprimées qu'en affectant les valeurs initiales à des variables auxiliaires introduites dans les états. L'utilisation de variables auxiliaires a le désavantage que le programme doit être transformé. Plus important, le fait que l'utilisation de variables auxiliaires est en un sens très couplé : on peut relier des états intermédiaires quelconques lors d'un calcul et même mémoriser tout le calcul. Une telle liberté d'utilisation de variables auxiliaires n'est pas dans l'esprit de Burstall [74] et Manna-Waldinger [78] où les lemmes sont toujours de la forme "if sometime $\beta(x_1, \ldots, x_n) \wedge X_1 = x_1 \wedge \ldots \wedge X_n = x_n$ at l then sometime $\psi(x_1, \ldots, x_n, X_1, \ldots, X_n)$ at l'"

(où $x_1, \ldots, x_m$ sont les variables du programme et $z_1, \ldots, z_m$ leurs valeurs symboliques respectives au point ℓ du programme). Ceci s'exprime dans notre principe d'induction de base par l'utilisation de propriétés de fatalité binaires (mieux qu'en imposant des restrictions adéquates sur l'utilisation des variables auxiliaires qui dépendraient de la syntaxe des programmes). Cependant, nous faisons la conjecture que même pour les programmes déterministes, il existe des propriétés de fatalité pour lesquelles l'utilisation d'assertions binaires n'est pas sémantiquement complète.

Cette conjecture nous conduit, dans 5.3.2, à généraliser la méthode des assertions intermittentes de Burstall en utilisant l'induction transfinie (pour traiter le non-déterminisme infini) et des assertions intermittentes ternaires (permettant ainsi des lemmes d'une forme plus générale "*if* sometime $\phi(z_1, \ldots, z_m, x_1, \ldots, x_m) \wedge x_1 = z_1 \wedge \ldots \wedge x_m = z_m$ at ℓ *then* sometime $\psi(z_1, \ldots, z_m, z_1, \ldots, z_m, x_1, \ldots, x_m)$ at ℓ" où $z_1, \ldots, z_m$ (respectivement $x_1, \ldots, x_m$) dénotent les valeurs des variables du programme au point d'entrée (respectivement au point ℓ)). Nous démontrons que ce principe d'induction généralisé est correct et sémantiquement complet.

Nous dérivons, dans 5.3.3, une série de principes d'induction qui sont des généralisations successives du principe d'induction ci-dessus. Ceci élargit le champ d'application de la méthode (par exemple lorsqu'on utilise des ensembles bien-ordonnés infinis d'assertions intermittentes (auxquels on peut donner des représentations finies au moyen de variables auxiliaires de récurrence), la méthode de Burstall peut être étendue de façon à incorporer la méthode de Floyd[...]. De plus, la considération de formalisations de plus en plus abstraites devrait permettre une meilleure compréhension de la méthode de Burstall [illegible] d'évaluation symbolique [illegible] d'une manière unifiée et réduire à une induction sur les calculs [illegible]

généralisations successives introduisent plus de souplesse dans l'écriture des preuves mais pas de puissance supplémentaire puisque nous démontrons que tous les principes de preuve considérés sont corrects et sémantiquement complets donc équivalents.

Le principe d'induction "à la Floyd" comporte une induction le long des traces d'exécution tandis que le principe d'induction "à la Burstall" comporte la combinaison d'une induction le long (de parties) de traces d'exécution (en relation avec l'"évaluation symbolique" de Burstall) et une récursivité (en relation avec l'"induction sur les données" de Burstall). Ainsi le principe d'induction "à la Floyd" correspond au cas particulier du principe d'induction "à la Burstall" où la récursivité n'est pas utilisée.

L'argument de complétude consiste à démontrer que les preuves "à la Floyd" peuvent être reformulées en des preuves "à la Burstall" (i.e. l'induction sur les calculs peut être réduite à une induction sur les données). Cependant, cet argument n'est pas pleinement satisfaisant parceque le style des preuves permises est fixé. Les utilisateurs de la méthode de Burstall ont besoin d'un résultat de complétude plus fort puisqu'ils veulent savoir si les lemmes qu'ils vont utiliser dans leurs preuves peuvent toujours être choisis librement. Une réponse affirmative est donnée dans 5.3.4 (avec la condition nécessaire et suffisante que chaque lemme concerne une propriété qui est fatale pour le programme mais aussi relativement aux autres lemmes qui sont utilisés dans la preuve).

5.3.1 LE PRINCIPE D'INDUCTION DE BASE SOUS-JACENT A LA METHODE DES ASSERTIONS INTERMITTENTES DE BURSTALL

Dans ce paragraphe, nous donnons un principe d'induction de base qui est une formulation très concise de la méthode de Burstall. Dans le paragraphe suivant, nous allons supprimer un certain nombre de restrictions (dont nous croyons qu'elles engendrent des problèmes d'incomplétude) et dériver des principes d'induction plus généraux et plus abstraits qui généralisent la méthode de Burstall.

Le meilleur moyen de convaincre le lecteur que notre principe d'induction de base correspond effectivement à la méthode de Burstall serait de le dériver d'une formalisation déjà existante de la méthode. Puisqu'il n'existe pas de formalisation suffisamment générale et largement admise, le mieux que nous puissions faire est de partir des exemples originaux de Burstall [74]. Nous avons choisi une version simplifiée de la preuve de Burstall [74] du programme suivant qui calcule z^m lorsque $m \geq 0$ (nous utilisons les notations de Manna-Waldinger [78]) :

Exemple 5.3.1-1

```
Start:  P:=1;
Loop:   if N>0 then begin P:=2×P; N:=N-1;
                          goto Loop
                    end;
Finish:
```

Ce programme définit un système de transition $\langle \Sigma, A, T, \Phi \rangle$ comme suit :

Les états du programme sont de la forme $\langle c, m, p \rangle$ où c est un point de contrôle du programme et $m, p \in Z$ sont les valeurs des variables N, P du programme :

$S = \{Start, Loop, Finish\} \times Z \times Z$

$A = \{a\}$

L'exécution commence au point "Start" du programme avec une valeur initiale positive m de N et une valeur arbitraire p de P. Donc

$\Phi(\langle c, m, p \rangle) = [c = Start \wedge m \geq 0]$

Le programme est total et déterministe (tous les états, à part les états finaux, ont un seul état successeur) :

$t_a(\langle c, m, p \rangle, \langle c', m', p' \rangle) =$

$$\left[\begin{array}{l} (c = Start \wedge c' = Loop \wedge m' = m \wedge p' = 1) \\ \vee\ (c = Loop \wedge m > 0 \wedge c' = Loop \wedge m' = m-1 \wedge p' = 2 \times p) \\ \vee\ (c = Loop \wedge m \leq 0 \wedge c' = Finish \wedge m' = m \wedge p' = p) \end{array}\right]$$

Ce programme calcule $P = 2^m$ quand la valeur initiale m de N est positive. La propriété de correction totale peut être exprimée formellement par :

$\Psi(\langle c, m, p \rangle, \langle c', m', p' \rangle) = [c' = Finish \wedge p' = 2^m]$

et fatale pour $\langle S, A, \Sigma \langle S, A, T, \Phi \rangle \rangle$.

□

Le traitement des autres exemples de Burstall est similaire mais simplement beaucoup plus long.

5.3.1.1 Preuves de propriétés de fatalité des programmes

La totalité du programme 5.3.1-1 est spécifiée par la proposition :

"if sometime $(m \geq 0 \wedge N = m)$ at Start then sometime $P = 2^m$ at Finish"

La preuve de cette proposition utilise le lemme suivant :

"if sometime $(m \geq 0 \wedge N = m \wedge P = p)$ at Loop then sometime $(N = 0 \wedge P = p \times 2^m)$ at Loop"

Burstall observe que dans les énoncés ci-dessus m et p sont des variables mathématiques alors que N et P ne le sont pas puisque leur signification dépend du contexte. L'utilisation, dans un même énoncé, de variables du programme et de variables mathématiques pourrait prêter à confusion. Cette confusion peut être évitée si nous nous débarassons des variables du programme en utilisant des variables mathématiques différentes pour dénoter les valeurs des variables du programme à différents instants du calcul. Par exemple, le lemme pourrait être écrit comme suit :

" if sometime $n \geq 0$ at Loop then sometime $(m'=0 \wedge p'=p \times 2^m)$ at Loop "

qui signifie que :

" pour tout m, si $n \geq 0$ est vrai et l'exécution du programme commence à l'étiquette Loop avec la valeur m de la variable N du programme, alors l'exécution passera fatalement en Loop avec des valeurs m' et p' des variables N et P du programme telles que $(m'=0 \wedge p'=p \times 2^m)$ est vrai ".

Alors le lemme établit simplement que :

$$\theta_0(\langle c,m,p\rangle, \langle c',m',p'\rangle) = [c'=\text{Loop} \wedge m'=0 \wedge p'=p \times 2^m]$$

est fatale pour $\langle S,A,t,\varepsilon_0\rangle$, où :

$$\varepsilon_0(\langle c,m,p\rangle) = [c=\text{Loop} \wedge n \geq 0]$$

De la même manière, la proposition établit la fatalité de

$$\theta_1(\langle c,m,p\rangle, \langle c',m',p'\rangle) = [c=\text{Finish} \wedge p'=2^m]$$

pour $\langle S,A,t,\varepsilon_1\rangle$, où :

$$\varepsilon_1(\langle c,m,p\rangle) = [c=\text{Start} \wedge n \geq 0]$$

Plus généralement, pour démontrer que Ψ est fatale pour $\langle S,A,t,\phi\rangle$, la méthode de Burstall consiste à découvrir des propriétés auxiliaires $\{\phi_\ell \in (S \to \{\top,\bot\}) : \ell \in \Lambda\}$ et des conditions initiales correspondantes $\{\varepsilon_\ell \in (S \to \{\top,\bot\}) : \ell \in \Lambda\}$ (telles que $\exists \pi \in \Lambda^*. [\varepsilon_\pi = \phi \wedge \theta_\pi = \Psi]$) dont on démontre la fatalité :

$$\forall \ell \in \Lambda. \forall p \in \Sigma\langle S,A,t,\varepsilon_\ell\rangle. \exists i \in |p|. \theta_\ell(p_0,p_i)$$

on ne peut utiliser qu'un nombre fini ($\text{card}(\Lambda) < \omega$) de lemmes.

Remarque

Puisque Burstall [74] ne considère que des programmes totaux et déterministes, l'énoncé :

" if sometime $P(m,p)$ at L then sometime $Q(m,p,m',p')$ at L' "

peut être compris également comme :

$$\exists p \in \Sigma\langle S,A,t,\varepsilon\rangle. \exists i \in |p|. \theta(p_0,p_i)$$

où

$$\varepsilon(\langle c,m,p\rangle) = [c=L \wedge P(m,p)]$$

$$\theta(\langle c,m,p\rangle, \langle c',m',p'\rangle) = [c'=L' \wedge Q(m,p,m',p')]$$

Tous les résultats de ce paragraphe peuvent être aisément adaptés pour cette interprétation existentielle. Cependant nous avons choisi de développer l'interprétation universelle parce qu'elle est plus adaptée à la construction totale (et plus généralement aux propriétés de fatalité) de programmes paramétriques.

□

5.3.1.2 Un exemple de preuve

Maintenant, nous allons essayer à l'aide de l'exemple, de saisir l'essence de la méthode de preuve de Burstall :

La preuve de la proposition θ_1 est la suivante :

Supposer :

"sometime (N>0 ∧ N=m) at start"

alors par évaluation symbolique :

"sometime (N>0 ∧ N=m ∧ P=1) at Loop"

puis d'après le lemme θ_0 :

"sometime (N=0 ∧ P=2^m) at Loop"

puis par évaluation symbolique :

"sometime P=2^m at Finish"

Q.E.D.

La preuve du lemme θ_0 est par induction sur m, comme suit :

Supposer :

"sometime (N>0 ∧ N=m ∧ P=p) at Loop"

soit N≤0 et Q.E.D.

ou N>0 et alors par évaluation symbolique :

"sometime (N>0 ∧ N=m-1 ∧ P=p×2) at Loop"

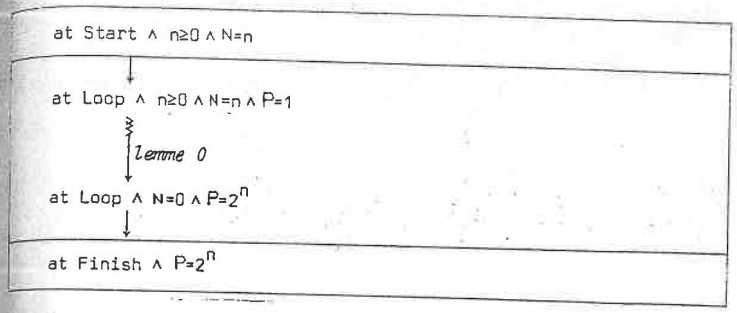
alors d'après le lemme θ_0 comme hypothèse d'induction pour m-1 (tel que m>m-1≥0) :

"sometime (N=0 ∧ P=p×2^m) at Loop"

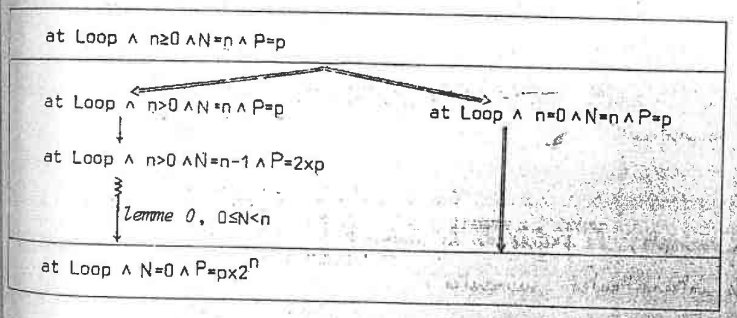
D.E.D.

Au paragraphe 5.6 nous utiliserons des chartes de preuves permettant de présenter la preuve ci-dessus comme suit :

Proposition 1 :



Lemme 0 :



5.3.1.3 Assertions intermittentes

La preuve d'un lemme est une séquence non-vide d'assertions intermittentes suivant les unes des autres par évaluation symbolique ou par application de [illegible]. Il est clair que des séquences de dérivation infinies conduiraient à des preuves invalides (puisque les exécutions infinies ne seraient pas écartées). En conséquent, une preuve d'une proposition θ_ℓ doit comporter un nombre $n_\ell+1$ d'assertions intermittentes que nous désignerons par $I^{n_\ell}_\ell, \ldots, I^1_\ell, I^0_\ell$.

Les assertions intermittentes utilisées dans la preuve n'ont pas besoin d'être distinctes comme nous le voyons dans le contre-exemple ci-dessus qui est une preuve valide de la proposition θ_1 pour tout $k \geq 1$ fini :

"sometime $(N \geq 0 \wedge N = m)$ at start" (Prémisse)

"sometime $(N \geq 0 \wedge N = m \wedge P = 1)$ at Loop" (Evaluation symbolique)

k fois {
- "sometime $(N = 0 \wedge P = 2^m)$ at Loop" (Lemme)
- ... ...
- "sometime $(N = 0 \wedge P = 2^n)$ at Loop" (Lemme)
}

"sometime $P = 2^n$ at Finish" (Conclusion)

Donc l'utilisation d'un nombre fini d'assertions intermittentes ne garantit pas que la longueur de la preuve soit finie. Par conséquent, nous devons garantir que le nombre de dérivations est fini. Pour cela, nous imposerons que toutes les occurences des assertions intermittentes utilisées dans une preuve, soient désignées par des nombres naturels et dérivées dans un ordre strictement décroissant.

Par exemple, la preuve de la proposition θ_1 comporte la découverte des assertions intermittentes suivantes :

$I_1^3(\langle c,m,p\rangle,\langle c',m',p'\rangle) = [c' = \text{start} \wedge m' \geq 0 \wedge m' = m]$

$I_1^2(\langle c,m,p\rangle,\langle c',m',p'\rangle) = [c' = \text{Loop} \wedge m' \geq 0 \wedge m' = m \wedge p' = 1]$

$I_1^1(\langle c,m,p\rangle,\langle c',m',p'\rangle) = [c' = \text{Loop} \wedge m' = 0 \wedge p' = 2^m]$

$I_1^0(\langle c,m,p\rangle,\langle c',m',p'\rangle) = [c' = \text{Finish} \wedge p' = 2^n]$

tandis que la preuve du lemme θ_0 comporte la découverte de :

$I_0^2(\langle c,m,p\rangle,\langle c',m',p'\rangle) = [c' = \text{Loop} \wedge m' \geq 0 \wedge m' = m \wedge p' = p]$

$I_0^1(\langle c,m,p\rangle,\langle c',m',p'\rangle) = [c' = \text{Loop} \wedge m' > 0 \wedge m = m'+1 \wedge p = p' \times 2]$

$I_0^0(\langle c,m,p\rangle,\langle c',m',p'\rangle) = [c' = \text{Loop} \wedge m' = 0 \wedge p = p' \times 2^{m'}]$

Remarque :

D'après Burstall [74], "'sometime P at L' says that there exists a state during the execution which is at L and has property P". Autrement dit, toutes les assertions intermittentes I_ℓ^i utilisées dans la preuve du lemme θ_ℓ devraient être fatales pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon_\ell\rangle\rangle$. Cette interprétation des assertions intermittentes est incorrecte. Par exemple, I_0^1 n'est jamais vrai durant l'exécution lorsque initialement $N=0$. Plus généralement, Burstall [74] traite les tests par cas de sorte que les assertions intermittentes utilisées dans chaque cas pourraient ne pas être fatales pour ceux des états initiaux ne correspondant pas au cas considéré. Nous choisissons une autre interprétation des assertions intermittentes de sorte que l'analyse de cas ne pose aucun problème puisque seulement la disjonction des assertions intermittentes correspondant à tous les cas, doit être fatale pour tous les états initiaux.

□

5.3.1.4 Conditions de vérification

Dans une preuve valide de fatalité de θ_ℓ pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon_\ell\rangle\rangle$, les assertions intermittentes $I_\ell^{m_\ell}, \ldots, I_\ell^1, I_\ell^0$ dérivent les unes des autres suivant des règles (pour calculer l'effet d'une affectation ou d'un test, pour utiliser un lemme, etc.). Les règles informelles de Burstall [74] doivent être comprises comme des conditions de vérification que doivent vérifier les assertions intermittentes. Nous exprimons maintenant, ces conditions de vérification formellement.

5.3.1.4.1 Prémisse

Toutes les preuves dans Burstall [74] commencent par supposer la prémisse ε_ℓ de la proposition ou du lemme θ_ℓ qu'on démontre. Autrement dit, $I^{m_\ell}_\ell$ doit être vérifiée par les états initiaux :

$\forall \Delta, \Delta' \in S.\ ([\varepsilon_\ell(\Delta) \wedge \Delta' = \Delta] \Rightarrow I^{m_\ell}_\ell(\Delta, \Delta'))$

ou plus simplement :

$\forall \Delta \in S.\ (\varepsilon_\ell(\Delta) \Rightarrow I^{m_\ell}_\ell(\Delta, \Delta))$

Par exemple, la preuve de la proposition θ_1 commence par la vérification que :

$\forall \langle c, m, p\rangle \in S.\ (\varepsilon_1(\langle c, m, p\rangle) \Rightarrow I^3_1(\langle c, m, p\rangle, \langle c, m, p\rangle))$

(où c = start, $m \geq 0$ ou la condition de vérification est vraie de manière évidente).

tandis que pour le lemme θ_0, nous avons :

$\forall \langle c, m, p\rangle \in S.\ (\varepsilon_0(\langle c, m, p\rangle) \Rightarrow I^2_0(\langle c, m, p\rangle, \langle c, m, p\rangle))$

(où c = Loop, $m \geq 0$ dans le cas non-évident)

5.3.1.4.2 Evaluation symbolique

Supposons que la preuve de la proposition θ_ℓ ait progressé jusqu'à atteindre l'assertion intermittente I^i_ℓ qui n'est pas la dernière. Le problème peut être traité par évaluation symbolique.

Pour des programmes moins déterministes, la méthode de l'évaluation pour calculer l'effet d'une affectation ou d'un test, vérifiant que l'état satisfaisant I^i_ℓ a un successeur satisfaisant une certaine assertion intermittente I^j_ℓ que doit être prise en considération plus tard dans la preuve, doit être

$\forall \Delta, \Delta' \in S.\ [I^i_\ell(\Delta, \Delta') \Rightarrow \exists \Delta'' \in S.\ t_{\underline{a}}(\Delta', \Delta'')]$

et que ces états successeurs possibles satisfont une certaine assertion intermittente I^j_ℓ qui sera considérée plus tard :

$\forall \Delta, \Delta', \Delta'' \in S.\ [(I^i_\ell(\Delta, \Delta') \wedge t_{\underline{a}}(\Delta', \Delta'')) \Rightarrow (\exists j < i.\ I^j_\ell(\Delta, \Delta''))]$

Par exemple, dans la preuve de la proposition θ_ℓ, l'affectation $P := 1$ conduit de σ_{13} à σ_{13} et correspond à la condition de vérification :

$[I^3_1(\langle c, m, p\rangle, \langle c', m', p'\rangle) \wedge t_{\underline{a}}(\langle c, m', p'\rangle, \langle c'', m'', p''\rangle)] \Rightarrow I^3_1(\langle c, m, p\rangle, \langle c'', m'', p''\rangle)$

(où c' = start, $m' > 0$, $m' = m$ ou la condition est vérifiée de manière évidente)

le test $N \leq 0$ conduit de σ_{11} à σ_{10} et correspond à la condition de vérification :

$[I^1_1(\langle c, m, p\rangle, \langle c', m', p'\rangle) \wedge t_{\underline{a}}(\langle c', m', p'\rangle, \langle c'', m'', p''\rangle)] \Rightarrow I^0_1(\langle c, m, p\rangle, \langle c'', m'', p''\rangle)$

(où c' = Loop, $m' > 0$, $p' = 2^m$, c'' = Finish, $m'' = m'$, $p'' = p'$)

Dans la preuve du lemme θ_0, le corps de la boucle mène de σ_{02} à σ_{01}. (En accord avec la sémantique opérationnelle du programme 5.3.1-1, le corps de la boucle doit être traité comme une action atomique). La condition de vérification correspondante est :

$[I^2_0(\langle c, m, p\rangle, \langle c', m', p'\rangle) \wedge m' > 0 \wedge t_{\underline{a}}(\langle c', m', p'\rangle, \langle c'', m'', p''\rangle)] \Rightarrow I^1_0(\langle c, m, p\rangle, \langle c'', m'', p''\rangle)$

(où c' = Loop, $m' = m$, $p' = p$, $m' > 0$, c'' = Loop, $m'' = m' - 1$, $p'' = 2 \times p$ ou la condition est trivialement satisfaite)

De telles conditions de vérification ne sont pas suffisantes lorsque les affectations ou les tests comportent des fonctions partielles. Dans ce cas il faut démontrer qu'aucun état de blocage n'est accessible. Plus généralement, lorsqu'il y a non-déterminisme, l'évaluation symbolique doit garantir l'existence d'au moins un état successeur :

5.3.1.4.3 Utilisation de lemmes dans la preuve de propositions

Dans la preuve de la proposition θ_1, l'assertion intermittente ς_{12} dérive de ς_{12} par le lemme θ_0. On doit d'abord vérifier que les états courants s' satisfaisant ς_{12} satisfont également la prémisse ε_0 du lemme θ_0. Alors, appliquant le lemme, on doit montrer que tous les successeurs s'' de s' par θ_0 satisfont ς_{12}. Les conditions de vérification correspondantes sont :

$$[I_4^2(\langle c,m,p\rangle,\langle c',m',p'\rangle) \rightarrow \varepsilon_0(\langle c',m',p'\rangle)]$$

$$\wedge\ [I_4^2(\langle c,m,p\rangle,\langle c',m',p'\rangle) \wedge \theta_0(\langle c',m',p'\rangle,\langle c'',m'',p''\rangle) \rightarrow I_4^2(\langle c,m,p\rangle,\langle c'',m'',p''\rangle)]$$

(où dans le cas non banal $c'=$ Loop, $m'>0$, $m''=m$, $p''=1$, $c''=$ Loop, $m''=0$, $p''=p'\times 2^{m'}$)

Plus généralement, la condition de vérification correspondant à l'utilisation d'un lemme dans la preuve d'une proposition est (temporairement) :

$$\forall s,s'\in S.\ [I_\ell^i(s,s') \rightarrow (\exists \ell'\in\Lambda.\ [\varepsilon_{\ell'}(s') \wedge \forall s''\in S.\ [\theta_{\ell'}(s',s'') \rightarrow \exists j<i.\ I_\ell^j(s,s'')]])]$$

Observer que (contrairement au cas de l'évaluation symbolique) le test que les états courants s' satisfont la prémisse $\varepsilon_{\ell'}$ du lemme $\theta_{\ell'}$ garantit l'existence d'au moins un successeur s'' à s'. Ceci parcequ'on a démontré séparément que le lemme $\theta_{\ell'}$ est fatal pour $\langle S, A, \Sigma\subset S\times A\times\Sigma, \varepsilon_{\ell'}\rangle$.

A propos de l'utilisation des lemmes, notez que Burstall [74] tient compte sur la culture mathématique de ses lecteurs et ne prend pas la peine de prouver les règles logiques élémentaires telles que les preuves par lemmes et des propositions ne doivent pas être circulaires. Cependant, de telles règles doivent être prises en compte dans la formalisation de la méthode de Burstall [74]. Une façon simple consiste à ordonner partiellement l'ensemble Λ des lemmes par un ordre bien fondé $\prec$ tel que $\ell'\prec\ell$ soit compris comme : la preuve de fatalité de $\theta_{\ell'}$ ne dépend pas de l'hypothèse que θ_ℓ est fatal. La condition de vérification (définitive) correspondant à l'utilisation d'un lemme dans la preuve d'une proposition est maintenant :

$$\forall s,s'\in S.\ [I_\ell^i(s,s') \rightarrow (\exists \ell'\in\Lambda.\ [\ell'\prec\ell \wedge \varepsilon_{\ell'}(s') \wedge \forall s''\in S.\ [\theta_{\ell'}(s',s'') \rightarrow \exists j<i.\ I_\ell^j(s,s'')]])]$$

De plus, puisque l'ensemble Λ est fini et $\prec$ est bien fondé, nous pouvons toujours (à un isomorphisme et une fonction-rang près) choisir Λ comme un ensemble d'entiers naturels et $\prec$ comme l'ordre naturel $<$ correspondant.

5.3.1.4.4 Preuve par induction sur les données

Burstall [74] démontre les lemmes en utilisant différentes formes de l'induction mathématique, qui sont toutes équivalentes à :

$$\forall m'\in\omega.\ [(\forall m<m'.\ P(m)) \rightarrow P(m')] \rightarrow [\forall m'\in\omega.\ P(m')]$$

Par exemple, dans la preuve du lemme θ_0, l'assertion intermittente ς_{00} dérive de l'assertion ς_{01} en utilisant le lemme θ_0 comme hypothèse d'induction. Ceci est valide parce que :

$$I_0^1(\langle c,m,p\rangle,\langle c',m',p'\rangle) \rightarrow [\varepsilon_0(\langle c',m',p'\rangle) \wedge m'<m].$$

Ainsi par hypothèse d'induction, nous dérivons l'assertion intermittente I_0^1 telle que :

$$[I_0^1(\langle c,m,p\rangle,\langle c',m',p'\rangle) \wedge \theta_0(\langle c',m',p'\rangle,\langle c'',m'',p''\rangle)] \Rightarrow I_0^1(\langle c,m,p\rangle,\langle c'',m'',p''\rangle)$$

(où $c'=$ loop, $m\geq 0$, $m'=m-1$, $p'=p\times 2$, $c''=$ loop, $m''=0$, $p''=p'\times 2^{m'}$)

Cette condition de vérification est propre à l'exemple considéré mais l'invariant intermittent précise que l'induction est sur les données. Puisque le concept de l'induction mathématique ci-dessus s'applique aux nombres naturels, l'induction sur les données utilise une fonction f_0 des données dans les nombres naturels.

Par exemple,

$$I_0^1(\langle c,m,p\rangle,\langle c',m',p'\rangle) \Rightarrow [\varepsilon_0(\langle c',m',p'\rangle) \wedge f_0(\langle c',m',p'\rangle) < f_0(\langle c,m,p\rangle)]$$

où :

$$f_0(\langle c,m,p\rangle) = m$$

Puisque les preuves de lemmes différents sont habituellement différentes, des fonctions f_ℓ différentes peuvent être utilisées, d'où $f_\ell \in (\Lambda \to (S \to \omega))$. Nous inférons, à partir de l'exemple, que la condition de vérification pour l'utilisation d'un lemme comme hypothèse d'induction dans la preuve de ce même lemme devrait être de la forme :

$$\forall \Lambda, \Lambda' \in S.\ [I_\ell^i(\Lambda,\Lambda') \Rightarrow [\varepsilon_\ell(\Lambda') \wedge f_\ell(\Lambda') < f_\ell(\Lambda) \wedge \forall \Lambda'' \in S.\ [\theta_\ell(\Lambda',\Lambda'') \Rightarrow \exists j < i.\ I_\ell^j(\Lambda,\Lambda'')]]]$$

5.3.1.4.5 Conclusion

Commençant par la prémisse d'un lemme, une preuve de ce lemme est finie lorsqu'on a dérivé une assertion intermittente qui implique la conclusion de ce lemme :

$$\forall \Lambda, \Lambda' \in S.\ [I_\ell^i(\Lambda,\Lambda') \Rightarrow \theta_\ell(\Lambda,\Lambda')]$$

Par exemple, la preuve de la proposition θ_1 se termine par :

$$I_1^0(\langle c,m,p\rangle,\langle c',m',p'\rangle) \Rightarrow \theta_1(\langle c,m,p\rangle,\langle c',m',p'\rangle)$$

(où $c' = \mathrm{Finish}$, $p' = z^m$ dans le cas non trivial)

On réduit que la preuve du lemme θ_0 se termine soit par :

$$[I_0^1(\langle c,m,p\rangle,\langle c',m',p'\rangle) \wedge m' < 0] \Rightarrow \theta_0(\langle c,m,p\rangle,\langle c',m',p'\rangle)$$

(où $c' = \ldots$, $m' = m$, $p = p'$)

ou par :

$$I_0^0(\langle c,m,p\rangle,\langle c',m',p'\rangle) \Rightarrow \exists j < \ldots\ [\ldots, m' < \ldots]$$

Finalement, observons que dans une preuve, toutes les assertions intermittentes intermédiaires devraient être traitées (soit par évaluation symbolique soit en utilisant un lemme (dans la preuve d'une proposition ou comme hypothèse d'induction)) ou impliquent la conclusion.

5.3.1.5 Le principe d'induction de base formalisant la méthode des assertions intermittentes

Nous pouvons maintenant résumer ce que nous avons appris à partir de l'exemple. Pour démontrer que Ψ est fatale pour $\langle S, A, \Sigma \langle S, A, t, \hat{\varphi}\rangle\rangle$, la méthode de Burstall [74] consiste à démontrer que :

$$\begin{aligned}
&[\exists \Lambda \in \omega,\ \varepsilon \in (\Lambda \to (S \to \{tt,ff\})),\ \theta \in (\Lambda \to (S^2 \to \{tt,ff\}))_\Delta,\ f \in (\Lambda \to (S \to \omega)),\\
&\ m \in (\Lambda \to \omega).\\
&\ (\exists \pi \in \Lambda.\ [\varepsilon_\pi = \hat{\Phi} \wedge \theta_\pi = \Psi])\\
&\wedge (\forall \ell \in \Lambda.\ \exists I_\ell \in (m_\ell \mapsto (S^2 \to \{tt,ff\}))_\Delta\\
&\ \forall i \le m_\ell,\ \Lambda,\Lambda' \in S.\\
(P)\quad &[\varepsilon_\ell(\Lambda) \Rightarrow I_\ell^{m_\ell}(\Lambda,\Lambda)]\\
&\wedge [I_\ell^i(\Lambda,\Lambda') \Rightarrow\\
(HS)\quad &(\exists \Lambda'' \in S, \ell \in \Lambda.\ [t_\ell(\Lambda',\Lambda'') \wedge \forall_{a \in A.}^{\ell \in \Lambda} \Lambda'' \in S.\ [t_a(\Lambda',\Lambda'') \Rightarrow \exists j < i.\ I_\ell^j(\Lambda,\Lambda'')]])\\
(LI)\quad &\vee (\exists \ell' \in \Lambda.\ [((\ell' < \ell) \vee (\ell = \ell' \wedge f_{\ell'}(\Lambda') < f_{\ell'}(\Lambda))) \wedge \varepsilon_{\ell'}(\Lambda') \wedge\\
&\qquad \forall \Lambda'' \in S.\ (\theta_{\ell'}(\Lambda',\Lambda'') \Rightarrow \exists j < i.\ I_\ell^j(\Lambda,\Lambda''))])\\
(C)\quad &\vee \theta_\ell(\Lambda,\Lambda')])]
\end{aligned} \qquad (\mathcal{B}_1)$$

5.3.1.6 Questions relatives à la correction et à la complétude sémantique de la méthode de Burstall

La question de la correction et de la complétude de la méthode de Burstall a déjà été partiellement abordée. En représentant les programmes par des relations de transition dont le non-déterminisme est fini et en donnant une interprétation temporelle de la méthode des assertions intermittentes, Pnueli [77] a montré la correction et la complétude sémantique d'une version de la méthode de Burstall. Des résultats similaires de correction et de complétude ont été obtenus par Apt-Delporte [83] pour des programmes séquentiels déterministes structurés. De tels résultats de complétude découlent informellement de la remarque de Manna-Waldinger [78] que la méthode des assertions intermittentes peut être utilisée pour exprimer les preuves classiques "à la Floyd", une méthode qui est sémantiquement complète (cf. théorème 5.2.6.2-1).

Cependant, l'exacte portée des résultats ci-dessus devrait être interprétée avec beaucoup de précautions puisque ces preuves traitent seulement le cas d'assertions intermittentes unaires (c'est-à-dire qui portent sur les états, comme "if sometime P(s) at L then sometime Q(s') at L' ") tandis que la méthode de Burstall et le principe d'induction ($\mathcal{B}_1$) utilisent des assertions intermittentes binaires (c'est-à-dire qui relient des états, comme "if sometime P(s) at L then sometime Q(s,s') at L' "). On a souvent soutenu que les deux approches sont équivalentes car l'effet d'assertions binaires peut être obtenu en utilisant des variables auxiliaires et des assertions unaires : en effet les valeurs initiales ou intermédiaires des variables d'un programme peuvent être mémorisées dans des variables auxiliaires dont les valeurs font partie de l'état. En fait l'utilisation de variables auxiliaires et d'assertions unaires est plus puissante que l'utilisation d'assertions binaires comme dans ($\mathcal{B}_1$). Ceci parce qu'en utilisant des variables auxiliaires, on peut exprimer des relations entre les valeurs des variables à deux instants différents quelconques au cours du calcul (et même mémoriser les traces d'exécution entières dans des variables d'histoire). Ceci n'est pas possible avec des assertions binaires puisque, par exemple, seulement la proposition principale (et non tous les lemmes) peut dépendre des valeurs initiales des variables du programme dans le principe d'induction ($\mathcal{B}_1$). Cependant, l'utilisation d'assertions binaires semble être beaucoup plus simple puisque la question de savoir quand on doit introduire des variables auxiliaires est résolue une fois pour toutes.

Le principe d'induction ($\mathcal{B}_1$) est correct mais nous faisons la conjecture qu'il n'est pas sémantiquement complet.

5.3.1.6.1 Correction

Théorème 5.3.1.6.1-1 (Correction)

($\mathcal{B}_1$) $\Rightarrow$ (Ψ est fatale pour $\langle S, A, \Sigma \langle S, A, t, \varepsilon \rangle\rangle$)

Démonstration

Nous introduirons plus tard ($\mathcal{B}_2$), une généralisation évidente de ($\mathcal{B}_1$) et montrerons que ($\mathcal{B}_2$) $\Rightarrow$ ($\mathcal{B}_1$)) et démontrerons que ($\mathcal{B}_2$) est correct. □

5.3.1.6.2 Conjectures à propos de la complétude sémantique

Bien que le principe d'induction ($\mathfrak{B}_1$) n'utilise que l'induction sur les entiers naturels (et contrairement à ($\mathfrak{B}_0^1$), 5.2.6.3-1), il permet de démontrer la terminaison faible de programmes qui ne se terminent pas fortement (rappelons que, d'après Dijkstra, la terminaison est forte si on peut donner une borne finie sur le nombre de pas du programme en fonction de l'état initial et qu'elle est faible sinon). Pour le montrer nous utilisons l'exemple suivant (pris littéralement dans Dijkstra[82, p.356]) :

Exemple 5.3.1.6.2-1

En général, le programme suivant (x et y étant des constantes naturelles) :

x,y := X,Y ;
do x>0 → x,y := x-1, un nombre naturel quelconque
▯ y>0 → y := y-1
od

n'a pas la propriété de terminaison forte, parceque l'exécution commence par une valeur donnée lorsque x>0.

La terminaison faible peut être démontrée au moyen de la méthode de Floyd[67] en utilisant l'ordre lexicographique gauche sur des paires de nombres naturels $\langle x,y\rangle$ (mais pas par simple induction sur les naturels).

Elle peut être démontrée également en utilisant le principe d'induction ($\mathfrak{B}_1$). Nous avons $\varepsilon = \mathbb{Z}^2$, $t_0(\langle x,y\rangle,\langle x',y'\rangle) = [(x>0 \wedge x'=x-1) \vee$ $(y>0 \wedge x'=x \wedge y'=y-1)]$, $t_1(\langle x,y\rangle,\langle x',y'\rangle) = [x=X \geq 0 \wedge y=Y \geq 0]$, $t_2(\langle x,y\rangle,\langle x',y'\rangle) = [x'=y]$, $n_0 = X+1$, $\varepsilon_{X+1} = \mathbb{N}$, $\varepsilon_2(\langle x,y\rangle) = [x=2]$ pour $i \in [1..X]$, $\theta_2 = y$ pour $i=X+1$, $\tau = X-1$, $\tau_i^2(\langle x,y\rangle) = y$ pour $i \in [1..X]$, $\tau_0^2(\langle x,y\rangle,\langle x',y'\rangle) = (\varepsilon_0(\langle x,y\rangle) \wedge t_0(\langle x,y\rangle,\langle x',y'\rangle) \wedge x=y)$ [(P), (C) quand $y'=0$, (US) quand $y'>0$], $I_0^1(\langle x,y\rangle,\langle x',y'\rangle) = (x=0 \wedge y>0 \wedge t_2(\langle x,y\rangle,\langle x',y'\rangle))$ [(P), (HS)], $I_\ell^1(\langle x,y\rangle,\langle x',y'\rangle) = (\varepsilon_\ell(\langle x,y\rangle) \wedge t_2(\langle x,y\rangle,\langle x',y'\rangle))$ [(LI) avec $\ell'=\ell-1$ quand $x'=x-1$, (LI) avec $\ell'=\ell$ quand $x'=x$ et $y'=y-1$], $I_\ell^0 = \theta_\ell$ [(C)], $n_{X-1} = 1$, $I_{X+1}^1(\langle x,y\rangle,\langle x',y'\rangle) = (\varepsilon_{X+1}(\langle x,y\rangle) \wedge \langle x',y'\rangle = \langle x,y\rangle)$ [(P), (LI) avec $\ell'=X$], $I_{X-1}^0 = \theta_{X+1}^0$ [(C)]. (Le lecteur pourra contrôler que les conditions de vérifications sont satisfaites. Nous avons indiqué après chaque assertion intermittente, l'alternative qui devrait être choisie).

□

Comme on l'a vu dans l'exemple ci-dessus, la plus grande généralité de la méthode de Burstall restreinte aux nombres naturels (qui peut être utilisée pour démontrer la terminaison faible) par rapport à la méthode de Floyd restreinte aux nombres naturels (qui ne peut être utilisée que pour démontrer la terminaison forte) est seulement spécieuse, parceque la méthode de Burstall est implicitement fondée sur l'ordre lexicographique de paires de nombres naturels comme le montre le principe d'induction ($\mathfrak{B}_2$).

Malgré cette supériorité apparente, le rang de l'ordre lexicographique sur des paires de nombres naturels, utilisé dans le principe d'induction ($\mathfrak{B}_2$) n'est pas aussi grand qu'il est nécessaire quand on considère un non-déterminisme arbitrairement infini. Aussi nous faisons la :

Conjecture 5.3.1.6.2-2 (Incomplétude sémantique)

$\mathcal{P}$ est fatale pour $\langle S, A, \Sigma\langle S,A,\iota,\varepsilon\rangle\rangle \not\Rightarrow (\mathfrak{B}_2)$

Par analogie avec la méthode de Floyd, deux solutions peuvent être envisagées pour résoudre les problèmes d'incomplétude relatifs au non-déterminisme infini. La première consiste à considérer seulement le non-déterminisme fini. L'autre consiste à considérer l'induction sur des ordres arbitraires (ou à un isomorphisme près sur des ordinaux).

Cependant, nous nous hasardons à faire la conjecture que le principe d'induction $(\mathcal{B}_3)$ n'est pas complet même avec ces hypothèses simplificatrices :

Conjecture 5.3.1.6.3 :3 (Incomplétude sémantique pour le nondéterminisme fini)

$(\Psi$ est fatale pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle \wedge \forall a\in S.\ (\mathrm{card}\,(\{a'\in S:\ t_\Sigma(a,a')\})<\omega))$

$\not\rightarrow (\mathcal{B}_2)$

Conjecture 5.3.1.6.3 :4 (Incomplétude sémantique pour des bons-ordres arbitraires)

$(\Psi$ est fatale pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle \not\rightarrow (\mathcal{B}_2)$ où $f_\varepsilon\in(A\to(S\to\Delta)),\ \Delta\in Ord)$

Ces conjectures découlent de la remarque qu'excepté pour les exemples triviaux (qui peuvent être traités par évaluation symbolique), les preuves utilisent une relation bien-fondée sur l'ensemble des descendants des états initiaux correspondant à $\langle A\times S,\prec\rangle$ où $\langle\ell',s'\rangle\prec\langle\ell,s\rangle$ si et seulement si $(\ell'\prec\ell \vee (\ell'=\ell \wedge f_\varepsilon(s')<f_\varepsilon(s)))$. Bien qu'il existe (d'après l'hypothèse de fatalité) une relation bien-fondée sur l'ensemble des descendants de chaque état initial, il peut ne pas exister de telle relation bien-fondée sur l'ensemble des descendants de tous les états initiaux comme c'est nécessaire dans le principe d'induction $(\mathcal{B}_3)$ parceque f_ε ne dépend pas des états initiaux; c'est le cas pour $S=\omega$, $t_\Sigma(x,x')=[x'=x+1]$, $\Phi(x)=\#$, $\Psi(x,x')=[x'=2x]$.

5.3.1.6.3 Un résultat de complétude sémantique partielle

Les conjectures ci-dessus n'ont que des conséquences limitées car elles ne s'appliquent pas pour un grand nombre de situations pratiques.

C'est le cas lorsque le nondéterminisme est fini et le nombre d'états initiaux est fini de sorte que (au moins en théorie) les preuves peuvent être entièrement faites par évaluation symbolique.

Des situations plus intéressantes sont celles de la correction totale des programmes séquentiels considérée par Burstall [74] ou bien les assertions intermittentes considérées par Pnueli [77] et Apt-Delporte [83].

Ces deux sortes de situations peuvent être traitées comme des cas particuliers de la situation plus générale où la fatalité de Ψ est indépendante des états initiaux pour $\langle S,A,t,\varepsilon\rangle$ (c'est-à-dire qu'aucun état intermédiaire ne peut être un but).

Définition 5.3.1.6.3 :1 (Indépendance des états initiaux)

$\mathrm{Ind}\langle S,A,t,\varepsilon,\Phi,\Psi\rangle = [(\mathrm{Inter}\langle S,A,t,\varepsilon,\Phi,\Psi\rangle \cap \mathrm{Goal}\langle S,A,t,\varepsilon,\Phi,\Psi\rangle)=\emptyset]$

Quand cette condition (suffisante mais non nécessaire) d'indépendance par rapport aux états initiaux est satisfaite, nous pouvons faire des preuves de fatalité en utilisant $(\mathcal{B}_2)$ avec $f_\varepsilon\in(A\to(S\to\Delta))$ pour un certain $\Delta\in Ord$.

Avant de démontrer ce fait, nous devons caractériser l'ordinal Δ qui est nécessaire, autrement dit proposer une "mesure" du non-déterminisme global du programme (par opposition aux caractérisations locales du nondéterminisme comme le nondéterminisme dit fini).

Nous démontrons d'abord le :

Lemme 5.3.1.6.3 ν 2 (Existence d'une relation bien-fondée pour les preuves de fatalité (avec l'hypothèse d'indépendance des états initiaux))

[(Ψ est fatale sous invariance de Φ pour ⟨S,A,Σ⟨S,A,t,ε⟩⟩) ∧ Isaind⟨S,A,t,ε,Φ,Ψ⟩]
⇒ wf(Acc⟨S,A,t,ε,Φ,Ψ⟩, t1Inter⟨S,A,t,ε,Φ,Ψ⟩^{-1})

Démonstration

Supposons par l'absurde que ∃p∈(ω→Acc⟨S,A,t,ε,Φ,Ψ⟩).∀i∈ω. t1Inter⟨S,A,t,ε,Φ,Ψ⟩(p_i, p_{i+1}). Nous pouvons supposer ε(p_0) (autrement nous pouvons adjoindre à gauche de p un préfixe r_0,...,r_k d'une trace de Σ⟨Acc⟨S,A,t,ε,Φ,Ψ⟩, A, t1Inter⟨S,A,t,ε,Φ,Ψ⟩, ε⟩ telle que ε(r_0) est vrai). Comme Ψ est fatale sous invariance pour p, il existe un plus petit i, i∈|p| tel que Ψ(p_0,p_i) est vrai. Alors p_i ∈ Goal⟨S,A,t,ε,Φ,Ψ⟩. Aussi t1Inter⟨S,A,t,ε,Φ,Ψ⟩(p_i,p_{i+1}) implique p_i ∈ Inter⟨S,A,t,ε,Φ,Ψ⟩ en contradiction avec Isaind⟨S,A,t,ε,Φ,Ψ⟩.
□

Le nondéterminisme global de ⟨S,A,t,ε⟩ relativement à Φ et Ψ peut être mesuré par le rang de l'inverse de t restreinte aux états intermédiaires :

Définition 5.3.1.6.3 : 3 (Rang du nondéterminisme global (avec l'hypothèse d'indépendance des états initiaux))

Quand Ψ est fatale sous invariance de Φ pour ⟨S,A,Σ⟨S,A,t,ε⟩⟩ et Isaind⟨S,A,t,ε,Φ,Ψ⟩ est vrai, nous définissons :

rkgnd⟨S,A,t,ε,Φ,Ψ⟩ = rk(Acc⟨S,A,t,ε,Φ,Ψ⟩, t1Inter⟨S,A,t,ε,Φ,Ψ⟩^{-1})

On observe que si le nondéterminisme est localement fini alors rkgnd⟨S,A,t,ε,Φ,Ψ⟩ < ω. De la même manière, si le nondéterminisme est dénombrable alors rkgnd⟨S,A,t,ε,Φ,Ψ⟩ < ω_1. Finalement, si t est récursive on a rkgnd⟨S,A,t,ε,Φ,Ψ⟩ < ω_1^{ck} (où ω_1^{ck}

est le premier ordinal non-récursif de Church-Kleene (Apt-Plotkin [82])).

Nous pouvons maintenant établir le résultat de complétude partielle concernant la méthode de Burstall :

Théorème 5.3.1.6.3 ν 4 (Complétude sémantique partielle)

[(Ψ est fatale pour ⟨S,A,Σ⟨S,A,t,Φ⟩⟩) ∧ Isaind⟨S,A,t,Φ,Π,Ψ⟩]
⇒ [(B_1) avec f_0 ∈ (Λ→(S→rkgnd⟨S,A,t,Φ,Π,Ψ⟩))]

Démonstration

Supposons que Ψ est fatale pour ⟨S,A,Σ⟨S,A,t,Φ⟩⟩ et Isaind⟨S,A,t,Φ,Π,Ψ⟩. Choisissons Λ:=ω, ε_0(a) = [a∈Acc⟨S,A,t,Φ,Π,Ψ⟩], ¬θ_0(a,a') = [∃p∈Σ⟨S,A,t,Φ⟩, i∈|p|. (∀j<i. ¬Ψ(p_0,p_j)) ∧ Ψ(p_0,p_i) ∧ (∃k<i. p_k=a) ∧ p_i=a'], f_0 ∈ (Acc⟨S,A,t,Φ,Π,Ψ⟩ → rkgnd⟨S,A,t,Φ,Π,Ψ⟩), f_0(a) = rk(Acc⟨S,A,t,Φ,Π,Ψ⟩, t1Inter⟨S,A,t,Φ,Π,Ψ⟩^{-1}), π_0 := ε, I_0^0(a,a') = [ε_0(a) ∧ a'=a], I_0^1(a,a') = [ε_0(a) ∧ ¬θ_0(a,a) ∧ ∃a∈A. t_a(a,a')], I_0^0 = θ_0, ε_1 = Φ, θ_1 = Ψ, π_1 = 1, I_1^1(a,a') = [ε_1(a) ∧ a'=a], I_1^0 = θ_1, π=1. Toutes les conditions de vérification sont trivialement satisfaites sauf pour ∀a,a'∈S.((I_0^1(a,a') ∧ ¬θ_0(a,a')) ⇒ (f_0(a') < f_0(a) ∧ ε_0(a')) ∧ ∀a''∈S. θ_0(a',a'') ⇒ I_0^1(a,a'')))

Si I_0^1(a,a') ∧ ¬θ_0(a,a') est vrai, nous avons par définition de I_0^1, ε_0 et la fatalité de Ψ que ∃p∈Σ⟨S,A,t,Φ⟩, i∈|p|.[(∀j<i. ¬Ψ(p_0,p_j)) ∧ Ψ(p_0,p_i) ∧ ∃k.(a=p_k ∧ (k+1)<i ∧ a'=p_{k+1})]. Puisque a,a'∈Inter⟨S,A,t,Φ,Π,Ψ⟩ et t_{p_k}(a,a'), nous avons f_0(a') < f_0(a) ∧ ε_0(a'). Si θ_0(a',a'') alors ∃q∈Σ⟨S,A,t,Φ,Π,Ψ⟩, i'∈|q|.[(∀j<i'. ¬Ψ(p_0,p_j)) ∧ Ψ(p_0,p_i) ∧ ∃k'<i'.(q_{k'}=a') ∧ q_{i'}=a'']. Nous avons ∀j.((k'<j<i') ⇒ ¬Ψ(p_0,q_j)) car sinon pour le plus petit j satisfaisant (k'<j<i') ∧ Ψ(p_0,q_j) nous aurions Inter⟨S,A,t,Φ,Π,Ψ⟩ ∧ Goal⟨S,A,t,Φ,Π,Ψ⟩. Observer que q_{i'} ∈ Goal⟨S,A,t,Φ,Π,Ψ⟩ de sorte que Ψ(p_0,q_{i'}) ne soit pas vrai car sinon q_{i'} serait un état intermédiaire de la trace p_0,...,p_k,q_{k'+1},...,q_{i'}. Puisque a=p_k et a''=q_{i'}, nous concluons que θ_0(a,a'') et donc I_0^1(a,a'') est vrai.
□

Le résultat de complétude sémantique partielle s'applique aux propriétés de fatalité telles que les états "buts" n'ont pas d'états successeurs :

Théorème 5.3.1.6.3∨5

$$[(\Psi \text{ est fatale pour } \langle S,A,\Sigma\langle S,A,t,\Phi\rangle,\Phi\rangle) \wedge \forall a\in S.(\underline{Goal}\langle S,A,t,\Phi,\Psi,\Psi\rangle(a) \Rightarrow \forall a'\in S, a\in A.\neg t_a(a,a'))] \rightarrow \underline{Iaind}\langle S,A,t,\Phi,\Psi,\Psi\rangle$$

Démonstration

Supposons $s\in \underline{Goal}\langle S,A,t,\Phi,\Psi,\Psi\rangle$. Nous avons $\forall a'\in S, a\in A.\neg t_a(a,a')$. Il s'ensuit que $A \notin \underline{Inter}\langle S,A,t,\Phi,\Psi,\Psi\rangle$ car sinon il existerait $n\in(\omega\vee o)$, $p\in\Sigma^n\langle S,A,t,\Phi\rangle$ tels que vient $\neg\Psi(p_0,p_2)$, en contradiction avec l'hypothèse de fatalité de Ψ pour $\langle S,A,\Sigma\langle S,A,t,\Phi\rangle,\Phi\rangle$.

□

Comme corollaire, nous obtenons que la méthode de preuve de correction totale de Burstall [74] pour les programmes séquentiels (i.e. (Φ_1) avec $f\in(\Lambda\to(S\to\omega))$) est sémantiquement complète parceque les états de partie n'ont pas de successeurs et que les programmes considérés sont déterministes.

Le théorème 5.3.1.6.3∨4 s'applique également à Pnueli [77] et Apt-Pnueli [83] parceque ils considèrent seulement des assertions intermittentes (i.e. les assertions intermittentes relationnelles sont exprimées en utilisant des variables auxiliaires dont les valeurs font partie de l'état).

Théorème 5.3.1.6.3∨6

$$\forall A,A'\in S.[\Psi(A,A') \Rightarrow (\forall A''\in S.\Psi(A'',A'))] \Longrightarrow \underline{Iaind}\langle S,A,t,\Phi,\Psi,\Psi\rangle$$

Démonstration

Si $A\in(\underline{Inter}\langle S,A,t,\Phi,\Psi,\Psi\rangle \cap \underline{Goal}\langle S,A,t,\Phi,\Psi,\Psi\rangle)$, alors il existe $A', A''\in S$ tels que $\neg\Psi(A',A)$ et $\Psi(A'',A)$, une contradiction.

□

5.3.2 LE PRINCIPE D'INDUCTION DE BASE GENERALISANT LA METHODE DES ASSERTIONS INTERMITTENTES DE BURSTALL

Bien que le principe d'induction $(\mathcal{B}_1)$ soit correct et sémantiquement complet dans un grand nombre de situations pratiques, nous faisons la conjecture qu'il n'est pas systématiquement général pour traiter certains types de propriétés de fatalité des programmes, comme celles considérées dans Manna-Waldinger[78] pour des programmes cycliques. D'où la nécessité de généraliser le principe d'induction $(\mathcal{B}_1)$.

La généralisation proposée est tout à fait simple. Pour garantir l'existence des bons-ordres à utiliser dans l'induction, les lemmes et les assertions intermittentes doivent dépendre des états initiaux. Pour traiter le nondéterminisme infini des bons-ordres transfinis doivent être utilisés. Ces remarques nous conduisent de $(\mathcal{B}_1)$ à $(\mathcal{B}_2)$ et nous démontrerons plus tard que $(\mathcal{B}_2)$ est sémantiquement complet.

$$
\begin{aligned}
&[\exists \Lambda\in\omega,\ \varepsilon\in(\Lambda\to(S^2\to\{tt,ff\})),\ \theta\in(\Lambda\to(S^3\to\{tt,ff\})),\ \Delta\in O_{wf},\ f\in(\Lambda\to(S^2\to\Delta)),\\
&\quad n\in(\Lambda\to\omega).\\
&\quad (\exists\pi\in\Lambda.\ \forall a,a,a'\in S.\ (\varepsilon_\pi(a,a)=[a=a\wedge\Phi(a)]\wedge\theta_\pi(a,a,a')=[a=a\wedge\Psi(a,a')]))\\
&\quad \wedge(\forall\ell\in\Lambda.\ \exists I_\ell\in(n_\ell+1\to(S^3\to\{tt,ff\})).\\
&\qquad \forall i\le n_\ell.\ a,a,a'\in S.\\
(\Xi)\qquad &\qquad [\varepsilon_\ell(a,a)\Rightarrow I_\ell^{n_\ell}(a,a,a)]\\
&\qquad \wedge[I_\ell^i(a,a,a')\Rightarrow\\
(\mathcal{B}_2)\qquad &\qquad\quad (\exists a\in\Delta.\ \exists\ell'<\ell.\ t_{\ell'}(a,a')\wedge\forall a''\in S.\ [t_{\ell'}(a,a'')\Rightarrow\exists j<i.\ I_{\ell'}^j(a,a,a'')]\\
(\mathcal{L}I)\qquad &\qquad\quad \vee(\exists a\in\Delta.\ [a=a\vee\varepsilon_\ell(a,a)]\wedge\varepsilon_\ell(a,a')\wedge\varepsilon_\ell(a,a)\wedge\varepsilon_\ell(a,a)\wedge\\
&\qquad\qquad\qquad \forall a''\in S.\ (\theta_\ell(a,a,a'')\Rightarrow\exists j<i.\ I_\ell^j(a,a,a'')]]\\
(C)\qquad &\qquad\quad \theta_\ell(a,a,a')]]]
\end{aligned}
$$

Pour illustrer l'utilisation de ce principe d'induction, considérons l'exemple suivant :

Exemple 5.3.2-1

$\Psi(x,x')=[x'=2x]$ est fatale pour $\langle\omega,\{\Omega\},\Sigma<\omega,\{\Omega\},t,\Phi\rangle$ telle que $t_\Omega(x,x')=[x'=x+1]$ et $\Phi(x)=tt$.

Observer que nous n'avons pas $\omega f(Acc\langle S,A,t,\Phi,tt,\Psi\rangle, t1Inter\langle S,A,t,\Phi,tt,\Psi\rangle^{-1})$.

La fatalité de Ψ peut être démontrée par le principe d'induction $(\mathcal{B}_2)$ en choisissant $\Lambda=2$, $\pi=1$, $\varepsilon_0(x,x)=[x\le x\le 2x]$, $\theta_0(x,x,x')=[x\le x<x+1=x'\le 2x]$, $\Phi_0(x,x)=[2x=x]$, $\varepsilon_1(x,x)=[x,x]$, $\theta_1(x,x,x')=[x=x\wedge x'=2x]$, $n_0=2$, $I_0^0(x,x,x')=[x\le x=x'\le 2x]$ (satisfaisant (P) et (C) quand $x'=2x$ ou (HS) quand $x'<2x$), $I_0^1(x,x,x')=[x\le x<x+1=x'\le 2x]$ (satisfaisant (LI) avec $\ell'=0$), $I_0^2=\theta_0$ (C), $n_1=1$, $I_1^0(x,x,x')=[x=x=x']$ ((P), (LI) avec $\ell'=0$), $I_1^1=\theta_1$ (C).

□

Le principe d'induction $(\mathcal{B}_2)$ est une généralisation évidente de $(\mathcal{B}_1)$:

Théorème 5.3.2-1 (Généralisation de la méthode de Burstall)

$(\mathcal{B}_1)\Longrightarrow(\mathcal{B}_2)$

Avant d'aborder la question de la complétude sémantique, nous remarquons que les ordinaux $\Delta\in O_{wf}$ sont suffisants dans une preuve par $(\mathcal{B}_2)$:

Théorème 5.3.2-2 (Rang du nondéterminisme global (cas général))

Lorsque Ψ est fatale sous l'observance de Φ pour $\langle S,A,\Sigma<S,A,=,\varepsilon\rangle$, nous définissons :

$rk_{gand}\langle S,A,t,\varepsilon,\Phi,\Psi\rangle = \sup^*\{rk(Acc\langle S,A,t,\varepsilon,\Phi,\Psi\rangle(a),t1Inter\langle S,A,t,\Phi,tt,\Psi\rangle^{-1}) \ldots$

(Cette définition se justifie par le fait que pour tout $\underline{z} \in S$, $\mu I_{\mathrm{inter}}\langle S,A,t,\varepsilon,\Phi,\Psi\rangle(\underline{z})$ est bien-fondée sur $A_{cc}\langle S,A,t,\varepsilon,\Phi,\Psi\rangle$, (cf. démonstration du théorème 5.2.6.2-1)).

La preuve de la complétude sémantique de $(\mathcal{P}_2)$ vient de la remarque que $(\mathcal{P}_2)$ peut être utilisé pour exprimer les preuves "à la Floyd".

Théorème 5.3.2.1.2 (Complétude sémantique)

(Ψ est fatale pour $\langle S,\underline{A},\Sigma\langle S,A,t,\Phi\rangle\rangle \Rightarrow \exists (\mathcal{P}_2)$ avec $\Delta = \mathrm{hgmd}\langle S,A,t,\Phi,\Psi\rangle$)

Démonstration

Choisir $\Lambda = 2$, $\pi = 1$, $\varepsilon_1(\underline{z},\underline{A}) = [\underline{z} = \underline{A} \wedge \Phi(\underline{z})]$, $\theta_1(\underline{z},\underline{A},\underline{A}') = [\underline{z} = \underline{A} \wedge \Psi(\underline{A},\underline{A}')]$, $\varepsilon_0(\underline{z},\underline{A}) = [\underline{A} \in A_{cc}\langle S,A,t,\Phi,\Psi\rangle(\underline{z})]$, $\theta_0(\underline{z},\underline{A},\underline{A}') = [\exists p \in \Sigma\langle S,A,t,\Phi\rangle, \lambda \in |p|.\ (\forall j \in \lambda.\ \neg\Psi(p_0,p_j)) \wedge \Psi(p_0,p_\lambda) \wedge \underline{z} = p_0 \wedge \exists k \leq \lambda.\ p_k = \underline{A} \wedge p_\lambda = \underline{A}')]$, f_1 est inutile, $m_1 = 1$, $I_1^1(\underline{z},\underline{A},\underline{A}') = [\underline{z} = \underline{A} = \underline{A}' \wedge \Phi(\underline{z})]$ (satisfait (P) et (LI) avec $\ell' = 0$), $I_1^2 = \theta_1$, $m_0 = 2$, $I_0^2(\underline{z},\underline{A},\underline{A}') = [\varepsilon_0(\underline{z},\underline{A}) \wedge \underline{A}' = \underline{A}]$ (satisfait (P) et (C) ou (HS)), $I_0^1(\underline{z},\underline{A},\underline{A}') = [\varepsilon_0(\underline{z},\underline{A}) \wedge \neg\theta_0(\underline{z},\underline{A},\underline{A}) \wedge \exists a \in A.\ t_0(\underline{A},\underline{A}')]$ (satisfait (LI) avec $\ell' = 0$), $f_0(\underline{z},\underline{A}) = \#\underline{k}(A_{cc}\langle S,A,t,\Phi,\Psi\rangle(\underline{z}), t\mu I_{\mathrm{inter}}\langle S,A,t,\Phi,\Psi\rangle(\underline{z})^{-1})(\underline{A})$ et $I_0^0 = \theta_0$ (satisfait (C)).

□

5.3.3 PRINCIPES D'INDUCTION EQUIVALENTS GENERALISANT LA METHODE DES ASSERTIONS INTERMITTENTES DE BURSTALL

Nous dérivons maintenant une série de principes d'induction dont nous montrons la correction et la complétude sémantique donc l'équivalence au principe d'induction $(\mathcal{P}_2)$. Pour être concis, nous ne reportons pas ici toutes les alternatives concevables. En particulier, les transformations présentées en 4.2.1.2 et 5.2.3 ne seront pas répétées. Un but de la série de principes d'induction est de proposer des formalisations de plus en plus abstraites qui devraient permettre une meilleure compréhension de la méthode de Burstall. L'autre but est d'augmenter le nombre de formes de preuves permises (de manière à introduire plus de souplesse dans l'écriture des preuves, mais pas de puissance supplémentaire puisque tous ces principes d'induction sont équivalents).

Le nombre de lemmes $\langle \varepsilon_\ell, \theta_\ell \rangle$, $\ell \in \Lambda$ qui peuvent être utilisés dans le principe d'induction $(\mathcal{P}_4)$ est fini. Avec une proposition informelle telle que

"if sometime $(\underline{x} \leq X = x \leq \underline{x})$ then sometime $(\underline{x} \leq x \leq \underline{x} = X)$"

doit être comprise comme un seul lemme de nom 0, par exemple et tel que $\varepsilon_0(\underline{x},x) = [\underline{x} \leq x \leq \underline{x}]$ et $\theta_0(\underline{x},x,x') = [\underline{x} \leq x \leq \underline{x} = x']$. Si nous éliminons cette restriction sur les noms des lemmes, la proposition informelle ci-dessus peut aussi être comprise comme une représentation d'un nombre infini de lemmes de noms $\underline{x}$ tels que $\varepsilon_{\underline{x}}(x) = [\underline{x} \leq x \leq \underline{x}]$ et $\theta_{\underline{x}}(x,x') = [\underline{x} \leq x \leq \underline{x} = x']$. Ce point de vue est compatible avec le fait que le seul but de l'état initial $\underline{x}$ du programme dans le principe d'induction $(\mathcal{P}_2)$ est d'offrir la possibilité d'utiliser des bons-ordres pour l'induction sur les données qui dépendent des états initiaux du programme. Ces bons-ordres peuvent également être distingués en fau

Aussi, la proposition principale $\langle \Phi, \Psi \rangle$ n'a pas besoin d'être la conséquence d'un seul lemme $\langle \varepsilon_\pi, \theta_\pi \rangle$ comme dans $(\mathcal{B}_2)$ mais peut aussi être la conséquence de différents lemmes pour différents états initiaux du programme. Ces remarques conduisent au principe d'induction :

$$
\begin{array}{l}
[\exists \Lambda \in \underline{Ord},\ \varepsilon \in (\Lambda \to (S \to \{tt, ff\})),\ \theta \in (\Lambda \to (S^2 \to \{tt, ff\})),\ \Delta \in \underline{Ord},\ f_\varepsilon \in (\Lambda \to (S \to \Delta)),\\
m \in (\Lambda \to \omega).\\
\\
\quad \forall \Delta \in S.\ \exists \pi \in \Lambda.\ (\varepsilon_\pi(\Delta) = \Phi(\Delta) \wedge \forall \Delta' \in S.\ (\theta_\pi(\Delta, \Delta') = \Psi(\Delta, \Delta')))\\
\quad \wedge (\forall \alpha \in \Lambda.\ \exists I_\alpha \in (m_\alpha + 1 \to (S^2 \to \{tt, ff\})).\\
\quad\quad \forall i \leq m_\alpha,\ \Delta, \Delta' \in S.\\
\\
\quad\quad\quad [\varepsilon_\alpha(\Delta) \Rightarrow I_\alpha^{m_\alpha}(\Delta, \Delta)]\\
\quad\quad\quad \wedge [I_\alpha^i(\Delta, \Delta') \Rightarrow\\
\quad\quad\quad\quad (\exists \Delta'' \in S, a \in A.\ t_a(\Delta', \Delta'') \wedge \forall \Delta'' \in S, a \in A.\ [t_a(\Delta', \Delta'') \Rightarrow \exists j < i.\ I_\alpha^j(\Delta, \Delta'')])\\
\quad\quad\quad\quad \vee (\exists \alpha' \in \Lambda.\ [((\alpha' < \alpha) \vee (\alpha' = \alpha \wedge f_\alpha(\Delta') < f_\alpha(\Delta))) \wedge \varepsilon_{\alpha'}(\Delta') \wedge\\
\quad\quad\quad\quad\quad\quad \forall \Delta'' \in S.\ (\theta_{\alpha'}(\Delta', \Delta'') \Rightarrow \exists j < i.\ I_\alpha^j(\Delta, \Delta''))])\\
\quad\quad\quad\quad \vee \theta_\alpha(\Delta, \Delta')])]
\end{array}
\qquad (\mathcal{B}_3)
$$

Théorème 5.3.3 ∾1

$(\mathcal{B}_2) \Rightarrow (\mathcal{B}_3)$

Démonstration

[illegible]

de sorte que $\underline{\varepsilon} \in ((\Lambda_2 \times \Sigma) \to \Sigma)$, $\underline{\lambda} \in ((\Lambda_2 \times \Sigma) \to \Lambda_2)$ et $\forall x \in (\Lambda_2 \times \Sigma).\ [x = \underline{\lambda}(\langle \underline{\varepsilon}(x), \underline{\lambda}(x) \rangle)]$. Nous choisissons $\Lambda_3 = \Lambda_2 \times \Sigma$, $\varepsilon_{3_x}(\Delta) = [\varepsilon_{2_{\underline{\lambda}(x)}}(\delta(\underline{\varepsilon}(x)), \Delta)]$, $\theta_{3_x}(\Delta, \Delta') = [\theta_{2_{\underline{\lambda}(x)}}(\delta(\underline{\varepsilon}(x)), \Delta, \Delta')]$, $\Delta_3 = \Delta_2$, $f_{3_x}(\Delta) = [f_{2_{\underline{\lambda}(x)}}(\delta(\underline{\varepsilon}(x)), \Delta)]$, $m_{3_x} = m_{2_{\underline{\lambda}(x)}}$, $I_{3_x}^i(\Delta, \Delta') = I_{2_{\underline{\lambda}(x)}}^i(\delta(\underline{\varepsilon}(x)), \Delta, \Delta')$. Il s'ensuit que $\varepsilon_{3_{\underline{\lambda}(\langle \delta^{-1}(\Delta), \pi_2 \rangle)}}(\Delta) = \Phi(\Delta)$ et $\theta_{3_{\underline{\lambda}(\langle \delta^{-1}(\Delta), \pi_2 \rangle)}}(\Delta, \Delta') = \Psi(\Delta, \Delta')$. Les autres conditions de vérification sont évidentes à contrôler.

□

Les noms $\alpha \in \Lambda$ des lemmes $\langle \varepsilon_\alpha, \theta_\alpha \rangle$ dans $(\mathcal{B}_3)$ sont bien-ordonnés. Pour un lemme donné $\langle \varepsilon_\alpha, \theta_\alpha \rangle$ le rôle de f_α est d'introduire un bon-ordre sur les états initiaux du lemme $\langle \varepsilon_\alpha, \theta_\alpha \rangle$. Le même effet peut être obtenu en considérant non pas un seul lemme $\langle \varepsilon_\alpha, \theta_\alpha \rangle$ mais une famille de lemmes $\{\langle \varepsilon_{\alpha, f_\alpha(\Delta)}, \theta_{\alpha, f_\alpha(\Delta)} \rangle : \Delta \in S\}$. Ce point de vue est plus abstrait du fait que nous n'avons besoin que d'un seul bon-ordre $\langle W, \prec \rangle$. Il est défini par $\langle \alpha', f_{\alpha'}(\Delta') \rangle \prec \langle \alpha, f_\alpha(\Delta) \rangle$ si et seulement si $(\alpha' < \alpha \vee (\alpha' = \alpha \wedge f_{\alpha'}(\Delta') < f_\alpha(\Delta)))$ sur $W = \{\langle \alpha, f_\alpha(\Delta) \rangle : \alpha \in \Lambda \wedge \Delta \in S\}$. Aussi, à un isomorphisme près, nous pouvons utiliser des ordinaux et reformuler le principe d'induction $(\mathcal{B}_3)$ comme suit :

$$
\begin{array}{l}
[\exists \Lambda \in \underline{Ord},\ \varepsilon \in (\Lambda \to (S \to \{tt, ff\})),\ \theta \in (\Lambda \to (S^2 \to \{tt, ff\})),\ m \in (\Lambda \to \omega).\\
\\
\quad (\forall \Delta \in S.\ \exists \pi \in \Lambda.\ [(\varepsilon_\pi(\Delta) = \Phi(\Delta)) \wedge \forall \Delta' \in S.\ (\theta_\pi(\Delta, \Delta') = \Psi(\Delta, \Delta'))])\\
\quad \wedge (\forall \lambda \in \Lambda.\ \exists I_\lambda \in (m_\lambda + 1 \to (S^2 \to \{tt, ff\})).\\
\quad\quad \forall i \in m_\lambda,\ \Delta, \Delta' \in S.\\
\\
\quad\quad\quad [\varepsilon_\lambda(\Delta) \Rightarrow I_\lambda^{m_\lambda}(\Delta, \Delta)]\\
\quad\quad\quad \wedge [I_\lambda^i(\Delta, \Delta') \Rightarrow\\
\quad\quad\quad\quad (\exists \Delta' \in S, a \in A.\ t_a(\Delta', \Delta'') \wedge \forall \Delta'' \in S, a \in A.\ [t_a(\Delta', \Delta'') \Rightarrow \exists j < i.\ I_\lambda^j(\Delta, \Delta'')])\\
\quad\quad\quad\quad \vee (\exists \lambda' < \lambda.\ [\varepsilon_{\lambda'}(\Delta') \wedge \forall \Delta'' \in S.\ (\theta_{\lambda'}(\Delta', \Delta'') \Rightarrow \exists j < i.\ I_{\lambda'}^j(\Delta, \Delta''))])\\
\quad\quad\quad\quad \vee \theta_\lambda(\Delta, \Delta')])]
\end{array}
\qquad (\mathcal{B}_4)
$$

Théorème 5.3.3 v 2

$(\mathcal{B}_3) \Rightarrow (\mathcal{B}_4)$

Démonstration

$\langle \Lambda_3, \Delta_3 \rangle$ bien-ordonné par l'ordre lexicographique gauche est isomorphe à $\Delta_3 \dot{\times} \Lambda_3$ par l'isomorphisme d'ordre $\underline{\ell}(\lambda,k) = [(\Delta_3 \dot{\times} \lambda) + k]$ dont l'inverse est $\langle \underline{\xi}, \underline{\lambda} \rangle$. Nous choisissons $\Lambda_4 = \Delta_3 \dot{\times} \Lambda_3$, $\varepsilon_{4_\lambda}(\Delta) = [\varepsilon_{3_{\underline{\lambda}(\lambda)}}(\Delta) \wedge f^3_{\underline{\lambda}(\lambda)}(\Delta) = \underline{\xi}(\lambda)]$, $\theta_{4_\lambda}(\Delta,\Delta') = [\theta_{3_{\underline{\lambda}(\lambda)}}(\Delta,\Delta') \wedge f^3_{\underline{\lambda}(\lambda)}(\Delta) = \underline{\xi}(\lambda)]$, $m_{4_\lambda} = m_{3_{\underline{\lambda}(\lambda)}}$, $I^i_{4_\lambda}(\Delta,\Delta') = [I^i_{3_{\underline{\lambda}(\lambda)}}(\Delta,\Delta') \wedge f^3_{\underline{\lambda}(\lambda)}(\Delta) = \underline{\xi}(\lambda)]$.

□

Les propriétés de fatalité des programmes ont été spécifiées par des paires $\langle \Phi, \Psi \rangle$ où Φ est une condition sur les états initiaux et Ψ une relation entre états finaux et initiaux comme dans la méthode de Burstall [74]. Cependant, une seule relation binaire suffit car Ψ est fatale pour $\langle s, A, \Sigma < s, A, t, \Phi \rangle\rangle$ si et seulement si $\Psi'(\Delta,\Delta') = [\Phi(\Delta) \Rightarrow \Psi(\Delta,\Delta')]$ est fatale pour $\langle s, A, \Sigma < s, A, t, \Psi \rangle\rangle$. Aussi, nous dérivons le principe d'induction plus abstrait

$[\exists \Lambda \in \underline{Ord},\ \theta \in (\Lambda \to (S^2 \to \{\Psi, \overline{\Psi}\})),\ m \in (\Lambda \to \omega).$

$(\forall \Delta \in S.\ \exists \pi \in \Lambda.\ \forall \Delta' \in S.\ [\theta_\pi(\Delta,\Delta') = (\Phi(\Delta) \Rightarrow \Psi(\Delta,\Delta'))]$

$\wedge (\forall \lambda \in \Lambda.\ \exists I_\lambda \in (m_\lambda \to (S^2 \to \{\Psi, \overline{\Psi}\}))$.

$\forall i < m_\lambda,\ \Delta,\Delta' \in S.$

(P) $I^{m_\lambda}_\lambda(\Delta,\Delta)$

$\wedge$ $I^{i+1}_\lambda(\Delta,\Delta') \Rightarrow$

$\exists \xi < \ell(\lambda).\ \forall \Delta'' \in S.\ [\theta_\xi(\Delta',\Delta'') \Rightarrow \exists j < \ell.\ I^j_\lambda(\Delta,\Delta'')]$

(HS) $\vee\ (\exists \lambda' < \lambda.\ \forall \Delta'' \in S.\ [\theta_{\lambda'}(\Delta',\Delta'') \Rightarrow \exists j < \ell.\ I^j_\lambda(\Delta,\Delta'')]$

(C) $\vee\ \theta_\lambda(\Delta,\Delta')]]$

Théorème 5.3.3 v 2

$(\mathcal{B}_4) \Rightarrow (\mathcal{B}_5)$

Démonstration

Choisir $\Lambda_5 = \Lambda_4$, $\theta_{5_\lambda}(\Delta,\Delta') = [\varepsilon_{4_\lambda}(\Delta) \Rightarrow \theta_{4_\lambda}(\Delta,\Delta')]$, $m_5 = m_4$ et $I^i_{5_\lambda}(\Delta,\Delta') = [\varepsilon_{4_\lambda}(\Delta) \Rightarrow I^i_{4_\lambda}(\Delta,\Delta')]$.

□

Si dans le principe d'induction $(\mathcal{B}_5)$, nous considérons la preuve d'un lemme θ_λ donné et que cette preuve peut être faite sous (LI), alors les conditions de vérification (P), (HS) et (C) ressemblent fortement aux conditions de vérification de la méthode de preuve de Floyd [67] telle qu'elle est formalisée par le principe d'induction $(\mathcal{F}_5)$ dans le paragraphe 5.2.3. Autrement dit, dans l'hypothèse d'induction $I^i_\lambda(\Delta,\Delta')$, i joue le rôle d'un entier non-négatif qui décroît strictement à chaque pas du programme. Par comparaison avec la méthode de Floyd, nous observons que $(\mathcal{B}_5)$ impose deux restrictions inutiles sur i : m_λ est une borne du nombre de pas du programme et ce nombre est indépendant de l'état initial considéré, m_λ et donc i doit être un entier (de sorte que, par exemple, le nondéterminisme infini ne puisse être traité dans (LI)).

Nous supprimons d'abord la première restriction en choisissant ℓ comme étant un ordinal :

Lemme 5.3.4 v 1

(a) $[(\mathcal{B}_2) \Rightarrow ((\mathcal{B}_2) \text{ avec } m \in (\Lambda \to Ord))]$, $i = 2, \ldots, 5$

(b) $[((\mathcal{B}_2) \text{ avec } m \in (\Lambda \to \underline{Ord})) \Rightarrow ((\mathcal{B}_{i+1}) \text{ avec } m \in (\Lambda \to \underline{Ord}))]$, $i = 2,3,4$

Démonstration

(a) est trivial parceque $\omega \in \underline{Ord}$ donc $\omega \in \underline{Ord}$.

(b) résulte des preuves des théorèmes 5.3.3.v.1, 5.3.3.v.2 et 5.3.3.v.3 qui n'utilisent jamais le fait que $n_\ell \in \omega$ mais seulement le fait que $\langle n_\ell+1, < \rangle$ est bien-fondé (ceci demeure vrai quand $n_\ell \in \underline{Ord}$ et $n_{\ell+1}$ est l'ordinal successeur de n_ℓ).

□

Nous supprimons ensuite la seconde restriction, en choisissant un "nombre maximum de pas du programme" qui peut être différent pour chaque état initial $\underline{s}$. (Le "nombre de pas du programme" ne doit pas être interprété à la lettre mais comme $\underline{rk}(\underline{Acc}\langle S,A,t,\Phi,\Psi,\Psi\rangle(\underline{s}), t^{-1}\underline{Inter}\langle S,A,t,\Phi,\Psi,\Psi\rangle(\underline{s})^{-1})(\underline{s})$)

$$
\begin{array}{l}
[\,\exists \Lambda \in \underline{Ord},\ \theta \in (\Lambda \to (S^2 \to \{\Psi, \mathit{ff}\})),\ \Delta \in \underline{Ord} \to I \in (\Lambda \to (\Delta \times S \times S \to \{\Psi, \mathit{ff}\})).\\
\quad (\forall a \in S.\ \exists \pi \in \Lambda.\ \forall a' \in S.\ [\,\theta_\pi(a,a') = (\Phi(a) \Rightarrow \Psi(a,a'))\,])\\
\quad \wedge (\forall \lambda \in \Lambda,\ a,a' \in S,\ \delta' \in \Delta. \qquad (\mathcal{B}_\sharp)\\
\qquad [\,\exists \delta \in \Delta.\ I_\lambda(\delta,a,a)\,]\\
\qquad \wedge [\,I_\lambda(\delta',a,a') \Longrightarrow\\
\qquad\quad (\exists a'' \in S, a \in A.\ t_a(a',a'') \wedge \forall a'' \in S, a \in A.\,[t_a(a',a'') \Rightarrow \exists \delta'' < \delta'.\ I_\lambda(\delta'',a,a'')])\\
\qquad\quad \vee (\exists \lambda' < \lambda.\ \forall a'' \in S.\,[\theta_{\lambda'}(a',a'') \Rightarrow \exists \delta'' < \delta'.\ I_\lambda(\delta'',a,a'')])\\
\qquad\quad \vee \theta_\lambda(a,a')])]
\end{array}
$$

Théorème 5.3.3.v.5

$(\mathcal{B}_5) \Rightarrow (\mathcal{B}_\sharp)$

Démonstration

Choisir $\Lambda_\sharp = \Lambda_5$, $\theta_\sharp = \theta_5$, $\Delta_\sharp = \omega$, $I_{\sharp\lambda}(\delta,a,a') = [\,\delta < m_{\sharp_\lambda} \wedge I^\delta_{5_\lambda}(a,a')\,]$.

□

Dans la suite nous utiliserons le plus souvent le principe d'induction suivant :

$$
\begin{array}{l}
[\,\exists \Lambda \in \underline{Ord},\ \theta \in (\Lambda \to (S \times S \to \{\Psi, \mathit{ff}\})),\ \pi \in \Lambda,\ \Delta \in \underline{Ord},\ I \in (\Lambda \to (\Delta \times S \times S \to \{\Psi, \mathit{ff}\})).\\
(\mathcal{B}_\sharp.1) \quad \forall a,a' \in S.\,(\,\theta_\pi(a,a') = [\Phi(a) \Rightarrow \Psi(a,a')]\,)\\
(\mathcal{B}_\sharp.2) \quad \wedge (\forall \lambda \in \Lambda.\ \forall a \in S.\ \exists \delta \in \Delta.\ I_\lambda(\delta,a,a))\\
(\mathcal{B}_\sharp.3) \quad \wedge (\forall \lambda \in \Lambda,\ a,a' \in S,\ \delta' \in \Delta. \qquad (\mathcal{B}_\sharp)\\
\qquad I_\lambda(\delta',a,a') \Rightarrow\\
(\mathcal{B}_\sharp.3.a) \qquad [\,(\exists a'' \in S, a \in A.\ t_a(a',a'') \wedge \forall a'' \in S, a \in A.\,[t_a(a',a'') \Rightarrow \exists \delta'' < \delta'.\ I_\lambda(\delta'',a,a'')])\\
(\mathcal{B}_\sharp.3.b) \qquad\quad \vee (\exists \lambda' < \lambda.\ \forall a'' \in S.\,[\theta_{\lambda'}(a',a'') \Rightarrow \exists \delta'' < \delta'.\ I_\lambda(\delta'',a,a'')])\\
(\mathcal{B}_\sharp.3.c) \qquad\quad \vee \theta_\lambda(a,a')])]
\end{array}
$$

Théorème 5.3.3.v.6

$(\mathcal{B}_6) \Rightarrow (\mathcal{B}_\sharp)$

Démonstration

Choisir $\Lambda_\sharp = \Lambda_{5_\pi}$, $\Lambda_\sharp = (\Lambda_{5_\pi} \cup \Gamma_{5_\pi}) = \Lambda_{5_\pi+1}$, $\theta_{\pi_\sharp} = \theta_{5_\pi}$ quand $\lambda \in \Lambda_{5_\pi}$ et $\theta_{\pi_\sharp}(a,a') = [\Phi(a) \Rightarrow \Psi(a,a')]$, $\Delta_\sharp = (\Delta_{5_\pi} \cup \omega)$, $I_{\sharp_\lambda}(\delta',a,a') = I_{5_\lambda}(\delta',a,a')$ quand $\lambda \in \Lambda_{5_\pi}$, $I_{\sharp_{\pi_\sharp}}(\delta',a,a') = \mathit{ff}$ si $\delta' \geq \omega$, $I_{\sharp_{\pi_\sharp}}(\lambda,a,a') = [a = a']$, $I_{\sharp_{\pi_\sharp}}(\gamma,a,a') = \theta_{\sharp_{\pi_\sharp}}(a,a')$.

L'utilisation de bons-ordres (ou à des isomorphismes d'ordre près, d'ordinaux) dans $(\mathcal{B}_6)$ n'est pas obligatoire. Des relations bien-fondées peuvent aussi bien servir de base pour l'induction.

De plus, comme noté par Schwarz [77], la méthode de Burstall peut être expliquée comme la déduction mathématique de théorèmes à partir d'axiomes spécifiant l'effet des commandes élémentaires du programme. Cette explication informelle de la méthode de Burstall peut être formalisée en considérant la relation de transition dans les principes d'induction précédents comme un ensemble d'axiomes ou encore comme un lemme donné à partir duquel les autres lemmes dérivent. Une différence (qui n'est pas prise en compte par Schwarz [77] qui considère seulement des programmes totaux déterministes) est que la fatalité de t pour $\langle s,A,\Sigma\rangle$ $\langle s,A,t,t\rangle\rangle$ n'est vraie que pour les états qui ont au moins un successeur. De plus, le processus de déduction (que Schwarz [77] ne spécifie pas) est toujours réductible à l'induction transfinie.

Finalement, la proposition principale $\theta_\pi(A,A') = [\Phi(A) \Rightarrow \Psi(A,A')]$ peut toujours être choisie comme un des lemmes intervenant dans la preuve.

Ces remarques conduisent au principe d'induction suivant :

$$[\exists \Lambda, \prec, \mu\in\Lambda, \pi\in(\Lambda\cup\mu),\ \Delta,\ <,\ \theta\in(\Lambda\to(S^2\to\{tt,ff\})),\ I\in(\Lambda\to(\Delta\times S\times S\to\{tt,ff\})).$$

$$\omega^{\Lambda,\mu}_{\prec}(\Lambda,\prec,\mu) \wedge \theta_\mu = (\exists a\in A.\ t_a) \wedge \omega^{\Delta}_{<}(\Delta,<) \wedge [\forall s,s'\in S.\ \theta_\pi(s,s') = [\Phi(s)\Rightarrow\Psi(s,s')]]$$

$$\wedge\ (\forall\lambda\in(\Lambda\cup\mu),\ \delta,\delta'\in\Delta,\ s,s'\in S,\ \delta'\in\Delta.$$

$$[\exists\delta\in\Delta.\ I_\lambda(\delta,s,s)]$$

$$\wedge\ [I_\lambda(\delta,s,s') \Rightarrow$$

$$(\exists\lambda'\in\Lambda.\ \prec(\lambda',\lambda) \wedge ((\lambda'=\mu) \Rightarrow [\exists a\in A.\ t_a(s,s')]) \wedge$$

$$\forall s''\in S.\ (\theta_{\lambda'}(s',s'') \Rightarrow [\exists\delta'\in\Delta.\ (\delta'<\delta \wedge I_\lambda(\delta',s,s''))]))]$$

$$\vee\ \ldots\ (\lambda=\mu)]$$

Remarquons que la condition $[\lambda'=\mu]$ sous laquelle $[\exists s''\in S.\ \theta_{\lambda'}(s',s'')]$ devrait être vrai est optionnelle. Lorsqu'elle est absente, la condition de vérification est simplement redondante quand $\lambda'\neq\mu$.

Théorème 5.3.3.v7

$$(\mathcal{B}_7) \Longrightarrow (\mathcal{B}_8)$$

Démonstration

Choisir $\mu=\Lambda_7$, $\Lambda_8=(\Lambda_7\cup\{\mu\})=(\Lambda_{7+1})$, $\pi_8=\pi_7$, $\Delta_8=\Delta_7$, $\lambda'\prec_8\lambda=$ $[\lambda\in\Lambda_7 \wedge ((\lambda'=\mu)\vee(\lambda'\in\Lambda_7 \wedge \lambda'<\lambda))]$, $<_8=<_{7+1}=<$, $\theta_{8\lambda}(s,s')=[(\lambda=\mu \wedge \exists a\in A.\ t_a(s,s'))\vee(\lambda\in\Lambda_7 \wedge \theta_{7\lambda}(s,s'))]$, $I_{8\lambda}(\delta,s,s') = [(\lambda\in\Lambda_7 \wedge I_{7\lambda}(\delta,s,s'))\vee(\lambda=\mu)]$.

□

Dans le principe d'induction $(\mathcal{B}_8)$, la condition de vérification $[\exists\delta\in\Delta.\ I_\lambda(\delta,s,s)]$ implique que le lemme θ_λ est fatal pour $\langle s,A,\Sigma\langle s,A,t,t\rangle\rangle$. Excepté pour la proposition principale θ_π, cette condition n'est pas nécessaire. Nous avons besoin seulement du fait que θ_λ doit être fatal pour les états particuliers pour lesquels il est utilisé. Aussi, la condition de vérification $[\exists\delta\in\Delta.\ I_\lambda(\delta,s,s)]$ de $(\mathcal{B}_8)$ peut être affaiblie dans :

$[\exists \Lambda \in \underset{\sim}{Ord},\ \theta \in (\Lambda \to (S^2 \to \{\Psi, \neg\Psi\})),\ \pi \in \Lambda,\ \Delta \in \underset{\sim}{Ord},\ I \in (\Lambda \to (\Delta \times S \times S \to \{\Psi, \neg\Psi\}))$.

$(\forall A, A' \in S.\ (\theta_\pi(A,A') = [\phi(A) \Rightarrow \Psi(A,A')]))$

$\wedge (\forall A \in S.\ \exists \delta \in \Delta.\ I_\pi(\delta, A, A))$

$\wedge (\forall \lambda \in \Lambda,\ A, A' \in S,\ \delta' \in \Delta.$

$[I_\lambda(\delta', A, A') \Rightarrow$

$(\exists A'' \in S, a \in A.\ t_a(A', A'') \wedge \forall A'' \in S, a \in A.[t_a(A', A'') \Rightarrow \exists \delta'' < \delta'.\ I_\lambda(\delta'', A, A'')])$

$\vee (\exists \lambda' < \lambda.\ [\exists \delta \in \Delta.\ I_{\lambda'}(\delta, A, A') \wedge \forall A'' \in S.\ [\theta_{\lambda'}(A', A'') \Rightarrow \exists \delta'' < \delta'.\ I_\lambda(\delta'', A, A'')]])$

$\vee\ \theta_\lambda(A, A')]])]$ $(\mathcal{B}_3)$

Théorème 5.3.3 v8

$(\mathcal{B}_3) \Rightarrow (\mathcal{B}_3)$

Démonstration

Nous montrons d'abord que si $0 < \varepsilon_0 < \gamma$ et $0 < \varepsilon_1 < \gamma$ alors $((\gamma \times \delta_0) + \varepsilon_0) < ((\gamma \times \delta_1) + \varepsilon_1)$ si et seulement si $((\delta_0 < \delta_1) \vee (\delta_0 = \delta_1 \wedge \varepsilon_0 < \varepsilon_1))$.

Si $\delta_0 < \delta_1$ alors $((\gamma \times \delta_0) + \varepsilon_0) < ((\gamma \times \delta_0) + \gamma) = (\gamma \times (\delta_0 + 1)) \leq (\gamma \times \delta_1) < ((\gamma \times \delta_1) + \varepsilon_1)$. Si $(\delta_0 = \delta_1) \wedge (\varepsilon_0 < \varepsilon_1)$ alors $(\gamma \times \delta_0) = (\gamma \times \delta_1)$ et donc $((\gamma \times \delta_0) + \varepsilon_0) < ((\gamma \times \delta_1) + \varepsilon_1)$.

Si inversement $\neg((\delta_0 < \delta_1) \vee (\delta_0 = \delta_1 \wedge \varepsilon_0 < \varepsilon_1))$ alors soit $\delta_0 = \delta_1$ et $\varepsilon_0 = \varepsilon_1$ de sorte que $\alpha = ((\gamma \times \delta_0) + \varepsilon_0) = ((\gamma \times \delta_1) + \varepsilon_1) = \beta$ et $\alpha \not< \beta$; ou bien $(\delta_0 > \delta_1) \vee (\delta_0 = \delta_1 \wedge \varepsilon_0 > \varepsilon_1)$ de sorte que d'après la première partie de la preuve (avec α et β interchangés) nous avons $\beta < \alpha$ donc $\alpha \not< \beta$.

Montrons maintenant qu'étant donné une relation [illegible] il existe une fonction $\tau(N, \prec)$ [illegible] de N dans $(Ord, <)$ des ordinaux.

Soit $E(N, \prec) \in (\underset{\sim}{rk}(N, \prec) \to \{X : X \subseteq N\})$ définie par $E(N, \prec)(\alpha) = \{x \in N : \underset{\sim}{rk}(N, \prec)(x) < \alpha\}$ [illegible] $\exists x \in N.\ \exists x \in \underset{\sim}{rk}(N, \prec).\ [x \in E(N, \prec)(\alpha)]$.

L'axiome du choix il y a un ordre total $\ll(W, \prec)(x)$ sur $E(W, \prec)(x)$.

Définissons $\varepsilon(N, \prec)(x, y) = \underset{\sim}{\xi}[\exists z \in N.\ \underset{\sim}{rk}(z) < \underset{\sim}{rk}(x).\ \ll(N, \prec)(x)[z] \wedge y \text{ est } \underset{\sim}{\xi}(N, \prec)(z)]$ si x est $\underset{\sim}{\xi}(N, \prec)(x)$ et $\varepsilon(N, \prec)(x) = (\varepsilon(N, \prec)(x, x) + 1)$ de sorte que $\forall x \in W.\ [0 < \varepsilon(N, \prec)(x)]$. Définissons $\gamma(N, \prec) = \sup^+\{\varepsilon(W, \prec)(x) : x \in W\}$ de sorte que $\forall x \in W.\ [\varepsilon(W, \prec)(x) < \gamma(W, \prec)]$ et $\tau(W, \prec)(x) = ((\gamma(W, \prec) \times \underset{\sim}{rk}(N, \prec)(x)) + \varepsilon(N, \prec)(x))$.

Si $x \prec y$ alors $\underset{\sim}{rk}(W, \prec)(x) < \underset{\sim}{rk}(W, \prec)(y)$ et donc d'après le lemme $\tau(W, \prec)(x) < \tau(W, \prec)(y)$. Si $\tau x = \tau(W, \prec)(x) = \tau(W, \prec)(y) = \tau y$ alors $x \not< y$ et $y \not< x$ de sorte que d'après le lemme $\underset{\sim}{rk}(W, \prec)(x) = \underset{\sim}{rk}(W, \prec)(y)$ et $\varepsilon(W, \prec)(x) = \varepsilon(W, \prec)(y)$ d'où $\varepsilon(W, \prec)(x, x) = \varepsilon(W, \prec)(x, y)$. Ceci implique que nous n'avons ni $x \ll(W, \prec)(\tau x)\ y$ ni $y \ll(W, \prec)(\tau x)\ x$ et puisque $x, y \in E(W, \prec)(\tau x)$ qui est totalement ordonné par $\ll(W, \prec)(\tau x)$ nous concluons que $x = y$.

La preuve $(\mathcal{B}_3) \Rightarrow (\mathcal{B}_3)$ est maintenant immédiate si nous choisissons $\Lambda_3 = \sup^+\{\tau(\Lambda_3 \vee \mu, \prec_3)(x) : x \in (\Lambda_3 \vee \mu)\}$, $\pi_3 = \tau(\Lambda_3 \vee \mu, \prec_3)(\pi_3)$, $\theta_{3\lambda}(A, A') = [\exists a \in (\Lambda_3 \vee \mu).\ (\lambda = \tau(\Lambda_3 \vee \mu, \prec_3)(a) \wedge \theta_{3a}(A, A'))]$, $\Delta_3 = \sup^+(\tau(\Delta_3, <_3)(x) : x \in \Delta_3\}$ et $I_{3\lambda}(\delta, A, A') = [\exists a \in (\Lambda_3 \vee \mu),\ d \in \Delta_3.\ (\lambda = \tau(\Lambda_3 \vee \mu, \prec_3)(a) \wedge \delta = \tau(\Delta_3, <_3)(d) \wedge I_{3a}(d, A, A'))]$.

□

L'utilisation des lemmes $\theta_\lambda(A, A')$ dans le principe d'induction $(\mathcal{B}_3)$ est redondante parceque nous pouvons utiliser à la place une certaine assertion intermittente $I_\lambda(\delta, A, A')$ pour un certain δ telle que $I_\lambda(\delta, A, A') \Rightarrow \theta_\lambda(A, A')$. En conséquence, nous pouvons choisir $\delta = 0$ et donc le principe d'induction $(\mathcal{B}_3)$ peut être simplifié en :

$$[\exists \Lambda \in \underline{Ord},\ \pi \in \Lambda,\ \Delta \in \underline{Ord},\ I \in (\Lambda \to (\Delta \times S \times S \to \{tt, ff\})).$$

$$(\forall \lambda, \lambda' \in S.\ [I_\pi(0, \lambda, \lambda') = (\phi(\lambda) \to \psi(\lambda, \lambda'))])$$

$$\wedge\ (\forall \lambda \in S.\ \exists \delta \in \Delta.\ I_\pi(\delta, \lambda, \lambda)) \qquad (\mathcal{B}_{10})$$

$$\wedge\ (\forall \lambda \in \Lambda,\ \lambda, \lambda' \in S,\ \delta' \in (\Delta \cup 0).$$

$$[I_\lambda(\delta', \lambda, \lambda') \Longrightarrow$$

$$(\exists \lambda'' \in S, a \in A.\ t_a(\lambda', \lambda'') \wedge \forall \lambda'' \in S, a \in A.\ [t_a(\lambda', \lambda'') \Rightarrow \exists \delta'' < \delta'.\ I_\lambda(\delta'', \lambda, \lambda'')])$$

$$\vee\ (\exists \lambda' < \lambda.\ [\exists \delta \in \Delta.\ I_{\lambda'}(\delta, \lambda', \lambda') \wedge \forall \lambda'' \in S.\ [I_{\lambda'}(0, \lambda', \lambda'') \Rightarrow \exists \delta'' < \delta'.\ I_\lambda(\delta'', \lambda, \lambda'')]])]]$$

Théorème 5.3.3~9

$(\mathcal{B}_9) \Longrightarrow (\mathcal{B}_{10})$

Démonstration

Choisir $\Lambda_{10} = \Lambda_9$, $\pi_{10} = \pi_9$, $\Delta_{10} = \Delta_9$, $I_{10_\lambda}(\delta, \lambda, \lambda') = [(\delta = 0 \wedge \theta_{9_\lambda}(\lambda, \lambda')) \vee (\delta > 0 \wedge I_{9_\lambda}(\delta, \lambda, \lambda'))]$

□

Comme le montre cette succession de transformations, la preuve qu'on ($\mathcal{B}_{10}$), un état λ' satisfaisant $I_\lambda(\delta', \lambda, \lambda')$ conduit fatalement à un état λ'' tel que $\theta_\lambda(\lambda, \lambda'')$ soit vrai met en jeu une induction sur des parties de chemins d'exécution traduite par δ' et une induction sur les données traduite par λ. Pour pouvoir établir une comparaison avec la méthode de Floyd-[54], ces deux cas peuvent être réduits à une induction sur des coeurs, en faisant abstraction de la nature de la liaison entre λ et λ' :

$$[\exists \Gamma \in \underline{Ord},\ I \in (\Gamma \times S \times S \to \{tt, ff\}),\ \sigma \in (\Gamma \to \Gamma).$$

$$(\mathcal{P}) \qquad (\forall \lambda \in S.\ \exists \gamma \in \Gamma.\ [I(\gamma, \lambda, \lambda) \wedge \forall \lambda \in S.\ (I(\sigma(\gamma), \lambda, \lambda') \Rightarrow [\phi(\lambda) \to \psi(\lambda, \lambda')]]])$$

$$\wedge\ (\forall \gamma' \in \Gamma,\ \lambda, \lambda' \in S. \qquad (\mathcal{B}_{11})$$

$$[I(\gamma', \lambda, \lambda') \Longrightarrow$$

$$(HS) \qquad (\exists \lambda'' \in S, a \in A.\ t_a(\lambda', \lambda'') \wedge \forall \lambda'' \in S, a \in A.\ [t_a(\lambda', \lambda'') \Rightarrow \exists \gamma'' < \gamma'.\ (\sigma(\gamma'') = \sigma(\gamma') \wedge I(\gamma'', \lambda, \lambda''))])$$

$$(LI) \qquad \vee\ (\exists \gamma < \gamma'.\ [I(\gamma, \lambda', \lambda') \wedge \forall \lambda'' \in S.\ [I(\sigma(\gamma), \lambda', \lambda'') \Rightarrow \exists \gamma'' < \gamma'.\ (\sigma(\gamma'') = \sigma(\gamma') \wedge I(\gamma'', \lambda, \lambda''))])$$

$$(C) \qquad \vee\ I(\sigma(\gamma'), \lambda, \lambda')]]]$$

Théorème 5.3.3~10

$(\mathcal{B}_{10}) \Longrightarrow (\mathcal{B}_{11})$

Démonstration

Soit $\preceq(\langle \lambda, \delta \rangle) = [(\Delta_{10} \times \lambda) + \delta]$ l'isomorphisme d'ordre entre $\Lambda_{10} \times \Delta_{10}$ bien-ordonné par l'ordre lexicographique gauche $\langle \lambda', \delta' \rangle \prec \langle \lambda, \delta \rangle$ si et seulement si $((\lambda' < \lambda) \vee (\lambda' = \lambda \wedge \delta' < \delta))$ et $\Gamma_{11} = (\Delta_{10} \times \Lambda_{10})$ bien-ordonné par $\prec$. Soit $\langle \underline{\alpha}, \underline{\delta} \rangle$ l'inverse de $\preceq$ de sorte que $\forall \lambda \in \Lambda_{10},\ \delta \in \Delta_{10}.\ (\lambda = \underline{\alpha}(\preceq(\langle \lambda, \delta \rangle)) \wedge \delta = \underline{\delta}(\preceq(\langle \lambda, \delta \rangle)))$ et $\forall \gamma \in \Gamma_{11}.\ \gamma = \preceq(\langle \underline{\alpha}(\gamma), \underline{\delta}(\gamma) \rangle)$. Choisir $I_{11}(\gamma, \lambda, \lambda') = I_{10_{\underline{\alpha}(\gamma)}}(\underline{\delta}(\gamma), \lambda, \lambda')$ et $\sigma(\gamma) = \preceq(\langle \underline{\alpha}(\gamma), 0 \rangle)$.

□

En utilisant la généralisation obtenue ($\mathcal{B}_{11}$) de la méthode de Burstall, nous pouvons la comparer équitablement avec la généralisation naturelle ($\mathcal{W}_2$) de la méthode de Floyd. Pour ($\mathcal{W}_2$), la clause (LI) est inutilisée (de sorte que nous pouvons toujours choisir $\sigma(\delta) = 0$). Par conséquent la différence cruciale entre les méthodes de Floyd et de Burstall, n'est l'utilisation d'assertions invariantes au lieu d'assertions intermittentes,

ni l'utilisation d'une induction sur les calculs au lieu d'une induction sur les données mais bien l'introduction de la récursivité.

L'équivalence des principes d'induction $(\mathcal{B}_2), \ldots, (\mathcal{B}_{11})$ vient de :

Théorème 5.3.3 N11 (Correction)

$(\mathcal{B}_{11}) \Rightarrow (\Psi$ est fatale pour $\langle S, A, \Sigma\langle S, A, t, \Phi\rangle\rangle)$

Démonstration

Posant $\varepsilon^{\gamma}_{A}(A') \in I(\gamma, A, A')$, nous démontrons par induction sur $(\Gamma, <)$ que $\forall \gamma \in \Gamma, \sigma \in S, p \in \Sigma\langle S, A, t, \varepsilon^{\gamma}_{A}\rangle . \exists i \in |p| . I(\sigma(\gamma), A, p_i)$. Supposons le vrai pour $\gamma' < \gamma$. Par l'absurde, soient $\sigma \in S$, $p \in \Sigma\langle S, A, t, \varepsilon^{\gamma}_{A}\rangle$ tels que $\forall i \in |p| . \neg I(\sigma(\gamma), A, p_i)$. Pour obtenir une contradiction, nous construisons une séquence infinie $\langle\langle i_R, \gamma_R\rangle : R \in \omega\rangle$ telle que $\forall R \geq 0 . [I(\gamma_R, A, p_{i_R}) \wedge \sigma(\gamma_R) = \sigma(\gamma) \wedge \gamma_R > \gamma_{R+1}]$. Choisir $\gamma_0 = \gamma$ et $i_0 = 0$. Si la séquence est construite jusqu'au point k alors $I(\gamma_k, A, p_{i_k})$ satisfait (HS), (LI) ou (C). (C) est impossible (car $I(\sigma(\gamma_k), A, p_{i_k})$ impliquerait $I(\sigma(\gamma), A, p_{i_k})$). Dans le cas (HS), $\exists \sigma'' \in S, a \in A . t_a(p_{i_k}, A'')$ implique que $i_{k+1} = (i_k + 1) \in |p|$. Donc $t_{p_{i_k}}(p_{i_k}, p_{i_{k+1}})$ implique $\exists \gamma_{k+1} < \gamma_k . (\sigma(\gamma_{k+1}) = \sigma(\gamma_k) = \sigma(\gamma) \wedge I(\gamma_{k+1}, A, p_{i_{k+1}}))$. Dans le cas (LI), il existe $\gamma' < \gamma_k$ tel que $I(\gamma', p_{i_k}, p_{i_k})$. Donc par hypothèse d'induction $\exists j \in |p^{>i_k}| . I(\sigma(\gamma'), (p^{>i_k})_0, (p^{>i_k})_j)$. Si nous posons $i_{k+1} = (i_k + j)$ alors nous avons $I(\sigma(\gamma'), p_{i_k}, p_{i_{k+1}})$ d'où $\exists \gamma_{k+1} < \gamma_k . [I(\gamma_{k+1}, A, p_{i_{k+1}}) \wedge \sigma(\gamma_{k+1}) = \sigma(\gamma_k) = \sigma(\gamma)]$. C.Q.F.D.

Traduisant $\ldots p \in \Sigma\langle S, A, t, \Phi\rangle$ et $\ldots \exists \gamma \in \Gamma . I(\sigma(\gamma), p_0, p_0)$ de sorte que $p \in \Sigma\langle S, A, t, \varepsilon^{\gamma}_{p_0}\rangle$ et d'après le lemme ci-dessus, $\exists i \in |p| . I(\sigma(\gamma), p_0, p_i) \ldots$ D'où $\ldots \exists i \in |p| . \Psi(\sigma, p_i)$ et $\ldots \exists i \in |p| . \neg \Psi(\sigma, p_i)$

□

Exemple 5.3.3-1 (Preuve d'un programme de parcours d'arbre en utilisant le principe d'induction $(\mathcal{B}_7)$)

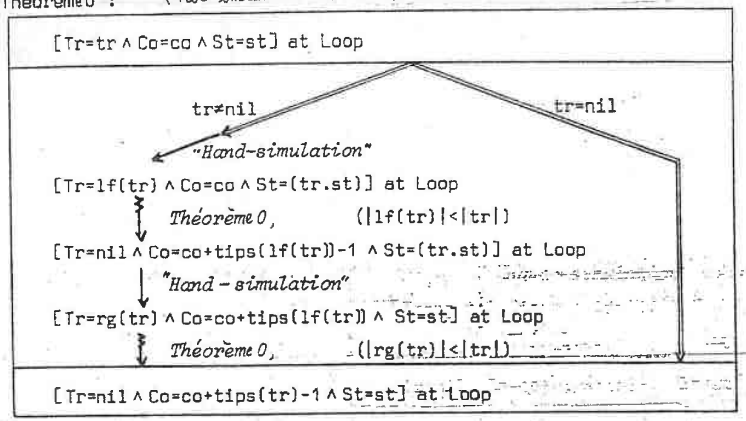
Suite à l'exemple 5.2.3-1 dans lequel nous avons démontré la correction totale du programme suivant :

```
Start:  St=();  Co:=0;
Loop:   if Tr ≠ nil
            then begin  Push Tr onto St;
                        Tr := lf(Tr);  goto Loop
                 end
            else begin  Co:=Co+1;
                        if St=() then goto Finish;
                        Pop Tr from St;
                        Tr:= rg(Tr);  goto Loop
                 end;
Finish:
```

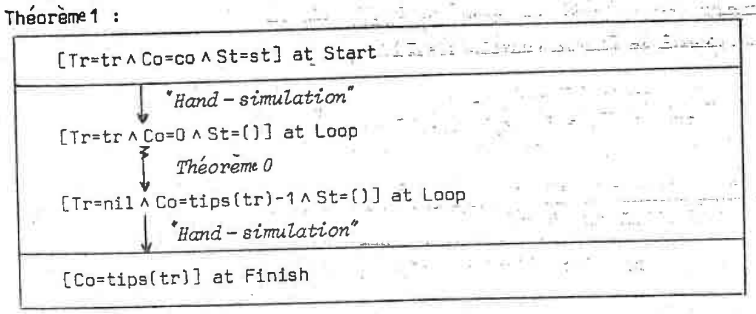
en utilisant le principe d'induction $(\mathcal{B}_6)$ correspondant à une preuve par la méthode de Floyd, nous montrons que la preuve que donne Burstall [74] de ce programme se ramène au principe d'induction $(\mathcal{B}_7)$. Cette preuve peut se formuler à l'aide d'une charte de preuve (que nous formaliserons dans le paragraphe suivant) comme suit :

Loop: $|Tr| = ((Tr = nil) \rightarrow 0 \mid (1 + |lf(Tr)| + |rg(Tr)|))$

Théorème 0 : (Par induction sur |tr|)



Théorème 1 :



La preuve du théorème i, $i=0,1$ commence par une hypothèse de la forme :

$[Tr=tr \wedge Co=co \wedge St=st]$ at L

il prend pour définition les assertions intermittentes obtenues par évaluation symbolique, par application d'une règle donnée ou par application récursive du théorème i. Puis les étapes de preuve suivantes sont [illegible] distance $d \in [0, n_i[$ du point de sortie de la chaîne de preuve. [illegible]

et doit être comprise comme une abréviation de :

"if sometime $[Tr=tr \wedge Co=co \wedge St=st]$ at L
then sometime $[Tr=f_d(tr) \wedge Co=g_d(tr,co) \wedge St=h_d(tr,st)]$ at L_d"

La chaîne de preuve elle-même peut s'exprimer sous une forme relationnelle si nous groupons ces assertions intermittentes sous forme disjonctive :

$PL_i(\delta', \langle \ell, tr, co, st\rangle, \langle \ell', tr', co', st'\rangle) =$

$[(\ell = L) \Rightarrow (\bigvee_{d=0}^{n_i} [\delta' = d \wedge \ell' = L_d \wedge tr' = f_d(tr) \wedge co' = g_d(tr,co) \wedge st' = h_d(tr,st)])]$

Plus précisément, nous avons :

$PL_0(\delta', \langle \ell, tr, co, st\rangle, \langle \ell', tr', co', st'\rangle) =$

$[(\ell = Loop) \Rightarrow$
$([\delta' = 4 \wedge \ell' = Loop \wedge tr' = tr \wedge co' = co \wedge st' = st]$
$\vee [\delta' = 3 \wedge \ell' = Loop \wedge tr' = lf(tr) \wedge co' = co \wedge st' = (tr.st)]$
$\vee [\delta' = 2 \wedge \ell' = Loop \wedge tr' = nil \wedge co' = (co + tips(lf(tr)) - 1) \wedge st' = (tr.st)]$
$\vee [\delta' = 1 \wedge \ell' = Loop \wedge tr' = rg(tr) \wedge co' = (co + tips(lf(tr)) \wedge st' = st]$
$\vee [\delta' = 0 \wedge \ell' = Loop \wedge tr' = nil \wedge co' = (co + tips(tr) - 1) \wedge st' = st])]$

$PL_1(\delta', \langle \ell, tr, co, st\rangle, \langle \ell', tr', co', st'\rangle) =$

$[(\ell = Start) \Rightarrow$
$([\delta' = 3 \wedge \ell' = Start \wedge tr' = tr \wedge co' = co \wedge st' = st]$
$\vee [\delta' = 2 \wedge \ell' = Loop \wedge tr' = tr \wedge co' = 0 \wedge st' = ()]$
$\vee [\delta' = 1 \wedge \ell' = Loop \wedge tr' = nil \wedge co' = (tips(tr) - 1) \wedge st' = ()]$
$\vee [\delta' = 0 \wedge \ell' = Finish \wedge co' = tips(tr)])]$

Dans la preuve du théorème [illegible] de la preuve du théorème 0 pour un arbre tr, nous supposons qu'elle est vraie pour des arbres tr' tels que $|tr'| < |tr|$. Donc la preuve est par induction sur le bon-ordre $\langle (\{1\} \cup (\{0\} \times \omega)), \prec \rangle$ avec $\prec$ défini par

$\langle 0,n\rangle \prec 1$ pour tout $n \in \omega$ et $\langle 0,n'\rangle \prec \langle 0,n\rangle$ si et seulement si $n'<n$.

Burstall [74] parle d'"induction on data" et ne considère pas explicitement un ordre sur les théorèmes parce qu'il compte sur la culture mathématique de ses lecteurs pour éviter les erreurs comme les preuves circulaires.

Le principe d'induction $(\mathcal{B}_\tau)$ garantit que l'induction est faite correctement puisque le théorème $\theta_{\lambda'}$ ne peut être utilisé dans la preuve du théorème θ_λ que s'il a été démontré avant θ_λ (i.e. $\lambda'<\lambda$). Tous les cas particuliers tels les preuves récursives ou mutuellement récursives de théorèmes dans la méthode de Burstall peuvent être pris en compte par un choix adéquat du bon-ordre $(\Lambda,<)$. A un isomorphisme près, nous pouvons toujours choisir $\Lambda \in \underline{Ord}$.

Par exemple, au lieu du bon-ordre $\langle W,<\rangle$ où $W=(\{1\}\cup(\{0\}\times\omega))$, nous pouvons utiliser $(\Lambda,<)$ où $\Lambda=\omega+1$ à l'isomorphisme près défini par $e(1)=\omega$ et $e(\langle 0,m\rangle)=m$.

Le lecteur peut maintenant vérifier que la preuve de Burstall [74] consiste exactement à appliquer le principe d'induction $(\mathcal{B}_\tau)$ avec :

$\Lambda = \omega+1$

$\theta_\lambda(\langle \ell, tr, co, \Delta t\rangle, \langle \ell', tr', co', \Delta t'\rangle) =$

$\quad ([(\lambda=\omega \wedge \ell=Start) \Rightarrow (\ell'=Finish \wedge co'=tips(tr))]$

$\quad \wedge [(|tr|=\lambda<\omega \wedge \ell=Loop) \Rightarrow (\ell'=Loop \wedge tr'=mil \wedge co'=(co+tips(tr)-1) \wedge \Delta t'=\Delta t)])$

$\pi = \omega$

$\Delta = S$

$I_\lambda(\delta', \langle \ell, tr, co, \Delta t\rangle, \langle \ell', tr', co', \Delta t'\rangle) =$

$\quad ([(\lambda=\omega) \Rightarrow P_{tr}(\delta', \langle \ell, tr, co, \Delta t\rangle, \langle \ell', tr', co', \Delta t'\rangle)]$

$\quad \wedge [(|tr|=\lambda<\omega) \Rightarrow P_{-1}(\langle \delta', \langle \ell, tr, co, \Delta t\rangle, \langle \ell', tr', co', \Delta t'\rangle)]$

Pour exemple à tout $\lambda \in \Lambda$

$I_{|tr|}(\delta, \langle \ell, tr, co, \Delta t\rangle, \langle \ell', tr', co', \Delta t'\rangle)$

$= [(\ell=Loop) \Rightarrow (\ell'=Loop \wedge tr'=mil \wedge co'=(co+tips(tr))-1 \wedge \Delta t'=(tr.\Delta t))]$

nous vérifions que $|tr'|=|tf(tr)|<|tr|$ et en utilisant le théorème :

$e_{|tr|}(\langle \delta', tr', co', \Delta t'\rangle, \langle \ell'', tr'', co'', \Delta t''\rangle)$

$= [(\delta'=\bot) \Rightarrow (\ell''=\bot \wedge tr''=\varepsilon \wedge co''=(co'+tips(tr')-1) \wedge \Delta t''=\Delta t')]$

nous dérivons

$[(\ell=Loop) \Rightarrow (\ell''=Loop \wedge tr''=mil \wedge co''=(co+tips(tf(tr))-1 \wedge \Delta t''=(tr.\Delta t))]$

$= I_{|tr|}(\delta, \langle \ell, tr, co, \Delta t\rangle, \langle \ell'', tr'', co'', \Delta t''\rangle)$

De même, à partir de :

$I_{|tr|}(\delta, \langle \ell, tr, co, \Delta t\rangle, \langle \ell', tr', co', \Delta t'\rangle)$

$= [(\ell=Loop) \Rightarrow (\ell'=Loop \wedge tr'=mil \wedge co'=(co+tips(tf(tr))-1) \wedge \Delta t'=(tr.\Delta t))]$

et

$t_{\underline{\delta}}(\langle Loop, mil, co, (tr.\Delta t)\rangle, \langle Loop, rg(tr), co'+1, \Delta t'\rangle)$

nous dérivons :

$[(\ell=Loop) \Rightarrow (\ell''=Loop \wedge tr''=rg(tr) \wedge co''=(co+tips(tf(tr)) \wedge \Delta t''=\Delta t)]$

$= I_{|tr|}(1, \langle \ell, tr, co, \Delta t\rangle, \langle \ell', tr', co', \Delta t'\rangle)$

La seule différence est que le programme ci-dérivé est total, de sorte qu'on n'a pas besoin de contrôler explicitement l'absence d'erreurs à l'exécution et plus généralement d'états de blocages : $(I_\lambda(\delta', \Lambda, \Lambda') \Rightarrow \exists \Delta' \in S : t_{\underline{\delta}}(\Delta', \Delta''))$.

□

5.3.4 COMPLETUDE SEMANTIQUE FORTE

L'argument de complétude sémantique donné dans le théorème 5.3.2∨3 est très faible parce qu'il consiste essentiellement à dire que $(\mathcal{B}_2)$ peut toujours être utilisé pour formuler les preuves "à la Floyd" (comme suggéré par Manna-Waldinger [78]). Ayant étendu la méthode de Burstall de sorte qu'elle intègre la méthode de Floyd (cf. 5.3.3∨4 et $(\mathcal{B}_6),\dots,(\mathcal{B}_{11})$), l'argument habituel de complétude sémantique pour la méthode de Floyd peut être transcrit pour la méthode de Burstall (par exemple, $(\Psi$ est fatale pour $\langle s,A,\Sigma\langle s,A,t,\Phi\rangle\rangle) \Rightarrow ((\mathcal{B}_{11})$ avec (LI) supprimée (et $\delta(\delta)=0$)), cf. 5.2.6.2-1). Cependant, de tels arguments de complétude ne sont pas dans l'esprit de Burstall [74] qui encourage à décomposer les preuves de propositions en lemmes, contrairement à Floyd [67] qui fait la preuve d'une seule proposition (décomposée en correction partielle, absence d'états de blocage et terminaison, une décomposition qui peut également s'appliquer à tout lemme mis en jeu dans la méthode de Burstall).

Nous démontrons maintenant un résultat plus fort de complétude sémantique, montrant que les lemmes mis en jeu dans des preuves "à la Burstall" peuvent être choisis plus librement.

Nous devons d'abord introduire un nouveau principe d'induction $(\mathcal{B}_{12})$ qui [illegible] entre évaluation symbolique (ES) et induction sur les données (LI) est [illegible]. En particulier les lemmes qui sont utilisés dans (LI) doivent être simples. Pour cela nous considérons une version de $(\mathcal{B}_2)$ où nous introduisons une variable de choix $\gamma_\lambda(A,A',A')$, de sorte que l'existence de $I_\lambda(\delta',A,A')$ peut être réduite en particulier à l'existence de $\gamma_\lambda(A,A',A')$ est vraie. (Notez que [illegible] dépendre γ de δ' ne serait utile que pour imposer des lemmes d'identité

Pour simplifier, (ES) sera traitée dans le style de $(\mathcal{B}_8)$, comme un cas particulier de (LI) et donc la relation de transition $\exists a\in A.\, t_a$ sera considérée comme un lemme particulier, disons θ_0, donné comme axiome.

De plus, comme nous l'avons noté pour $(\mathcal{B}_9)$, la condition de vérification $[\exists\delta\in\Delta.\, I_\lambda(\delta,A,A)]$ de $(\mathcal{B}_5)$ ou $(\mathcal{B}_8)$ implique que le lemme θ_λ est fatal pour $\langle s,A,\Sigma\langle s,A,t,\Phi\rangle\rangle$ mais nous en avons besoin que pour les états particuliers A pour lesquels θ_λ est utilisé, c'est-à-dire lorsque $\exists\lambda'<\lambda, A'\in S.\, \gamma_{\lambda'}(A',A,\lambda)$.

Finalement, puisque tous les lemmes jouissent des mêmes propriétés de fatalité, nous n'avons pas vraiment besoin de distinguer une proposition principale particulière.

Ces remarques nous conduisent au principe d'induction suivant (où $\Lambda\in\mathbb{O}_{wd}$, $\theta\in(\Lambda\to(S^2\to\{\#,\#\#\}))$, $\forall\lambda,A'\in S.\,(\theta_\lambda(A,A')=\exists a\in A.\, t_a(A,A'))$, $\gamma\in((\Lambda\vee 0)\to(S\times S\times\Lambda\to\{\#,\#\#\}))$:

$$
\begin{aligned}
&[\exists\Delta\in\mathbb{O}_{wd}.\ I\in((\Lambda\vee 0)\to(\Delta\times S\times S\to\{\#,\#\#\}))\\
&\quad(\forall\lambda\in(\Lambda\vee 0),\, A\in S.\,[(\exists\lambda'\in(\Lambda\vee 0),\, A'\in S.\ \gamma_{\lambda'}(A',A,\lambda))\Rightarrow\exists\delta\in\Delta.\, I_\lambda(\delta,A,A)])\\
&\quad\wedge(\forall\lambda\in(\Lambda\vee 0),\, A,A'\in S,\ \delta'\in\Delta.\\
&\qquad[I_\lambda(\delta',A,A')\Rightarrow\\
&\qquad\quad([\exists\lambda'<\lambda.\,\gamma_\lambda(A,A',A')]\\
&\qquad\quad\wedge[\forall\lambda'<\lambda.(\gamma_\lambda(A,A',A')\Rightarrow\\
&\qquad\qquad([\lambda'=0\Rightarrow\exists a'\in S.\,\theta_{\lambda'}(A,A')]\wedge\\
&\qquad\qquad\quad[\forall a''\in S.(\theta_{\lambda'}(A,a'')\Rightarrow\exists\delta''.\, I_{\lambda'}(\delta'',A,a''))])])]\\
&\qquad\vee\theta_\lambda(A,A')])]
\end{aligned}
\tag{$\mathcal{B}_{12}$}
$$

Nous montrons d'abord que $(\mathcal{B}_{12})$ est une autre formulation des principes d'induction généralisant la méthode de Burstall.

Théorème 5.3.4 v 1 (Equivalence des principes d'induction)

$$(\mathcal{B}_6) \Rightarrow [\exists \Lambda, \theta, \prec . (\forall A, A' \in S . (\theta_0(A,A') = \exists a \in A . t_a(A,A'))) \wedge (\mathcal{B}_{12})]$$

Démonstration

Choisir $\Lambda_{12} = (1 + \Lambda_6)$, $\Delta_{12} = \Delta_6$, $\forall A, A' \in S . (\theta_{12_0}(A,A') = \exists a \in A . t_a(A,A'))$, si $\lambda, \lambda' \in \Lambda_6$ alors $\theta_{12_{1+\lambda}} = \theta_{6_\lambda}$, $I_{12_{1+\lambda}}(\delta', A, A') = [I_{6_\lambda}(\delta', A, A') \wedge \forall \delta \in \Delta_6 . (I_{6_\lambda}(\delta, A, A') \Rightarrow \delta' \leq \delta)]$, $\prec_{12_{1+\lambda}}(A, A', 0) = [\exists \delta' \in \Delta_6 . (I_{12_{1+\lambda}}(\delta', A, A') \wedge \neg \theta_{6_\lambda}(A, A') \wedge \exists A'' \in S, a \in A . t_a(A', A'') \wedge \forall A'' \in S, a \in A . (t_a(A', A'') \Rightarrow [\exists \delta'' < \delta' . I_{6_\lambda}(\delta'', A, A'')]))]$ et $\prec_{12_{1+\lambda}}(A, A', 1+\lambda') = [\exists \delta' \in \Delta_6 . (I_{12_{1+\lambda}}(\delta', A, A') \wedge \neg \theta_{6_\lambda}(A, A') \wedge \forall A'' \in S . (\theta_{6_\lambda}(A', A'') \Rightarrow [\exists \delta'' < \delta' . I_{6_\lambda}(\delta'', A, A'')]))]$.

□

Théorème 5.3.4 v 2 (Equivalence des principes d'induction (suite))

$$[\exists \Lambda, \theta, \prec, \pi \in (\Lambda \to 0) . (\forall A, A' \in S . (\prec_\pi(A, A, \pi) = \mathbb{H} \wedge \theta_\pi(A,A') = [\Phi(A) \Rightarrow \Psi(A,A')] \wedge \theta_0(A,A') = \exists a \in A . t_a(A,A') \wedge (\mathcal{B}_{12}))] \Rightarrow (\mathcal{B}_9)$$

Démonstration

Choisir $\Lambda_9 = \Lambda_{12_\pi}$, $\theta_{9_\lambda}(A,A') = (A=0 \to \mathbb{H} \mid \theta_{12_\lambda}(A,A'))$, $\prec_9 = \prec_{12_\pi}$, $\Delta_9 = \Delta_{12_\pi}$, $I_{9_\lambda}(\delta, A, A') = (A=0 \to \mathbb{H} \mid I_{12_\lambda}(\delta, A, A'))$

□

Puisque nous ne distinguerons plus entre programme et prédicat θ_π comme dans $(\mathcal{B}_9)$, la correction de $(\mathcal{B}_{12})$ est mieux formalisée comme suit :

Condition 5.3.4 v 1 (Définition de la correction (relativement à t))

$$\forall \lambda \in (\Lambda \vee 0), p \in \Sigma(S, A, t, \xi_\lambda) . \exists \iota \in |p| . \theta_\lambda(p_0, p_\iota) \text{ où } \xi_\lambda(A) = [\exists \lambda' \in \Lambda, A' \in S . \prec_\lambda(A', A, \lambda)]$$

Théorème 5.3.4 v 3 (Correction (relativement à t))

$$(\mathcal{B}_{12}) \Rightarrow (5.3.4 : 1)$$

Démonstration

Découle des théorèmes 5.3.4 v 5 et 5.3.4 v 6 que nous démontrerons plus tard.

□

Nous nous intéressons principalement à la complétude de $(\mathcal{B}_{12})$. La réciproque du théorème 5.3.4 v 3 n'est pas vraie :

Théorème 5.3.4 v 4 (Condition insuffisante de complétude)

$$(5.3.4 : 1) \not\Rightarrow (\mathcal{B}_{12})$$

Démonstration

Considérer le contre-exemple : $S = \{a, b, c\}$, $A = \{a\}$, $t_a(A,A') = [(A = a \wedge A' = b) \vee (A = b \wedge A = c)]$, $\Lambda = 2$, $\theta_\lambda = t_a$, $\theta_1(A,A') = [A = a \wedge A' = c]$, $\theta_2(A,A') = [A = a \wedge A' = b]$, $\Delta \neq 0$, $\prec_\lambda(A, A', \lambda') = [(\lambda = 1 \wedge A = a \wedge A' \in \{a, b\} \wedge \lambda' = 0) \vee (\lambda = 2 \wedge A' = A = a \wedge \lambda' \in \{1, 2\})]$.

La condition (5.3.4 : 1) est vérifiée trivialement. Si $(\mathcal{B}_{12})$ était vrai alors nous aurions $\exists \delta_1 \in \Delta . \prec_1(a, a, \delta_1)$ donc $\exists \delta_1 \in \Delta . I_2(\delta_1, a, a)$ et par $\prec_2(a, a, \delta_1)$ et $\Delta = 2$, nous aurions $\exists \delta_2 (\delta_2 < \delta_1 \wedge I_2(\delta_2, a, c))$. Mais $\neg \theta_2(a, c)$ et $\forall \lambda' < 2 . \neg \prec_2(a, c, \lambda')$, une contradiction.

□

Dans la méthode de Burstall, l'utilisation d'un lemme θ_ℓ dans la preuve de la proposition θ_p a l'effet de couvrir un certain nombre de transitions en un seul pas θ_ℓ. Aussi, θ_ℓ peut être utilisé dans la preuve de θ_p seulement si cette réduction conserve la fatalité de θ_p. Autrement dit, θ_p doit être fatal pour les "transitions" θ_ℓ résultant des lemmes utilisés dans la preuve de θ_p. Ceci s'exprime plus formellement par la condition (où $\forall \Lambda,\Lambda' \in S.(\theta_\alpha(\Lambda,\Lambda') \bullet \exists \alpha \in R.\, t_\alpha(\Lambda,\Lambda'))$) :

Condition 5.3.4:2

$$\forall \lambda \in (\Lambda \cup 0),\ \Lambda \in S.\ [(\exists \lambda' \in \Lambda, \Lambda' \in S.\ \zeta_{\lambda'}(\Lambda',\Lambda,\lambda)) \Rightarrow (\forall p \in \Sigma^{<S,\{R\},\tau_{\lambda\Lambda},\varepsilon_\Lambda>}.\ \exists i \in |p|.\ \theta_\lambda(p_0,p_i))]$$

où $\quad \varepsilon_\Lambda(\underline{\Lambda}) = [\underline{\Lambda} = \Lambda]$

$$\tau_{\lambda\Lambda}{}_{\underline{\Lambda}}(\Lambda',\Lambda'') = [\exists \lambda' < \lambda.\ (\zeta_\lambda(\Lambda,\Lambda',\lambda') \wedge \theta_{\lambda'}(\Lambda',\Lambda''))]$$

La condition (5.3.4:2) est une condition nécessaire de complétude :

Théorème 5.3.4∾5 (Correction (relativement à τ), Condition nécessaire de complétude)

$$(\mathcal{B}_{12}) \Longrightarrow (5.3.4:2)$$

Démonstration

Supposons $(\mathcal{B}_{12})$. Si $\lambda = 1$ ou $\forall \lambda', \Lambda'.\ \neg \zeta_\lambda(\Lambda',\Lambda,\lambda)$ alors (5.3.4:2) est vrai trivialement sinon nous démontrons (5.3.4:2) par induction transfinie sur $\lambda \in (\lambda \cup 0)$. Etant donnés $\lambda \in \lambda$ et $\Lambda \in S$, supposons par l'absurde que $\exists \gamma \in \Lambda,\ \Lambda \in S.$ $p \in \Sigma^{<S,\{R\},\tau_{\lambda\Lambda},\varepsilon_\Lambda>}.\ (\zeta_{\lambda'}(\Lambda',\Lambda,\alpha) \wedge \forall i \in |p|.\ \neg\theta_\lambda(p_0,p_i))$. Pour avoir une contradiction, nous construisons par récurrence sur $i \in \omega$ une suite $\langle \Lambda_i : i \in \omega \rangle$ d'éléments de S et une suite $\langle \lambda_i : i \in \omega \rangle$ d'éléments de λ telles que $\Lambda_0 = \Lambda$, $\lambda_0 = \lambda$, et pour tout $i \in \omega$, $\Lambda_i \in S$, $\lambda_i < \lambda$, et $\zeta_{\lambda_i}(\Lambda_{i+1},\Lambda_i,\lambda_{i+1})$. Nous prenons λ_{i+1} de sorte que $\lambda_{i+1} < \lambda_i$ si $\lambda_i > 0$ et $\lambda_{i+1} = 0$ si $\lambda_i = 0$. Nous prenons Λ_{i+1} de sorte que $\zeta_{\lambda_i}(\Lambda_{i+1},\Lambda_i,\lambda_{i+1})$ soit vrai si $\lambda_i > 0$ et $\tau_{\lambda_i\Lambda_i}(\Lambda_i,\Lambda_{i+1})$ soit vrai si $\lambda_i = 0$. Nous prenons λ_{i+1} de sorte que $\zeta_{\lambda_i}(\Lambda_{i+1},\Lambda_i,\lambda_{i+1})$ soit vrai si $\lambda_i > 0$. Nous prenons Λ_{i+1} de sorte que $\tau_{\lambda_i\Lambda_i}(\Lambda_i,\Lambda_{i+1})$ soit vrai si $\lambda_i = 0$.

Nous prenons $\lambda_{i+1}(\Lambda,\Lambda,\alpha)$ de sorte que par $(\mathcal{B}_{12})$ nous dérivons $\Sigma_\alpha(\delta_1,\Lambda,\Lambda)$

alors d'après $(\mathcal{B}_{12})$, $\Sigma_\lambda(\delta_R,\Lambda,p_{i_R})$ implique $([\exists \underline{\lambda} < \lambda.\ \zeta_\lambda(\Lambda,p_{i_R},\underline{\lambda})] \wedge [\forall \lambda' < \lambda.\ (\zeta_\lambda(\Lambda,p_{i_R},\Lambda') \Rightarrow ([\lambda' = 0 \Rightarrow \exists \Lambda'' \in S.\ \theta_{\lambda'}(p_{i_R},\Lambda'')] \wedge [\forall \Lambda'' \in S.\ (\theta_{\lambda'}(p_{i_R},\Lambda'') \Rightarrow \exists \delta' < \delta_R.\ \Sigma_\lambda(\delta',\Lambda,\Lambda''))]))])$ parce que nous avons supposé $\neg\theta_\lambda(p_0,p_{i_R})$ et $p_0 = \Lambda$. Si $\underline{\lambda} = 0$ alors $\exists \Lambda'' \in S.\ \theta_{\underline{\lambda}}(p_{i_R},\Lambda'')$ donc $\exists \Lambda'' \in S.\ \tau_{\lambda\Lambda}(p_{i_R},\Lambda'')$. Sinon $0 < \underline{\lambda} < \lambda$ de sorte que par hypothèse d'induction $\forall \Lambda' \in S.\ [(\exists \lambda'', \Lambda''.\ \zeta_{\lambda''}(\Lambda'',\Lambda',\underline{\lambda})) \Rightarrow (\forall p' \in \Sigma^{<S,A,\tau_{\underline{\lambda}\Lambda'},\varepsilon_{\Lambda'}>}.\ \exists i \in |p'|.\ \theta_{\underline{\lambda}}(p'_0,p'_i))]$. En particulier pour $\Lambda' = p_{i_R}$, $\zeta_\lambda(\Lambda,p_{i_R},\underline{\lambda})$ est vrai et $\Sigma^{<S,A,\tau_{\underline{\lambda}p_{i_R}},\varepsilon_{p_{i_R}}>}$ n'est pas vide, d'où $\exists \Lambda'' \in S.\ \theta_{\underline{\lambda}}(p_{i_R},\Lambda'')$ donc $\exists \Lambda'' \in S.\ \tau_{\lambda\Lambda}(p_{i_R},\Lambda'')$. Puisque p_{i_R} n'est pas un état de blocage $i_{R+1} = i_R + 1$ appartient à $|p|$ et nous avons $\tau_{\lambda\Lambda}(p_{i_R},p_{i_{R+1}})$. Il s'ensuit que $\exists \lambda' < \lambda.\ (\zeta_\lambda(\Lambda,p_{i_R},\lambda') \wedge \theta_{\lambda'}(p_{i_R},p_{i_{R+1}}))$ d'où $\exists \delta_{R+1} < \delta_R.\ \Sigma_\lambda(\delta_{R+1},\Lambda,p_{i_{R+1}})$.

□

La condition (5.3.4:2) (i.e. chaque lemme est fatal relativement aux lemmes utilisés pour sa preuve) implique la condition (5.3.4:1) (i.e. chaque lemme est fatal relativement au système de transition) :

Théorème 5.3.4∾6 (La fatalité relativement à τ implique la fatalité relativement à t)

$$(5.3.4:2) \Longrightarrow (5.3.4:1)$$

Démonstration

Supposons (5.3.4:2), nous démontrons (5.3.4:1) par induction transfinie sur $\lambda \in (\lambda \cup 0)$. Supposons par l'absurde que $\exists p \in \Sigma^{<S,A,t,\varepsilon_\lambda>}.\ \forall i \in |p|.\ \neg\theta_\lambda(p_0,p_i)$. Pour avoir une contradiction, nous allons construire une séquence $\langle i_R : R > 0 \rangle$ telle que $p_{i_0} \xrightarrow{a} p_{i_1} \xrightarrow{a} \cdots$ soit un contre-exemple pour (5.3.4:2) c'est-à-dire $\exists \lambda'.\ \zeta_{\lambda'}(\Lambda',p_{i_R},\lambda) \wedge \forall R > 0.\ [\tau_{\lambda p_{i_0}}(p_{i_0},p_{i_{R+1}}) \wedge \neg\theta_\lambda(p_{i_0},p_{i_R})]$. Posons $i_0 = 0$. Si la séquence est construite jusqu'au i_R, alors elle peut être prolongée car $\exists i_{R+1} > i_R.\ \tau_{\lambda p_{i_0}}(p_{i_R},p_{i_{R+1}})$ (et $\neg\theta_\lambda(p_{i_0},p_{i_R})$ par hypothèse et $i_0 = 0$). Autrement $\forall j > i_R.\ \neg\tau_{\lambda p_{i_0}}(p_{i_R},p_j)$ de sorte que par définition de $\tau_{\lambda p_{i_0}}$ il viendrait $\forall j > i_R.\ \forall \lambda' < \lambda.\ [\neg\zeta_\lambda(p_0,p_{i_R},\lambda') \vee \neg\theta_{\lambda'}(p_{i_R},p_j)]$. Si $\forall \lambda' < \lambda.\ \neg\zeta_\lambda(p_0,p_{i_R},\lambda')$ alors

$\forall s' \in S. \neg \tau_{\lambda p_{i_0}}(p_{i_R}, s')$ de sorte que $p_{i_0} \xrightarrow{a} p_{i_1} \xrightarrow{a} \cdots \xrightarrow{a} p_{i_R} \in \Sigma^{<S, A, \tau_{\lambda p_{i_0}}, \varepsilon_{p_{i_0}}>}$, en contradiction avec (5.3.4 : 2). Soit donc $\exists s' \in S. \tau_\lambda(p_0, p_{i_R}, s')$ de sorte que pour un $\lambda' < \lambda$ nous dérivons $\forall j \geq i_R. \neg \theta_{\lambda'}(p_{i_R}, p_j)$ donc $p_{i_R} \xrightarrow{a} p_{i_{R+1}} \cdots \in \Sigma^{<S, A, \tau, \varepsilon_{\lambda'}>}$, en contradiction avec l'hypothèse d'induction (5.3.4 : 1).

□

Nous pouvons maintenant donner une condition nécessaire et suffisante de complétude sémantique pour $(\mathcal{B}_{1,2})$.

Théorème 5.3.4 ν 7 (Condition nécessaire et suffisante de complétude sémantique forte)

$(5.3.4 : 2) \iff (\mathcal{B}_{1,2})$

Démonstration

Par suite de 5.3.4 ν 6, nous devons démontrer seulement que $(5.3.4 : 2) \Rightarrow (\mathcal{B}_{1,2})$. Etant donnés $\lambda \in (\Lambda \nu 0)$, $s \in S$, nous définissons :

$$\underline{\underline{Im}}_{\lambda s} = \cup\{\underline{\underline{Inter}}^{<S, \{q\}, \tau_{\lambda s}, \#, \#, \theta_\lambda>}(s) : \exists \lambda' \in \Lambda, s' \in S. \gamma_{\lambda'}(s', s, \lambda)\}$$

$$\underline{\underline{Go}}_{\lambda s} = \cup\{\underline{\underline{Goal}}^{<S, \{q\}, \tau_{\lambda s}, \#, \#, \theta_\lambda>}(s) : \exists \lambda' \in \Lambda, s' \in S. \gamma_{\lambda'}(s', s, \lambda)\}$$

$$\underline{\underline{Ac}}_{\lambda s} = \underline{\underline{Im}}_{\lambda s} \cup \underline{\underline{Go}}_{\lambda s}$$

Nous démontrons d'abord que $(5.3.4 : 2) \Rightarrow (\forall \lambda \in (\Lambda \nu 0), s \in S. \underline{\underline{wf}}(\underline{\underline{Ac}}_{\lambda s}, \tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}^{-1}))$.

Ceci est évident lorsque $\forall \lambda' \in \Lambda, s' \in S. \neg \tau_{\lambda'}(s', s, \lambda)$ puisque $\underline{\underline{Ac}}_{\lambda s}$ est vide. Sinon, étant donnés $\lambda \in (\Lambda \nu 0)$ et $s \in S$ tels que $\exists \lambda' \in \Lambda, s' \in S. \gamma_{\lambda'}(s', s, \lambda)$, raisonnons par l'absurde que $\exists p \in (\omega \to \underline{\underline{Ac}}_{\lambda s}). \forall i \in \omega. \tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}(p_i, p_{i+1})$. Nous avons $\forall i \in \omega.$ $(p_i \in \underline{\underline{Im}}_{\lambda s})$, de sorte que $\neg \theta_\lambda(\lambda, p_i)$ donc $p_i \in \underline{\underline{Go}}_{\lambda s}$. Puisque $p_0 \in \underline{\underline{Im}}_{\lambda s}$ nous pouvons subdiviser $p_0 s$ (puisque nous sommes ramifiés à gauche de t par définition) en $p_0 \cdots p_{i_R}$ d'une certaine trace $\nu \in \Sigma^{<S, A, \tau, \varepsilon_\lambda>}$ telle que $\forall j < i_R. \neg \theta_\lambda(p_0, p_j)$ et $p_{i_R} = p_\lambda$ de sorte que $\forall i < i_R. \tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}(p_i, p_{i+1})$ donc $\nu \in \Sigma^{<S, \{q\}, \tau_{\lambda s}, \varepsilon>}$ et $\forall i \in |p|. (p_i \in \underline{\underline{Go}}_{\lambda s})$ donc $\neg \theta_\lambda(p_0, p_i)$, en contradiction avec $(5.3.4 : 1)$. □

Supposant $(5.3.4 : 2)$, d'après le lemme ci-dessus, nous pouvons définir :

$$\Delta = \sup^+\{\underline{\underline{rk}}(\underline{\underline{Ac}}_{\lambda s}, \tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}^{-1}) : \lambda \in (\Lambda \nu 0) \wedge s \in S\}$$

$$\Xi_\lambda(\delta', s, s') = [s \in \underline{\underline{Ac}}_{\lambda s} \wedge \delta' = \underline{\underline{rk}}(\underline{\underline{Ac}}_{\lambda s}, \tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}^{-1})(s')]$$

Si $\lambda \in (\Lambda \nu 0)$, $s \in S$ et $\exists \lambda' \in (\Lambda \nu 0), s' \in S. \gamma_{\lambda'}(s', s, \lambda)$ alors $s \in \underline{\underline{Ac}}_{\lambda s}$ de sorte que $\Xi_\lambda(\delta, s, s)$ est vrai avec $\delta = \underline{\underline{rk}}(\underline{\underline{Ac}}_{\lambda s}, \tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}^{-1})(s)$.

Supposons $\lambda \in (\Lambda \nu 0)$, $s, s' \in S$, $\delta' \in \Delta$ et $\Xi_\lambda(\delta', s, s')$. Nous avons $s' \in \underline{\underline{Ac}}_{\lambda s}$. Si $s' \in \underline{\underline{Go}}_{\lambda s}$ alors $\theta_\lambda(s, s')$ est vrai. Sinon $s' \in \underline{\underline{Im}}_{\lambda s}$ d'où, d'après $(5.3.4 : 2)$, il existe $s'' \in S$ tel que $\tau_{\lambda s}(s', s'')$ donc un certain $\lambda' < \lambda$ tel que $\gamma_\lambda(s, s', \lambda') \wedge \theta_{\lambda'}(s', s'')$. Si $\lambda' = 0$ nous concluons que $\exists s'' \in S, a \in A. t_a(s', s'')$ par définition de θ_0. Soit $\lambda' \neq 0$ et si $\forall s'' \in S, a \in A. \neg t_a(s', s'')$ alors $\langle s' \rangle \in \Sigma^{<S, A, t, \varepsilon_{\lambda'}>}$ et donc d'après $(5.3.4 : 2)$ et le théorème 5.3.4 ν 6 nous concluons à partir de $\neg(5.3.4 : 1)$ que $\theta_{\lambda'}(s', s'')$ d'où $\tau_{\lambda s}(s', s'')$. Il s'ensuit d'après $s' \in \underline{\underline{Im}}_{\lambda s}$ que $\exists \underline{c}, \underline{s}, p \in \Sigma^{<S, \{q\}, \tau_{\lambda s}, \varepsilon_\lambda>}, i \in |p|.$ $(\gamma_{\underline{\lambda}}(\underline{s}, s, \lambda) \wedge \forall j \leq i. \neg \theta_\lambda(p_0, p_j) \wedge p_R = s')$ de sorte que la trace infinie $p_0, \ldots, p_R, s', \ldots$ est un contre-exemple pour $(5.3.4 : 2)$. Aussi par l'absurde, nous concluons $\exists s'' \in S, a \in A. t_a(s', s'')$. Finalement, étant donnés $\lambda' < \lambda$ et $s'' \in S$ tels que $\gamma_\lambda(s, s', \lambda')$ et $\theta_{\lambda'}(s', s'')$ nous avons $\tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}(s', s'')$ donc $s'' \in \underline{\underline{Ac}}_{\lambda s}$ et il existe $\delta'' = \underline{\underline{rk}}(\underline{\underline{Ac}}_{\lambda s}, \tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}^{-1})(s'') < \underline{\underline{rk}}(\underline{\underline{Ac}}_{\lambda s}, \tau_{\lambda s} \restriction \underline{\underline{Im}}_{\lambda s}^{-1})(s') = \delta'$ tel que $\Xi_\lambda(\delta'', s', s'')$ soit vrai.

□

5.3.5 COMPARAISON METHODOLOGIQUE DES METHODES DE FLOYD ($\mathcal{F}_6$) ET DE BURSTALL ($\mathcal{B}_7$) GENERALISEES

Comme nous l'avons vu dans les exemples 5.2.3-1 et 5.3.3-1, une différence majeure entre la méthode de Burstall [74] (et ses suivants comme Owicki-Lamport [82] ou Manna-Pnueli [82] qui présentent les preuves au moyen de formes limitées de chartes de preuves) et le principe d'induction ($\mathcal{B}_7$) est que Burstall insiste sur la présentation finie des preuves (ou de manière équivalente à considérer des chartes de preuves sans cycles et $\Delta \in \omega$ au lieu de $\Delta \in \omega_{cd}$ dans ($\mathcal{B}_7$)) tandis que ($\mathcal{B}_7$) n'exige qu'une relation bien-fondée. En particulier, ($\mathcal{B}_7$) peut être présenté au moyen de chartes de preuves comportant des cycles le long desquels une certaine entité (représentée par δ dans le principe d'induction ($\mathcal{B}_7$)) doit décroître strictement.

L'importance de cette remarque est d'attirer l'attention sur le fait que lorsqu'elle est convenablement généralisée (comme par ($\mathcal{B}_7$)), la méthode de Burstall contient la méthode de Floyd (plus précisément le principe d'induction ($\mathcal{F}_6$)) comme un cas particulier. Ceci parceque nous pouvons choisir $\Lambda = 1 = \{0\}$, $\theta_0 = \Psi$, $\pi = 0$ dans le principe d'induction ($\mathcal{B}_7$) de sorte que ($\mathcal{B}_{7.1}$) est toujours vrai et ($\mathcal{B}_{7.3.b}$) ne s'applique jamais, auquel cas ($\mathcal{B}_7$) se réduit exactement à ($\mathcal{F}_6$).

L'avantage net du principe d'induction ($\mathcal{B}_7$) sur ($\mathcal{F}_6$) est que ($\mathcal{B}_7$) introduit la possibilité, inexistante dans ($\mathcal{F}_6$), de preuves récursives. En particulier clarifie la remarque de Burstall [74] que des preuves récursives peuvent être retenues pour des versions itératives de programmes récursifs. Beaucoup plus important est le fait que ($\mathcal{B}_7$) permet, une preuve étant donnée, d'en obtenir [illegible] indépendants tandis que ($\mathcal{F}_6$) nécessite une preuve globale. [illegible] l'argument de "séparation des soucis" en faveur du principe d'induction ($\mathcal{B}_7$)

correction partielle, de terminaison, d'absence d'états de blocage. Comme le remarque Gries [79], tout est groupé dans la méthode de Burstall. Cependant ceci n'est qu'une question de présentation à laquelle nous pouvons aisément remédier puisque la décomposition en preuves indépendantes de correction partielle, de terminaison, d'absence d'état de blocage (et d'absence d'interférences dans le cas de programmes parallèles) s'appliquant à ($\mathcal{F}_6$) peut tout aussi s'appliquer à chaque lemme θ_λ, $\lambda \in \Lambda$ dans ($\mathcal{B}_7$).

5.4 EQUIVALENCE FORTE DES PRINCIPES D'INDUCTION $(\mathcal{F}_i)$ ET $(\mathcal{B}_j)$

Puisque, (en posant $e=\xi$ et $\xi_c=\#$), $(\mathcal{F}_6)\Leftrightarrow(\psi$ est fatale pour $\langle S,A,\Sigma\langle S,A,t,\varepsilon\rangle\rangle)\Leftrightarrow(\mathcal{B}_7)$, les principes d'induction $(\mathcal{F}_6)$ et $(\mathcal{B}_7)$ sont équivalents. Ceci signifie que lorsqu'une preuve par une méthode existe, une preuve par l'autre méthode doit exister. Cependant d'un point de vue pratique, utiliser $(\mathcal{F}_6)$ pour un programme qui a une preuve "naturelle" par $(\mathcal{B}_7)$ a été considéré quelquefois comme un défi (cf. Manna-Waldinger[78], Gries[79]). Nous démontrons maintenant un résultat d'équivalence plus fort qui montre que ce défi peut toujours être relevé parce qu'une preuve par $(\mathcal{B}_7)$ peut être réécrite systématiquement en une preuve par $(\mathcal{F}_6)$ et vice versa, (et d'après les résultats des paragraphes 5.2.3 et 5.3.3, ceci est vrai entre $(\mathcal{F}_i)$ et $(\mathcal{B}_j)$ quelconques). La transformation entre les deux preuves est très similaire à l'élimination de la récursivité dans les programmes et la présentation récursive de programmes itératifs. Ainsi, (comme pour les programmes), nous ne prétendons pas naturellement, que cette transformation préserve le "naturel" de preuves.

5.4.1 $(\mathcal{F}_6)\Rightarrow(\mathcal{B}_7)$

Comme nous l'avons fait observer au paragraphe 5.3.5, $(\mathcal{B}_7)$ satisfait $(\mathcal{F}_6)$ comme un cas particulier (en choisissant $\Lambda=\{\psi_0\}$, $\psi_0=\psi$, $\tau=\omega$). De ce fait, si une preuve par $(\mathcal{F}_6)$ est aussi (à des détails mineurs près) une preuve par $(\mathcal{B}_7)$.

Je vais maintenant [illegible] paragraphe [illegible] pour faire une comparaison équitable entre les méthodes de Burstall [illegible] comme nous l'avons également fait observer

au paragraphe 5.3.5, la méthode de Burstall correspond plus précisément au cas $\Delta\in\omega$ qu'au cas $\Delta\in\underline{Ord}$ dans $(\mathcal{B}_7)$.

Puisque Floyd[67] et Burstall[74] utilisent l'induction mathématique seulement sous la forme d'une induction ordinaire sur les nombres naturels, on pourrait nous demander d'ajouter les restrictions $\Gamma=\omega$ dans $(\mathcal{F}_6)$ et $\Lambda=\omega$ dans $(\mathcal{B}_7)$. Nous savons que lorsque le non-déterminisme est infini, $(\mathcal{F}_6)$ n'est pas sémantiquement complet avec $\Gamma=\omega$ et nous faisons la conjecture que $(\mathcal{B}_7)$ n'est pas sémantiquement complet avec $\Lambda=\omega$ (et $\Delta\in\omega$). Cependant, considérer que $\Gamma\in\underline{Ord}$ dans $(\mathcal{F}_6)$ et considérer que $\Lambda\in\underline{Ord}$ dans $(\mathcal{B}_7)$ sont des généralisations respectivement des méthodes de Floyd et Burstall de même nature de sorte que nous disons qu'une conséquence du théorème 5.4.1∨1 est qu'une preuve par la méthode de Floyd peut se réécrire en une preuve par la méthode de Burstall :

Théorème 5.4.1∨1

Soit $\langle S,A,t,\varepsilon\rangle$, si nous avons démontré que $\Gamma\in\underline{Ord}$, $J\in(\Gamma\times S\times S\to\{\#,\#\#\})$, $\xi=t$ satisfont $(\mathcal{F}_6.1)$ et $(\mathcal{F}_6.2)$ alors réécrivant la preuve, nous pouvons trouver $\Lambda\in\underline{Ord}$, $\theta\in(\Lambda\to(S\times S\to\{\#,\#\#\}))$, $\pi\in\Lambda$, $\Delta\in\omega$, $I\in(\Lambda\to(\Delta\times S\times S\to\{\#,\#\#\}))$ satisfaisant $(\mathcal{B}_7.1)$ et $(\mathcal{B}_7.2)$ où $\xi=\varepsilon$.

Démonstration

S étant un ensemble, il existe (d'après l'axiome du choix) un ordinal Σ et une bijection ξ de Σ dans S.

Le plus, le produit cartésien $\Gamma\times\Sigma$ est bien ordonné par l'ordre lexicographique droit $\prec$ défini par $\langle\delta,\sigma\rangle\prec\langle\delta',\sigma'\rangle$ si et seulement si $\sigma<\sigma'$ ou bien $\sigma=\sigma'$ et $\delta<\delta'$. Alors la fonction $F(\langle\delta,\sigma\rangle)$ [illegible]

isomorphisme entre $\langle \Gamma \times \Sigma^*, \prec \rangle$ et $\langle \Gamma \dot{\times} \Sigma^*, < \rangle$.

Il s'ensuit que $\ell \in ((S \times \Gamma) \to \pi)$ défini par $\pi = \Gamma \dot{\times} \Sigma^*$ et $\ell(\langle \Delta, \gamma \rangle) = F(\langle \gamma, f_\gamma^{-1}(\Delta) \rangle)$ est un isomorphisme. Ainsi nous pouvons définir $d \in (\pi \to S)$ et $r \in (\pi \to \Gamma)$ par $d(\lambda) = \Delta$ et $r(\lambda) = \gamma$ si et seulement si $\ell(\langle \Delta, \gamma \rangle) = \lambda$.

Choisir :

$\Lambda = \pi + 1 \quad (= \pi \cup \{\pi\})$

$\theta_\lambda(\Delta, \Delta') = ([\lambda = \pi \wedge \Psi(\Delta, \Delta')] \vee [\lambda < \pi \wedge (J(r(\lambda), d(\lambda), \Delta) \to \Psi(d(\lambda), \Delta'))])$

$\Delta = 3$

$I_\pi(2, \Delta, \Delta') = \mathit{ff}$

$I_\pi(1, \Delta, \Delta') = [\exists \gamma \in \Gamma.\ \bar{J}(\gamma, \Delta, \Delta') \wedge \Delta' = \bar{\Delta}]$

$I_\pi(0, \Delta, \Delta') = \Psi(\Delta, \Delta')$

Pour tous $\lambda < \pi$ (ou de manière équivalente $\lambda \in \pi$) :

$I_\lambda(2, \Delta, \Delta') = [J(r(\lambda), d(\lambda), \Delta) \Rightarrow (\Delta = \Delta')]$

$I_\lambda(1, \Delta, \Delta') = [J(r(\lambda), d(\lambda), \Delta) \Rightarrow (\exists a.\ t_a(\Delta, \Delta') \wedge \neg \Psi(d(\lambda), \Delta'))]$

$I_\lambda(0, \Delta, \Delta') = \theta_\lambda(\Delta, \Delta')$

□

5.4.2 $(\mathcal{B}_\tau) \Rightarrow (\mathcal{F}_\sigma)$

Une conséquence du théorème 5.4.2∨1 suivant est qu'une preuve par la méthode de Burstall peut se recécrire en une preuve par la méthode de Floyd. Sans surprise, la technique est analogue à la transformation d'un programme récursif en un programme itératif équivalent. En poursuivant cette comparaison jusqu'à la caricature, c'est comme si on remplaçait tous les théorèmes (et leurs preuves) d'un livre de mathématiques par une proposition (et sa preuve) !

Théorème 5.4.2∨1

Si nous avons démontré que $\Lambda \in \mathbb{O}_{rd}$, $\theta \in (\Lambda \to (S \times S \to \{\mathit{tt}, \mathit{ff}\}))$, $\pi \in \Lambda$, $\Delta \in \mathbb{O}_{rd}$, $I \in (\Lambda \to (\Delta \times S \times S \to \{\mathit{tt}, \mathit{ff}\}))$, $\phi = \mathit{tt}$ satisfont $(\mathcal{B}_\tau.1)$, $(\mathcal{B}_\tau.2)$ et $(\mathcal{B}_\tau.3)$ pour un système de transition $\langle S, A, t, \varepsilon \rangle$, alors recécrivant cette preuve, nous pouvons trouver $\Gamma \in \mathbb{O}_{rd}$, $J \in (\Gamma \times S \times S \to \{\mathit{tt}, \mathit{ff}\})$ satisfaisant $(\mathcal{F}_\sigma.1)$ et $(\mathcal{F}_\sigma.2)$ où $\phi = \mathit{tt}$.

Démonstration

(a) Nous définissons $\delta m \in (\Lambda \to (S \times S \to \Delta))$ telle que pour tout $\lambda \in \Lambda$ nous avons $dom(\delta m(\lambda)) = \{\langle \Delta, \Delta' \rangle \in S^2 :\ \exists \delta' \in \Delta.\ I_\lambda(\delta', \Delta, \Delta')\}$ et $\delta m(\lambda)(\Delta, \Delta') = \delta$ si et seulement si $[\delta \in \Delta \wedge I_\lambda(\delta, \Delta, \Delta') \wedge \forall \delta' \in \Delta.\ (I_\lambda(\delta', \Delta, \Delta') \Rightarrow [\delta \leq \delta'])]$, de sorte que $\delta m(\lambda)(\Delta, \Delta')$ est le plus petit δ tel que $I_\lambda(\delta, \Delta, \Delta')$ soit vrai.

(b) Soient $HS \in (\Lambda \to (S^3 \to \{\mathit{tt}, \mathit{ff}\}))$ et $LI \in (\Lambda \to (S \times S \times \Delta \times S \to \{\mathit{tt}, \mathit{ff}\}))$ tels que

$HS(\lambda)(\Delta, \Delta', \Delta'') = [I_\lambda(\delta m(\lambda)(\Delta, \Delta'), \Delta, \Delta') \wedge \exists a \in A.\ t_a(\Delta', \Delta'') \wedge$

$\forall \Delta''' \in S.\ (\exists a \in A.\ t_a(\Delta', \Delta''') \Rightarrow [\exists \delta''' < \delta m(\lambda)(\Delta, \Delta').\ I_\lambda(\delta''', \Delta, \Delta''')])]$

$LI(\lambda)(\Delta, \Delta', \Delta'', \Delta''') = [I_\lambda(\delta m(\lambda)(\Delta, \Delta'), \Delta, \Delta') \wedge t_a(\Delta', \Delta'') \wedge$

$\forall \Delta'''' \in S.\ (\theta_\lambda(\Delta', \Delta'''') \Rightarrow [\exists \delta'''' < \delta m(\lambda)(\Delta, \Delta').\ I_\lambda(\delta'''', \Delta, \Delta'''')])]$

Informellement, $HS(\lambda)(A,A',A'')$ signifie que dans la preuve du lemme θ_λ, nous pouvons montrer par évaluation symbolique que si l'exécution commence dans un état A et atteint plus tard l'état A' alors l'exécution d'un pas du programme peut conduire à A''. De manière similaire, $LI(\lambda)(A,A';\lambda',A'')$ signifie que dans la preuve du lemme θ_λ, nous pouvons montrer par induction sur les données que si l'exécution commence dans l'état A et atteint plus tard l'état A' alors d'après le lemme $\theta_{\lambda'}$, elle peut conduire à l'état A''.

(c) Nous définissons la relation $\succ$ sur $\Lambda\times S\times S$ telle que

$$\langle\lambda_1,A_1,A_2'\rangle \succ \langle\lambda_2,A_2,A_2'\rangle$$

si et seulement si

$$\begin{aligned}[& (\lambda_2=\lambda_1 \wedge A_2=A_1 \wedge HS(\lambda_2)(A_1,A_1',A_2')) \\ & \vee (\lambda_2=\lambda_1 \wedge A_2=A_1 \wedge \exists\lambda'<\lambda_1.\, LI(\lambda_1)(A_1,A_1',\lambda',A_2')) \\ & \vee (\lambda_2<\lambda_1)]\end{aligned}$$

(d) La relation $\succ$ sur $\Lambda\times S\times S$ est bien-fondée.

Par l'absurde, s'il existait une chaîne infinie telle que $\forall i\geq 0.(\langle\lambda_i,A_i,A_i'\rangle \succ \langle\lambda_{i+1},A_{i+1},A_{i+1}'\rangle)$ alors d'après (c) et (b) la chaîne $\langle\lambda_i,\delta m(\lambda_i)(A_i,A_i')\rangle$, $i\geq 0$ serait strictement décroissante pour l'ordre lexicographique gauche sur des paires d'ordinaux, une contradiction.

(e) Il s'ensuit à partir de (d) que nous pouvons définir $\sigma\in(\Lambda\to(S\times S\to \mathrm{Ord}))$ par induction transfinie de sorte que pour tout $\lambda\in\Lambda$ et $A,A'\in S$ nous ayons :

$$\sigma(\lambda)(A,A') = \sup\{\alpha+1 : \neg\theta_\lambda(A,A') \wedge ([\exists A''\in S.\,(HS(\lambda)(A,A',A'') \wedge \alpha=\sigma(\lambda)(A,A''))] \vee$$

$$[\exists\lambda'<\lambda.\,\exists A''\in S.\,(LI(\lambda)(A,A',\lambda',A'') \wedge \alpha=(\sigma(\lambda')(A,A'')+\sigma(\lambda)(A'',A'))]\}$$

Intuitivement, si l'exécution commence dans un état A et atteint un état A' [illegible] atteindra fatalement un certain état

Nous choisissons :

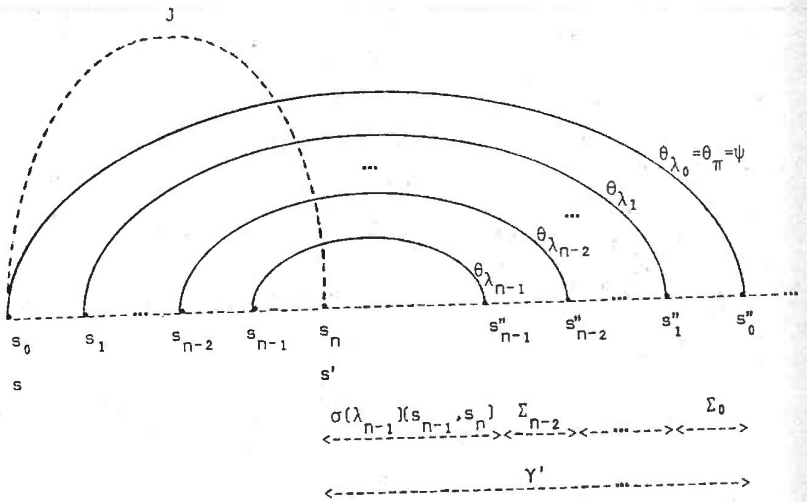
(f) $T(\lambda,A,A') = [\exists m\in(0\vee 0),\ \lambda\in(\pi\to\Lambda),\ A_0\in S,\ \ldots A_m\in S,\ \Sigma\in(m-1\to \mathrm{Ord})$.

$$\begin{aligned}(& (A_0=A)\wedge(\lambda_0=\pi)\wedge(A_m=A') \\ & \wedge \forall i\in(m\vee 0).\,\exists A_i''\in S.\, LI(\lambda_{i-1})(A_{i-1},A_i,\lambda_i,A_i'') \\ & \wedge I_{\lambda_{m-1}}(\delta m(\lambda_{m-1})(A_{m-1},A_m),A_{m-1},A_m) \\ & \wedge \forall i\in(m\vee 0).\,\neg\theta_{\lambda_i}(A_i,A_{i+1}) \\ & \wedge \forall i\in(m\vee 0).(\Sigma_{i-1}=\sup\{\sigma(\lambda_{i-1})(A_{i-1},A'') : LI(\lambda_{i-1})(A_{i-1},A_i,\lambda_i,A'')\}) \\ & \wedge \alpha'=(\Sigma_0+\ldots+\Sigma_{m-2}+\sigma(\lambda_{m-1})(A_{m-1},A_m)))]\end{aligned}$$

(g) $\Gamma = \sup^+\{\alpha'\in\mathrm{Ord} : \exists A,A'\in S.\,T(\alpha',A,A')\}$

$T(\alpha',A,A')$ est choisi de façon à exprimer que "si l'exécution commence dans l'état A et atteint plus tard l'état A' alors il atteindra fatalement "dans au plus α' pas" un état A'' satisfaisant $\psi(A,A'')$". Puisque nous considérons le non-déterminisme non-borné, la terminaison peut être faible. Donc la phrase "dans au plus α' pas" ne doit pas être prise littéralement lorsque nous devons recourir à des ordinaux transfinis $\alpha'>\omega$. Dans ce cas, nous pouvons démontrer la terminaison faible seulement c'est-à-dire trouver une classe bien fondée $\langle W,<\rangle$ et une fonction de terminaison $f\in(S\times S\to W)$ telle que $\exists a\in A.\,t_a(A',A'')\Rightarrow(f(A,A')<f(A,A''))$. Alors la phrase "dans au plus α' pas" est une abréviation pour la phrase plus rigoureuse "α' est le rang $\rho(f(A,A'))$ de $\ldots$". De plus nous choisissons $\langle\Gamma,<\rangle$ pour $\langle W,<\rangle$ et laissons f implicite. On aurait pu définir f comme étant le plus petit ordinal α' pour lequel $T(\alpha',A,A')$ est vrai).

Les chemins (?) [illegible] peuvent être caractérisés intuitivement au moyen du diagramme suivant (où seulement les chemins d'exécution commençant [illegible] ont été considérés) :



Dans la preuve de fatalité du lemme θ_{λ_i}, $i=0,\dots,n-2$, il a été montré que commençant dans l'état A_i, satisfaisant l'assertion intermittente $I_{\lambda_i}(\delta m(\lambda_i)(A_i,A_i''),A_i,A_i)$, l'exécution atteindra l'état A_{i+1} satisfaisant $I_{\lambda_i}(\delta m(\lambda_i)(A_i,A_{i+1}),A_i,A_{i+1})$. Appliquant $\theta_{\lambda_{i+1}}$ en ce point, il a été montré que l'exécution atteindra fatalement un certain état A''_{i+1} tel que $\theta_{\lambda_{i+1}}(A_{i+1},A''_{i+1})$ soit vrai et satisfaisant l'assertion intermittente $I_{\lambda_i}(\delta m(\lambda_i)(A_i,A''_{i+1}),A_i,A''_{i+1})$. Alors $LI(\lambda_i)(A_i,A_{i+1},\lambda_{i+1},A''_{i+1})$ est vrai. Puis, à partir de la dernière assertion intermittente, on a déduit que $I_{\lambda_i}(\delta m(\lambda_i)(A_i,A_i''),A_i,A_i'')$ qui implique $\theta_{\lambda_i}(A_i,A_i'')$ et termine la preuve. De plus l'exécution conduit de A''_{i+1} à A''_i en "au plus $\sigma(\lambda_i)(A_i,A''_{i+1})$ pas" donc "en au plus Σ_i pas" suivant on considère que l'état A''_{i+1} [illegible]

[illegible] suivant découle de la preuve du lemme $\theta_{\lambda_{n-1}}$. Il a été montré que commençant dans l'état A_{n-1}, l'exécution atteindra l'état $s_n=A$ [illegible] atteindra l'état s_{n-1} en au plus $\sigma(\lambda_{n-1})(s_{n-1},s_n)$ pas [illegible]

Après "élimination de la récursivité", si l'exécution commence en $s_0=A$ et

(3) Preuve de $(\mathcal{P}_6.2)$.

Il suffit $(\mathcal{P}_6.1)$ $\forall s\in S.\ \exists\Sigma\in\Delta.\ \Gamma_\pi(\delta,s,\Sigma)$ de sorte que d'après (a) nous avons $\forall s\in S.\ \Gamma_\pi(\delta m(\pi)(A,A'),s,\Sigma)$ qui implique $\forall s\in S.\ \exists\sigma(\pi)(A,A'),s,\Sigma)$ en choisissant $n=1$, $\lambda_0=\pi$, $A_0=A_1=A$.

La preuve de $(\mathcal{P}_6.2)$ se décompose aux lemmes suivants:

(i) $\forall\lambda\in\Lambda,\ A,A'\in S.\ (I_\lambda(\delta m(\lambda)(A,A'),A,A')\Rightarrow[\exists A''\in S,\ a\in A.\ t_a(A',A'')\ \vee\ \theta_\lambda(A,A')])$

Par l'absurde supposons $I_\lambda(\delta m(\lambda)(A,A'),A,A')$, $\neg\theta_\lambda(A,A')$ et $\forall A''\in S,a\in A.\ \neg t_a(A',A'')$. La contradiction est que nous pouvons construire par induction une chaîne strictement décroissante $\lambda,\lambda_1,\lambda_2,\dots$ d'ordinaux, comme suit:

- Puisque $I_\lambda(\delta m(\lambda)(A,A'),A,A')$ est vrai mais ni $(\mathcal{P}_7.3.a)$ ni $(\mathcal{P}_7.3.c)$ ne s'appliquent, $(\mathcal{P}_7.3.b)$ implique l'existence de $\lambda_1<\lambda$ tel que $\forall A''\in S.\ (\theta_{\lambda_1}(A',A'')\Rightarrow\exists\delta''<\delta m(\lambda)(A,A').\ I_\lambda(\delta'',A,A'))$. D'après $(\mathcal{P}_7.2)$ et (a), $I_{\lambda_1}(\delta m(\lambda_1)(A',A''),A',A')$ est vrai et nous ne pouvons pas avoir $\theta_{\lambda_1}(A',A')$ car autrement $\exists\delta''<\delta m(\lambda)(A,A').\ I_\lambda(\delta'',A,A')$, en contradiction avec (a).

- Supposons que nous ayons construit une séquence finie strictement décroissante $\lambda,\lambda_1,\dots,\lambda_i$, $i\geq 0$ telle que $I_{\lambda_i}(\delta m(\lambda_i)(A',A''),A',A')\wedge\neg\theta_{\lambda_i}(A',A')$ soit vrai. Nous pouvons la prolonger par λ_{i+1} puisque $(\mathcal{P}_7.3.a)$ et $(\mathcal{P}_7.3.c)$ ne s'appliquant pas, $(\mathcal{P}_7.3.b)$ implique l'existence d'un $\lambda_{i+1}<\lambda_i$ tel que $\forall A''\in S.\ (\theta_{\lambda_{i+1}}(A',A'')\Rightarrow\exists\delta''<\delta m(\lambda_i)(A',A'').\ I_{\lambda_i}(\delta'',A',A''))$ donc $\neg\theta_{\lambda_{i+1}}(A',A')$. De plus $I_{\lambda_{i+1}}(\delta m(\lambda_{i+1})(A',A'),A',A')$ découle de $(\mathcal{P}_7.2)$ et (a). C.Q.F.D.

(ii) $\forall\lambda'\in\Gamma,\ A,A'\in S.\ [(I_{\lambda'}(\delta',A,A')\wedge\neg\psi_{\lambda'}(A,A'))\Rightarrow\exists A''\in S.\ a\in A.\ t_a(A',A'')]$

Lorsque $I_{\lambda'}(\delta',A,A')$ est vrai nous avons $I_{\lambda_{m-1}}(\delta m(\lambda_{m-1})(A_{m-1},s_m),s_{m-1},\lambda_m)$. De plus [illegible] suivant [illegible] car $\psi_{\lambda'}(A,A)=\theta_\pi(A,A)=\psi_0(A_0,A_1)$. Maintenant (ii) dérive de (i). C.Q.F.D.

(k) $\forall \lambda \in \Lambda,\ \delta \in \Delta,\ A, A' \in S.\ [I_\lambda(\delta, A, A') \Rightarrow \exists A'' \in S.\ \theta_\lambda(A, A'')]$

Par induction transfinie sur λ, supposons (k) vrai pour $\forall \lambda' < \lambda$. Nous montrons que (k) est vrai pour λ, par l'absurde. Supposons donc $\forall A'' \in S.\ \neg\theta_\lambda(A, A'')$. Nous avons $I_\lambda(\delta_0, A, A'_0)$ avec $\delta_0 = \delta$ et $A'_0 = A'$. Supposons avoir construit une chaine $\delta_0 > \ldots > \delta_k$ avec $I_\lambda(\delta_k, A, A'_k)$. Puisque $\neg\theta_\lambda(A, A'_k)$, alors nous avons d'après ($\mathcal{B}_T$.3.a) ou ($\mathcal{B}_T$.3.b) et l'hypothèse d'induction, $\exists \delta_{k+1} < \delta_k.\ I_\lambda(\delta_{k+1}, A, A'_{k+1})$. La contradiction est que, de cette manière, nous pouvons construire une chaine infinie strictement décroissante d'ordinaux. Q.E.D.

(r) $\forall \gamma' \in \Gamma,\ A, A', A'' \in S.\ ([J(\gamma', A, A') \wedge \neg\Psi(A, A') \wedge \exists q \in A.\ t_q(A', A'')] \Rightarrow [\exists \gamma'' < \gamma'.\ J(\gamma'', A, A'')])$

Supposant $[J(\gamma', A, A') \wedge \neg\Psi(A, A') \wedge \exists q \in A.\ t_q(A', A'')]$ nous avons $A_m = A'$, $I_{\lambda_{m-1}}(\delta m(\lambda_{m-1})(A_{m-1}, A_m), A_{m-1}, A_m)$ et $\neg\theta_{\lambda_{m-1}}(A_{m-1}, A_m)$. ($\mathcal{B}_T$.3.c) ne s'appliquant pas, deux cas seulement sont à considérer :

(r.1) ($\mathcal{B}_T$.3.a) s'applique à $I_{\lambda_{m-1}}(\delta m(\lambda_{m-1})(A_{m-1}, A_m), A_{m-1}, A_m)$, de même ($\mathcal{B}_T$.3.b) pour un $\lambda' < \lambda_{m-1}$ tel que $\theta_{\lambda'}(A_{m-1}, A'')$.

Dans le premier cas nous avons $HS(\lambda_{m-1})(A_{m-1}, A', A'')$ et dans le second $LI(\lambda_{m-1})(A_{m-1}, A', \lambda', A'')$. Dans les deux cas (b) implique $I_{\lambda_{m-1}}(\delta m(\lambda_{m-1})(A_{m-1}, A''), A_{m-1}, A'')$ et à partir de $J(\gamma', A, A')$ donc $\neg\theta_{\lambda_{m-1}}(A_{m-1}, A')$ et (c) nous dérivons $\delta(\lambda_{m-1})(A_{m-1}, A') > \delta(\lambda_{m-1})(A_{m-1}, A'')$.

(r.1.1) Cas $\neg\theta_{\lambda_{m-1}}(A_{m-1}, A'')$

Nous venons de montrer en effet que $\delta(\lambda_{m-1})(A_{m-1}, A') > \delta(\lambda_{m-1})(A_{m-1}, A'')$ donc $\gamma'' = (\Sigma_0 + \ldots + \Sigma_{m-2} + \delta(\lambda_{m-1})(A_{m-1}, A'')) < (\Sigma_0 + \ldots + \Sigma_{m-2} + \delta(\lambda_{m-1})(A_{m-1}, A')) = \gamma'$. Si nous

(r.1.2) Cas $\theta_{\lambda_{m-1}}(A_{m-1}, A'')$

Soit ℓ le plus petit nombre naturel tel que $\ell \le m$ et $\theta_{\lambda_\ell}(A_\ell, A'')$ soit vrai de sorte que si $\ell > 0$ alors nous avons $\neg\theta_{\lambda_{\ell-1}}(A_{\ell-1}, A'')$. Intuitivement, considérer la transition $t_a(A', A'')$ dans la preuve provoque un retour de lemmes $\theta_{\lambda_{m-1}}, \ldots, \theta_{\lambda_\ell}$ utilisés récursivement.

Notons que d'après (e), nous avons $\delta(\lambda_j)(A_j, A'') = 0$ pour $j = \ell, \ldots, m-1$.

Montrons maintenant que $I_{\lambda_j}(\delta m(\lambda_j)(A_j, A''), A_j, A'')$ est vrai pour $j = \sup(\ell-1, 0), \ldots, m-1$. Le cas $j = m-1$ a été déjà considéré. Si $\sup(0, \ell-1) \le j < m-1$ alors $J(\gamma', A, A')$ implique $\exists \delta'' \in S.\ LI(\lambda_j)(A_j, A_{j+1}, \lambda_{j+1}, A'')$ donc (b), $\theta_{\lambda_{j+1}}(A_{j+1}, A'')$ et (a) entraînent $I_{\lambda_j}(\delta m(\lambda_j)(A_j, A''), A_j, A'')$.

(r.1.2.1) Cas $\ell = 0$

Définissons $J(\gamma'', A, A'')$ au moyen de la formule (f) en choisissant $n = 1$, $\lambda_0 = \pi$, $A_0 = A$, $A_1 = A''$ et $\gamma'' = \delta(\lambda_0)(A, A'') = 0$. $J(\gamma'', A, A'')$ est vrai parce qu'il revient à $I_{\lambda_0}(\delta m(\lambda_0)(A_0, A''), A_0, A'')$. De plus $J(\gamma', A, A')$ implique que $\gamma' = \Sigma_0 + \ldots + \Sigma_{m-2} + \delta(\lambda_{m-1})(A_{m-1}, A')$. Donc $\delta(\lambda_{m-1})(A_{m-1}, A') \ge \delta(\lambda_{m-1})(A_{m-1}, A'')$ implique $\gamma' > 0 = \gamma''$.

(r.1.2.2) Cas $\ell > 0$ donc $m > 1$

Définissons $J(\gamma'', A, A'')$ au moyen de la formule (f) en choisissant $n = \ell$, $\lambda_0, \ldots, \lambda_{\ell-1}$, $A_0, \ldots, A_{\ell-1}$ et $\Sigma_0, \ldots, \Sigma_{\ell-2}$ comme définis par $J(\gamma', A, A')$, $A_\ell = A''$ et $\gamma'' = (\Sigma_0 + \ldots + \Sigma_{\ell-2} + \delta(\lambda_{\ell-1})(A_{\ell-1}, A''))$. Alors $J(\gamma'', A, A'')$ est vrai car nous avons déjà montré que $I_{\lambda_{\ell-1}}(\delta m(\lambda_{\ell-1})(A_{\ell-1}, A''), A_{\ell-1}, A'')$ est vrai et $\neg\theta_{\lambda_{\ell-1}}(A_{\ell-1}, A'')$ découle de la définition de ℓ et A_ℓ.

Comme $\delta(\lambda_{m-1})(A_{m-1}, A') > \delta(\lambda_{m-1})(A_{m-1}, A'') = 0$, nous devons [illegible] avec $\lambda_{\ell-1}$ est vrai et $\gamma' > \gamma''$ comme nous le voulions. Nous savons $\lambda_{\ell-1} \in S.\ LI(\lambda_{\ell-1})(A_{\ell-1}, A_\ell, \lambda_\ell, A'')$ donc d'après (c), $LI(\lambda_{\ell-1})(A_{\ell-1}, A_\ell, \lambda_\ell, A'')$. Par définition de $\Sigma_{\ell-1}$ nous définissons $\Sigma_{\ell-1} \ge \delta(\lambda_{\ell-1})(A_{\ell-1}, A'')$. Il s'ensuit que $\gamma'' = (\Sigma_0 + \ldots + \Sigma_{\ell-2} + \delta(\lambda_{\ell-1})(A_{\ell-1}, A'')) \le$

(7.8) $(\mathcal{B}_7.3.b)$ s'applique à $I_{\lambda_{m-1}}(\delta m(\lambda_{m-1})(A_{m-1},A_m),A_{m-1},A_m)$ pour un $\lambda'<\lambda_{m-1}$ tel que $\neg\theta_{\lambda'}(A_m,\Delta'')$.

Intuitivement, nous obtenons $J(\delta'',A,\Delta'')$ à partir de $J(\delta',A,\Delta'')$ en conservant "dans une pile" tous les lemmes applicables en A_m (sauf les $\theta_{\lambda'}$ tels que $\theta_{\lambda'}(A_m,A_m)$ ou bien $\theta_{\lambda'}(A_m,\Delta'')$) et en considérant la transition $t_a(A_m,\Delta'')$

Si nous prenons λ_m égal à λ', alors d'après (7.2) nous avons $\lambda_m<\lambda_{m-1}$ et $\forall\Delta'''\in S.\,[\theta_{\lambda_m}(A_m,\Delta''')\Rightarrow\exists\delta'''<\delta m(\lambda_{m-1})(A_{m-1},A_m).\,I_{\lambda_{m-1}}(\delta''',A_{m-1},A_m)]$. Il s'ensuit que $\neg\theta_{\lambda_m}(A_m,A_m)$ car sinon $\exists\delta'''<\delta m(\lambda_{m-1})(A_{m-1},A_m).\,I_{\lambda_{m-1}}(\delta''',A_{m-1},A_m)$, en contradiction avec la définition (a) de $\delta m(\lambda_{m-1})(A_{m-1},A_m)$. De plus $(\mathcal{B}_7.2)$ implique $I_{\lambda_m}(\delta m(\lambda_m)(A_m,A_m),A_m,A_m)$.

Supposons avoir construit une chaîne $\lambda_{m+k}<\ldots<\lambda_m<\lambda_{m-1}$ avec $k\geq 0$ telle que $\forall j\in(k+1).\,[(\forall\delta'''\in S.\,[\theta_{\lambda_{m+j}}(A_m,\Delta''')\Rightarrow\exists\delta'''<\delta m(\lambda_{m+j-1})(A_{m-k^o_j},A_m).\,I_{\lambda_{m+j-1}}(\delta''',A_{m-k^o_j},\Delta''')]\wedge\neg\theta_{\lambda_{m+j}}(A_m,A_m)\wedge I_{\lambda_{m+j}}(\delta m(\lambda_{m+j})(A_m,A_m),A_m,A_m)]$, où $k^o_j=(j\div 2\to 1|0)$. Si $(\mathcal{B}_7.3.b)$ s'applique à $I_{\lambda_{m+k}}(\delta m(\lambda_{m+k})(A_m,A_m),A_m,A_m)$ alors il existe $\lambda_{m+k+1}<\lambda_{m+k}$ tel que $\forall\Delta'''\in S.\,[\theta_{\lambda_{m+k+1}}(A_m,\Delta''')\Rightarrow\exists\delta'''<\delta m(\lambda_{m+k})(A_m,A_m).\,I_{\lambda_{m+k}}(\delta''',A_m,\Delta''')]$ donc $\neg\theta_{\lambda_{m+k+1}}(A_m,A_m)$ d'après (a) et $I_{\lambda_{m+k+1}}(\delta m(\lambda_{m+k+1})(A_m,A_m),A_m,A_m)$ d'après $(\mathcal{B}_7.2)$.

Puisque la chaîne $\lambda_m,\ldots,\lambda_{m+k},\ldots$ d'ordinaux est strictement décroissante, elle doit être finie de sorte qu'il existe un k que nous notons k pour lequel $(\mathcal{B}_7.3.b)$ ne s'applique pas à $I_{\lambda_{m+k}}(\delta m(\lambda_{m+k})(A_m,A_m),A_m,A_m)$. $(\mathcal{B}_7.3.c)$ ne s'applique pas non plus car $\theta_{\lambda_{m+k}}(A_m,A_m)$ n'est pas vrai. Il découle que $(\mathcal{B}_7.3.a)$ s'applique de sorte que nous avons $HS(\lambda_{m+k})(A_m,A_m,\Delta'')$ De plus, $\exists A''_{m+j+1}\in S.\,LI(\lambda_{m+j})(A_{m-k^o_j},A_m,\lambda_{m+j+1},A''_{m+j+1})$ découle de $I_{\lambda_{m+j}}(\delta m(\lambda_{m+j})(A_{m-k^o_j},A_m),A_{m-k^o_j},A_m)$ pour $j=-1,\ldots,k-1$ et $(\mathcal{B}_7.$

Nous allons maintenant montrer que $\delta''<\delta'$ et $J(\delta'',A,\Delta'')$ à partir de δ' et $J(\delta',A,\Delta'')$ comme suit :

Puisque $\neg\theta_{\lambda_m}(\lambda_m,A_m)$ est vrai alors il existe un plus grand nombre naturel ℓ tel que $0\leq\ell\leq k$ et $\neg\theta_{\lambda_{m+\ell}}(A_m,\Delta'')$. Définissons $J(\delta'',A,\Delta'')$ au moyen de la formule (ξ) où n est $m+\ell-1$; $\lambda_j,j\in(n+\ell-1)$, $\lambda_j,j\in(m+\ell)$, $\Sigma_j,j\in(m-1)$ sont définis comme ci-dessus tandis que $\Delta=A_m=A_{m+1}=\ldots=A_{m+\ell}$, $A_{m+\ell+1}=\Delta''$, $\Sigma_j=\sup\{\delta(\lambda_j)(A_j,\Delta'''):LI(\lambda_j)(A_j,A_{j+1},\lambda_{j+1},\Delta''')\}$, $j=m-1,\ldots,m+\ell-1$ et $\delta''=(\Sigma_0+\ldots+\Sigma_{m+\ell-1}+\delta(\lambda_{m+\ell})(A_{m+\ell},A_{m+\ell+1}))$.

Nous avons déjà démontré que $\forall i=m,\ldots,m+\ell,\ldots,m+K.\,\exists\Delta''_i\in S.\,LI(\lambda_{i-1})(A_{i-1},A_i,\lambda_i,\Delta''_i)$ et $\forall i=m,\ldots,m+\ell-1,\ldots,m+K-1.\,\neg\theta_{\lambda_i}(A_i,A_{i+1})$. De plus nous avons $\neg\theta_{\lambda_{m+\ell}}(A_{m+\ell},A_{m+\ell+1})$ par définition de ℓ, $A_{m+\ell}$ et $A_{m+\ell+1}$. Nous avons aussi $I_{\lambda_{m+\ell}}(\delta m(\lambda_{m+\ell})(A_{m+\ell},A_{m+\ell+1}),A_{m+\ell},A_{m+\ell+1})$ qui est vrai, il est impliqué par $HS(\lambda_{m+K})(A_m,A_m,\Delta'')$ quand $\ell=K$, autrement $\ell<K$ et $\theta_{\lambda_{m+\ell+1}}(A_m,\Delta'')$ implique $\exists\delta'''.\,I_{\lambda_{m+\ell}}(\delta''',A_{m-k^o_{\ell+1}},\Delta'')$ donc d'après (a), $I_{\lambda_{m+\ell}}(\delta m(\lambda_{m+\ell})(A_m,\Delta''),A_m,\Delta'')$ est vrai. Nous conclurons que $J(\delta'',A,\Delta'')$ est vrai.

Il reste à montrer que $\delta''<\delta'$. Nous avons montré que $\forall j=-1,\ldots,K-1$ il existe un certain Δ''_{m+j+1} tel que $LI(\lambda_{m+j})(A_{m-k^o_j},A_m,\lambda_{m+j+1},\Delta''_{m+j+1})=LI(\lambda_{m+j})(A_{m+j},A_{m+j+1},\lambda_{m+j+1},\Delta''_{m+j+1})$. De plus $\forall i=1,\ldots,m+\ell$ nous avons $\neg\theta_{\lambda_i}(A_i,A_{i+1})$ de sorte que d'après (e), $\delta(\lambda_{m+j})(A_{m+j},A_{m+j+1})>(\sup\{\delta(\lambda_{m+j})(A_{m+j},\Delta'''):LI(\lambda_{m+j})(A_{m+j},A_{m+j+1},\lambda_{m+j+1},\Delta''')\}+\delta(\lambda_{m+j+1})(A_{m+j+1},A_{m+j+1}))=(\Sigma_{m+j}+\delta(\lambda_{m+j+1})(A_{m+j+1},A_{m+j+1}))$ pour tout $j=-1,\ldots,\ell-1$. Grâce à cette inégalité et à $A_{m+j+1}=A_{m+j+2}$ pour $j=-1,\ldots,\ell-2$, nous obtenons $\delta(\lambda_{m-1})(A_{m-1},A_m)>(\Sigma_{m-1}+\delta(\lambda_m)(A_m,A_{m+1})>\ldots>(\Sigma_{m-1}+\ldots+\Sigma_{m+\ell-1}+\delta(\lambda_{m+\ell})(A_{m+\ell},A_{m+\ell}))$. Si $\ell=K$ alors $HS(\lambda_{m+K})(A_m,A_m,\Delta'')$, $A_{m+\ell}=A_m$, $\neg\theta_{m+\ell}(A_m,A_m)$ et (e) impliquent $\delta(\lambda_{m+\ell})(A_{m+\ell},A_{m+\ell})>\delta(\lambda_{m+\ell})(A_{m+\ell},\Delta'')$. Autrement $\ell<K$ et nous avons $\neg\theta_{\lambda_{m+\ell}}(A_m,A_m)$, $LI(\lambda_{m+\ell})(A_m,A_m,\lambda_{m+\ell+1},\Delta'')$, $A_{m+\ell}=A_m$ de sorte d'après (e), $\delta(\lambda_{m+\ell})(A_{m+\ell},A_{m+\ell})=\delta(\lambda_{m+\ell})(A_m,A_m)>\delta(\lambda_{m+\ell})(A_m,\Delta'')=\delta(\lambda_{m+\ell})(A_{m+\ell},\Delta'')$. Dans les deux cas, nous concluons que $\delta'=(\Sigma_0+\ldots+\Sigma_{r-2}+\delta(\lambda_{m-1})(A_{m-1},A_m))>(\Sigma_0+\ldots+\Sigma_{r-2}+\Sigma_{r-1}+\ldots+\Sigma_{m+\ell-1}+\delta(\lambda_{m+\ell})(A_{m+\ell},\Delta''))=\delta''$. C.Q.F.D.

Exemple 5.4 - 1

le système de transition $\langle S, A, t, \Vdash \rangle$ correspondant au programme suivant (pris littéralement dans Dijkstra[77]) :

do odd(X) **and** X≥3 → X := X+1
▯ even(X) **and** X≥2 → X := X/2
od

est défini par :

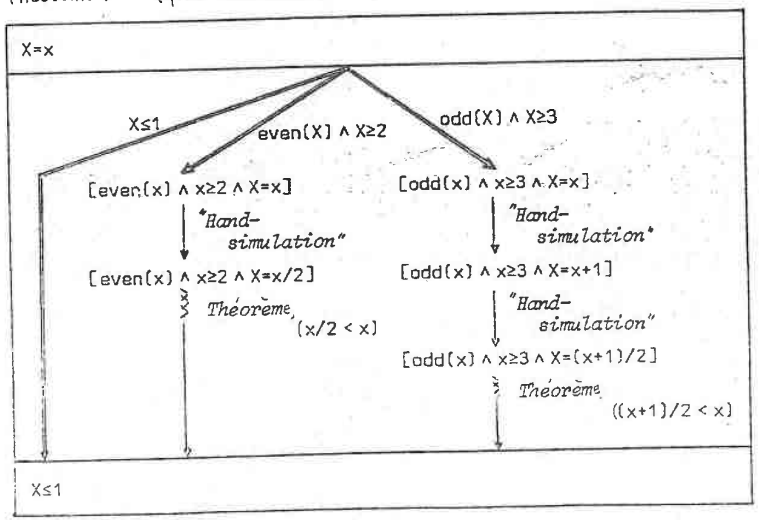
$S = \mathbf{Z}$

$A = \{a\}$

$t_a(x,x') = [(odd(x) \wedge x \geq 3 \wedge x' = x+1) \vee (even(x) \wedge x \geq 2 \wedge x' = x/2)]$

Une preuve que $\varphi(x,x') = [x' \leq 1]$ est fatale est donnée par la charte de preuve suivante :

Théorème : (par induction sur α)



Nous pouvons également utiliser le principe d'induction (β_2) avec :

$\Lambda = \omega+1$

$\theta_\omega = \psi$

$\theta_\lambda(x,x') = [(\lambda = \underline{\lambda}(x)) \Rightarrow (x' \leq 1)]$ quand $\lambda < \omega$ et $\underline{\lambda}(x) = (x \leq 1 \to 0 \mid x-1)$

$\pi = \omega$

$\Delta = 4$

$I_\omega(\delta, x, x') = [(\delta = 1 \wedge x = x') \vee (\delta = 0 \wedge x' \leq 1)]$

$I_\lambda(\delta, x, x') = [(\lambda = \underline{\lambda}(x)) \Rightarrow ((\delta = 3 \wedge x' = x) \vee (\delta = 2 \wedge t_{\underline{a}}(x,x')) \vee (\delta = 1 \wedge odd(x) \wedge t_{\underline{a}}^2(x,x')) \vee (\delta = 0 \wedge x' \leq 1))]$

D'après la définition (a) de $\delta m \in (\Lambda \to (S \times S \to \Delta))$, nous obtenons :

$\delta m(\lambda)(x,x') = 3$ ssi $[\lambda < \omega \wedge x = x' \wedge x' > 1]$

$= 2$ ssi $[\lambda < \omega \wedge t_{\underline{a}}(x,x') \wedge x' > 1]$

$= 1$ ssi $[(\lambda = \omega \wedge x = x' > 1) \vee (\lambda < \omega \wedge odd(x) \wedge t_{\underline{a}}^2(x,x') \wedge x' > 1)]$

$= 0$ ssi $[x' \leq 1]$

Le développement de (b) donne :

$HS(\omega)(x,x',x'') = [x = x' \wedge t_{\underline{a}}(x',x'') \wedge x'' \leq 1]$

quand $\lambda < \omega$,

$HS(\lambda)(x,x',x'') = [[(x = x') \vee (t_{\underline{a}}(x,x') \wedge [(\lambda = \underline{\lambda}(x)) \Rightarrow (odd(x) \vee x'' \leq 1)]) \vee (odd(x) \wedge t_{\underline{a}}^2(x,x') \wedge [(\lambda = \underline{\lambda}(x)) \Rightarrow (x'' \leq 1)])] \wedge t_{\underline{a}}(x',x'')]$

de la manière :

$LI(\omega)(x,x',\lambda',x'') = [x' = x > 1 \wedge \lambda' = \underline{\lambda}(x') \wedge x'' \leq 1]$

quand $\lambda < \omega$.

$LI(\lambda)(x,x',\lambda',x'') = [[(x = x') \vee t_{\underline{a}}(x,x') \vee (odd(x) \wedge t_{\underline{a}}^2(x,x'))] \wedge x' > 1 \wedge \lambda > \lambda' = \underline{\lambda}(x') \wedge x'' \leq 1]$

Noter que lorsque $(\beta_2.3.a)$ et $(\beta_2.3.b)$ sont tous deux vrais (par exemple quand $x = x' = 3$ et $x'' = 1$) alors $HS(\lambda)(x,x',x'')$ et $LI(\lambda)(x,x',\lambda',x')$ le sont aussi car il n'y a pas moyen [illegible]

Nous pouvons maintenant déterminer $\sigma(\lambda)(x,x')$. Parceque $LI(\lambda)(x,x',\lambda',x'')$ implique $\theta_\lambda(x,x'')$ donc $\sigma(\lambda)(x,x'')=0$, la formule (e) se réduit à :

$$\sigma(\lambda)(x,x') = \underset{\alpha\in\omega}{\mathrm{Min}}\{\alpha+1 : \neg\theta_\lambda(x,x') \wedge ([HS(\lambda)(x,x',x'') \wedge \alpha=\sigma(\lambda)(x,x'')] \vee [\exists\lambda'\in\Lambda, x''\in S.\ LI(\lambda)(x,x',\lambda',x'') \wedge \alpha=\sigma(\lambda')(x',x')])\}$$

Ce n'est pas nécessaire de chercher une définition non-récursive de σ, car nous n'aurons besoin que des propriétés suivantes :

- $t_{\underline{a}}(x,x') \Rightarrow [\neg\theta_{\underline{\lambda}(x)}(x,x) \wedge HS(\underline{\lambda}(x))(x,x,x')] \Rightarrow [\sigma(\underline{\lambda}(x))(x,x') < \sigma(\underline{\lambda}(x))(x,x)]$

- $[even(x) \wedge t_{\underline{a}}(x,x') \wedge t_{\underline{a}}(x',x'')] \Rightarrow [\neg\theta_{\underline{\lambda}(x')}(x',x') \wedge HS(\underline{\lambda}(x'))(x',x',x'') \wedge \neg\theta_{\underline{\lambda}(x)}(x,x') \wedge \exists z'''.\ LI(\underline{\lambda}(x))(x,x',\underline{\lambda}(x'),x''')] \Rightarrow [\sigma(\underline{\lambda}(x'))(x',x'') < \sigma(\underline{\lambda}(x'))(x',x') < \sigma(\underline{\lambda}(x))(x,x')]$

- $[odd(x) \wedge t_{\underline{a}}(x,x') \wedge t_{\underline{a}}(x',x'')] \Rightarrow [(\exists z'''.\ LI(\underline{\lambda}(x))(x,x''',\underline{\lambda}(x''),x''') \vee \theta_{\underline{\lambda}(x)}(x,x''')) \wedge HS(\underline{\lambda}(x))(x,x',x'')] \Rightarrow [((\sigma(\underline{\lambda}(x''))(x'',x''') < \sigma(\underline{\lambda}(x))(x,x''') \vee \sigma(\underline{\lambda}(x'''))(x'',x''')=0) \wedge \sigma(\underline{\lambda}(x))(x,x'') < \sigma(\underline{\lambda}(x))(x,x')] \Rightarrow [\sigma(\underline{\lambda}(x''))(x'',x''') < \sigma(\underline{\lambda}(x))(x,x')]$

- $[odd(x) \wedge t_{\underline{a}}^2(x,x') \wedge t_{\underline{a}}(x',x'')] \Rightarrow [\neg\theta_{\underline{\lambda}(x')}(x',x') \wedge HS(\underline{\lambda}(x'))(x',x',x'') \wedge \neg\theta_{\underline{\lambda}(x)}(x,x') \wedge \exists x'''.\ LI(\underline{\lambda}(x))(x,x',\underline{\lambda}(x'),x''')] \Rightarrow [\sigma(\underline{\lambda}(x'))(x',x'') < \sigma(\underline{\lambda}(x))(x,x')]$.

Dans la définition (f) de $J(\gamma',x,x')$, le cas $m=1$ se réduit à $[(x=x' \vee x'\le 1) \wedge \gamma'=\sigma(\omega)(x,x')]$. Autrement $m>1$ et pour $i\in(m\setminus 0)$ nous avons $\Sigma_{i-1}=0$ parceque $LI(\lambda_{i-1})(\lambda_{i-1},\lambda_i,\lambda_i,\lambda'')$ implique $\lambda''<1$ donc $\sigma(\lambda_{i-1})(\lambda_{i-1},\lambda'')=0$. De plus, λ est une fonction de Λ car $\lambda_0=\omega$ et $LI(\lambda_{i-1})(\lambda_{i-1},\lambda_i,\lambda_i,\lambda_i'')$ implique $\lambda_i=\underline{\lambda}(\lambda_i)$ pour tout $i\in(m\setminus 0)$. Quand $i=1$, ceci implique aussi $\lambda_0=\lambda_1=\omega>1$. Quand $i\in(2,\dots,m-1)$, les termes $\exists\lambda_i''\in S.\ LI(\lambda_{i-1})(\lambda_{i-1},\lambda_i,\lambda_i,\lambda_i'')$ sont de la forme :

$$\underline{\lambda}(\lambda_{i-1})>\underline{\lambda}(\lambda_i) \wedge \lambda_i>1 \wedge [(\lambda_{i-1}=\lambda_i) \vee t_{\underline{a}}(\lambda_{i-1},\lambda_i) \vee (odd(\lambda_{i-1}) \wedge t_{\underline{a}}^2(\lambda_{i-1},\lambda_i))]$$
$$= (\lambda_i>1 \wedge [even(\lambda_{i-1}) \wedge t_{\underline{a}}(\lambda_{i-1},\lambda_i)) \vee (odd(\lambda_{i-1}) \wedge t_{\underline{a}}^2(\lambda_{i-1},\lambda_i)]$$

parceque $\lambda_{i-1}=\lambda_i$ ne $\lambda_i=\lambda_{i-1}=1$, quand $odd(\lambda_{i-1}) \wedge t_{\underline{a}}^2(\lambda_{i-1},\lambda_i)$ ne sont pas compatibles avec $\underline{\lambda}(\lambda_{i-1})>\underline{\lambda}(\lambda_i)>\lambda_i>1$. Le terme $\neg\theta_{\lambda_i}(\lambda_i,\lambda_{i+1})$ revient à $\lambda_{i+1}>1$

$$J(\gamma',x,x') = \Big([(x=x' \vee x'\le 1) \wedge \gamma'=\sigma(\omega)(x,x')]$$
$$\vee [\exists n>1,\ \lambda\in(n+1\to S).\ ((x=\lambda_0=\lambda_1>1) \wedge (\lambda_n=x'>1)$$
$$\wedge \forall i\in(1,\dots,n-1).\ [(even(\lambda_{i-1}) \wedge t_{\underline{a}}(\lambda_{i-1},\lambda_i)) \vee (odd(\lambda_{i-1}) \wedge t_{\underline{a}}^2(\lambda_{i-1},\lambda_i))]$$
$$\wedge [(\lambda_{n-1}=\lambda_n) \vee t_{\underline{a}}(\lambda_{n-1},\lambda_n) \vee (odd(\lambda_{n-1}) \wedge t_{\underline{a}}^2(\lambda_{n-1},\lambda_n))]$$
$$\wedge \gamma'=\sigma(\underline{\lambda}(\lambda_{n-1}))(\lambda_{n-1},\lambda_n))]\Big)$$

Si nous posons $t'(x,x')$ égal à $[(even(x) \wedge t_{\underline{a}}(x,x')) \vee (odd(x) \wedge t_{\underline{a}}^2(x,x'))]$, ceci peut s'écrire plus simplement comme suit :

$$J(\gamma',x,x') = [\exists x''\in S.\ (t'^*(x,x'') \wedge (x'>1) \Rightarrow [(x''=x') \vee t_{\underline{a}}(x'',x') \vee (odd(x'') \wedge t_{\underline{a}}^2(x'',x'))] \wedge \gamma'=\sigma(\underline{\lambda}(x''))(x'',x'))]$$

Noter que cette formule met en évidence l'essence de la preuve par la méthode de Burstall qui consiste à considérer un pas pour les états pairs et deux pas pour les états impairs. Il reste à montrer que $J(\gamma',x,x')$ satisfait $(\mathcal{F}_c.1)$ (ce qui est évident) et $(\mathcal{F}_c.2)$. De manière évidente, si $\neg\Psi(x,x')$ alors $x'>1$ donc $\exists x'''\in S.\ t_{\underline{a}}(x',x''')$. Mais aussi, si $t_{\underline{a}}(x',x''')$ alors quatre cas doivent être considérés :

- Si $x'=x''$ alors $t'^*(x,x') \wedge t_{\underline{a}}(x',x''')$ implique $J(\sigma(\underline{\lambda}(x'))(x',x'''),x',x''')$ et $\sigma(\underline{\lambda}(x'))(x',x''') < \sigma(\underline{\lambda}(x'))(x'',x')$.

- Si $even(x'')$ et $t_{\underline{a}}(x',x')$ alors $t'^*(x,x'') \wedge (even(x'') \wedge t_{\underline{a}}(x'',x')) \wedge t_{\underline{a}}(x',x''')$ implique $t'^*(x,x') \wedge t_{\underline{a}}(x',x''')$ donc $J(\sigma(\underline{\lambda}(x'))(x',x'''),x,x''')$ et $\sigma(\underline{\lambda}(x'))(x',x''') < \sigma(\underline{\lambda}(x''))(x'',x')$.

- Si $odd(x'')$ et $t_{\underline{a}}(x',x')$ alors $t'^*(x,x'') \wedge (odd(x'') \wedge t_{\underline{a}}(x'',x') \wedge t_{\underline{a}}(x',x'''))$ implique $t'^*(x,x'') \wedge (x''=x')$ donc $J(\sigma(\underline{\lambda}(x'))(x'',x'''),x,x''')$ et $\sigma(\underline{\lambda}(x''))(x'',x''') < \sigma(\underline{\lambda}(x''))(x'',x')$.

- Si $odd(x'')$ et $t_{\underline{a}}^2(x',x')$ alors $t'^*(x,x'') \wedge odd(x'') \wedge t_{\underline{a}}^2(x',x'') \wedge t_{\underline{a}}(x',x''')$ implique $t'^*(x,x') \wedge t_{\underline{a}}(x',x''')$ donc $J(\sigma(\underline{\lambda}(x'))(x',x'''),x,x''')$ et $\sigma(\underline{\lambda}(x''))(x',x''') < \sigma(\underline{\lambda}(x''))(x'',x')$.

□

5.5 CHARTES DE PREUVE

Ayant montré que la méthode de Floyd est un cas particulier de la méthode de Burstall (après des généralisations adéquates), il nous reste à étudier une présentation uniforme des preuves par l'une ou l'autre des méthodes. A cet effet nous utiliserons une présentation graphique des preuves.

L'idée de présenter les preuves de programmes par des diagrammes acycliques fut introduite par Lamport [77] et développée ultérieurement par Owicki-Lamport [82] et Manna-Pnueli [82]. Cependant ces méthodes n'étaient pas sémantiquement complètes à cause d'un certain nombre de restrictions (comme l'impossibilité de faire des inductions infinies ou la restriction à des programmes dont le nombre d'état est fini (et petit), etc.). Notre formalisation est plus générale du fait qu'elle consiste à introduire des chartes de preuve bien-structurées présentant eventuellement des cycles et dont nous démontrons la correction et la complétude sémantique.

Finalement, nous montrons que les chartes de preuve sont adéquats pour démontrer les propriétés de fatalité des programmes parallèles.

5.5.1 DEFINITION D'UNE CHARTE DE PREUVE D'UN PROGRAMME

Une charte de preuve pour un système de transition $(S, \Sigma, \tau, \Theta)$ sera construite à partir d'un ensemble fini de graphes avec étiquettes aux sommets et aux arcs. Nous utilisons $I^E \rightsquigarrow I^S$ pour désigner un tel graphe avec un sommet d'entrée unique noté I^E

Nous définissons l'ensemble des graphes permis par une grammaire. Les graphes élémentaires sont de la forme $I \longrightarrow J$ où I est le sommet d'entrée J le sommet de sortie et il n'existe qu'un seul arc entre I et J. Il y a différents types d'arcs (que nous représentons différemment), certains pouvant être étiquetés (l'étiquette est alors écrite sur l'arc correspondant). Pour composer ces graphes, nous utilisons les opérations de composition suivantes :

- Si $I \rightsquigarrow J$ et $K \rightsquigarrow L$ sont deux graphes tels que $J=K$ alors $I \rightsquigarrow J \rightsquigarrow L$ dénote le graphe composé obtenu en confondant le sommet de sortie J avec le sommet d'entrée K. Il n'y a pas d'autres fusions possibles des sommets des graphes originaux et le sommet d'entrée (respectivement de sortie) du graphe composé est le sommet étiqueté I (respectivement L).

- Si $I \rightsquigarrow J$ et $K \rightsquigarrow L$ sont deux graphes tels que $I=K$ et $J=L$ alors $I \overset{\rightsquigarrow}{\rightsquigarrow} J$ dénote le graphe composé où les sommets d'entrée (respectivement de sortie) sont confondus, les autres sommets restant deux à deux distincts.

- Si $I \rightsquigarrow J$ et $K \rightsquigarrow L$ sont deux graphes tels que $I=K$ alors la boucle $I \overset{\rightsquigarrow}{\underset{\uparrow}{\rightsquigarrow}} J$ est le graphe composé avec le sommet d'entrée I confondu avec K, un sommet de sortie J et un nouvel arc reliant le sommet L au sommet d'entrée I.

Nous écrivons $I(\Lambda_0, \underline{\Lambda}, \vec{\Lambda}, \Lambda)$ (respectivement $I(\Lambda_0, \underline{\Lambda}, \vec{\Lambda}, \tilde{\Lambda}, \Lambda)$ et $I(\Lambda_0, \underline{\Lambda}, \tilde{\Lambda})$) pour désigner un sommet d'entrée avec l'étiquette I associé à un sommet d'un graphe appartenant à $(\{\text{entrée}\}^m \{\text{entrée}\} \{\text{sortie}\}^n \{\text{sortie}\})^+$ où $m=n$ (respectivement $m=n+1$, $n=0$), n étant le nombre de boucles imbriquées du graphe contenant ce sommet. D'une manière informelle, Λ_0 est la valeur de l'état d'entrée du programme, $\underline{\Lambda}$ (respectivement $\vec{\Lambda}_i$) est la valeur de l'état correspondant à l'entrée du graphe (respectivement à l'entrée de la $i$ème boucle imbriquée du graphe) et Λ est la valeur de l'état courant.

Définition 5.5.1 : 1 (Chartes de preuve)

Une charte de preuve pour $\langle S,A,t,\#\rangle$ est une paire $\langle\langle\Lambda,\prec\rangle,\{(G_\ell,(f_\ell,W_\ell,\prec_\ell)) : \ell\in\Lambda\}\rangle$ où $\langle\Lambda,\prec\rangle$ est un ensemble fini bien-fondé (de noms de graphes) et où $\forall \ell\in\Lambda$, $f_\ell\in(S^3\to W_\ell)$, $\omega f(W_\ell,\prec_\ell)$ et G_ℓ est une charte bien-formée $I_\ell^{\varepsilon_\ell}(A_0,\underline{A},A) \rightsquigarrow I_\ell^{\sigma_\ell}(A_0,\underline{A},A)$ générée par la grammaire de graphes suivante :

$J(s_0,\underline{s},\vec{s},s) \rightsquigarrow K(s_0,\underline{s},\vec{s},s) ::=$

$J(s_0,\underline{s},\vec{s},s) \longrightarrow K(s_0,\underline{s},\vec{s},s)$

ssi $\forall s_0,\underline{s},s\in S,\vec{s}\in S^n.[J(s_0,\underline{s},\vec{s},s)\Rightarrow(\exists s'\in S, a\in A.t_a^{(n,A)}\wedge \forall s'\in S, a\in A.(t_a^{(s,s')}\Rightarrow K(s_0,\underline{s},\vec{s},s'))]$

$J(s_0,\underline{s},\vec{s},s) \overset{\ell'}{\rightsquigarrow} K(s_0,\underline{s},\vec{s},s)$

ssi $\ell'\vdash\ell \wedge \forall s_0,\underline{s},s\in S,\vec{s}\in S^n.[J(s_0,\underline{s},\vec{s},s)\Rightarrow(I_{\ell'}^{\varepsilon_{\ell'}}(s_0,s,s)\wedge \forall s'\in S.(I_{\ell'}^{\sigma_{\ell'}}(s_0,s,s')\Rightarrow K(s_0,\underline{s},\vec{s},s'))]$

$J(s_0,\underline{s},\vec{s},s) \overset{\ell,(f_\ell,W_\ell,\prec_\ell)}{\rightsquigarrow} K(s_0,\underline{s},\vec{s},s)$

ssi $\forall s_0,\underline{s},s\in S,\vec{s}\in S^n.[J(s_0,\underline{s},\vec{s},s)\Rightarrow(f_\ell(s_0,s)\prec_\ell f_\ell(s_0,\underline{s})\wedge I_\ell^{\varepsilon_\ell}(s_0,s,s)\wedge\forall s'\in S.(I_\ell^{\sigma_\ell}(s_0,s,s')\Rightarrow K(s_0,\underline{s},\vec{s},s'))]$

$J(s_0,\underline{s},\vec{s},s) \longrightarrow K(s_0,\underline{s},\vec{s},s)$

ssi $\forall s_0,\underline{s},s\in S,\vec{s}\in S^n.[J(s_0,\underline{s},\vec{s},s)\Rightarrow K(s_0,\underline{s},\vec{s},s)]$

$J(s_0,\underline{s},\vec{s},s) \Rightarrow L^1(s_0,\underline{s},\vec{s},s) \mid L^2(s_0,\underline{s},\vec{s},s) \mid \dots \mid L^p(s_0,\underline{s},\vec{s},s) \Rightarrow K(s_0,\underline{s},\vec{s},s)$

ssi $\forall s_0,\underline{s},s\in S,\vec{s}\in S^n.[J(s_0,\underline{s},\vec{s},s)\Rightarrow \bigvee_{i=1}^{p} L^i(s_0,\underline{s},\vec{s},s)]$

$J(s_0,\underline{s},\vec{s},s) \rightsquigarrow L(s_0,\underline{s},\vec{s},s) \rightsquigarrow K(s_0,\underline{s},\vec{s},s)$

$J(s_0,\underline{s},\vec{s},s) \Rightarrow K(s_0,\underline{s},\vec{s},s) \mid L(s_0,\underline{s},\vec{s}.\tilde{s},s) \rightsquigarrow M(s_0,\underline{s},\vec{s}.\tilde{s},s)$, $M \xrightarrow{(f,W,\prec)} L$

$f\in(S^3\to W)\wedge W^*(W,\prec)\wedge\forall s_0,\underline{s},s,\tilde{s}\in S,\vec{s}\in S^n.($
$[J(s_0,\underline{s},\vec{s},s)\Rightarrow(K(s_0,\underline{s},\vec{s},s)\vee L(s_0,\underline{s},\vec{s}.s,s))]$
$\wedge[M(s_0,s,\vec{s}.\tilde{s},s)\Rightarrow([f(s_0,\underline{s},s)\prec f(s_0,\underline{s},\tilde{s})\wedge L(s_0,\underline{s},\vec{s}.s,s)]$

(Observons que les chartes de preuve sont des graphes reductibles, donc que nous aurions pu aussi formaliser à l'aide d'un langage bien structuré).

Nous pouvons démontrer que $\vdash$ est fatale pour $\langle S,A,\Sigma\langle S,A,t,\#\rangle\rangle$ en démontrant que :

Condition 5.5.1 : 2 (Preuves par chartes)

Il existe une charte de preuve $\langle\langle\Lambda,\prec\rangle,\{(I_\ell^{\varepsilon_\ell}(A_0,\underline{A},A)\rightsquigarrow I_\ell^{\sigma_\ell}(A_0,\underline{A},A),(f_\ell,W_\ell,\prec_\ell)):\ell\in\Lambda\}\rangle$ et $\pi\in\ell$ tels que :

$\forall A_0,\underline{A},A\in S.\left[I_\pi^{\varepsilon_\pi}(A_0,\underline{A},A)=[A=\underline{A}\wedge\Phi(\underline{A})]\ \wedge\ I_\pi^{\sigma_\pi}(A_0,\underline{A},A)=[A_0=\underline{A}\wedge\Psi(\underline{A},A)]\right]$

5.5.2 CORRECTION ET COMPLETUDE SEMANTIQUE DES PREUVES PAR CHARTES

Théorème 5.5.2 : 1 (Correction des preuves par chartes)

$(5.5.1:2) \Rightarrow ((\mathcal{B}_2)$ avec $m\in(\Lambda\to\underline{Ord}))$

Démonstration

Soit $\langle\langle\Lambda,\prec\rangle,\{(G_\ell,(f_\ell,W_\ell,\prec_\ell)):\ell\in\Lambda\}\rangle$ une charte de preuve. Puisque $\omega f(W_\ell,\prec_\ell)$ nous pouvons supposer, sans perte de généralité que $W_\ell\in\underline{Ord}$ et $\prec_\ell=<$ (éventuellement moins puissante utilisée des fonctions $\max_\ell$). Nous pouvons également supposer que $\Lambda\subseteq\omega$ et $\prec=<$ puisque Λ est fini. Chaque graphe G_ℓ étant fini, nous pouvons supposer que ses sommets prennent leurs noms dans un ensemble fini N_ℓ, le sommet j étant étiqueté par $I_\ell^j\in(S\times S\times S^{e(j)}\times S\to\{\#,\not\#\})$ où $e(j)$ est le nombre de boucles renfermant j. Soient π et σ tels

respectifs du sommet d'entrée unique et du sommet de sortie unique de G_ℓ.

Pour tout $\ell \in \Lambda$, nous considérons l'ensemble T_ℓ de tuples $\langle j, A_0, \underline{A}, \vec{A}, A\rangle$ tels que $j \in N_\ell$, $A_0, \underline{A}, A \in S$, $\vec{A} \in S^{e(j)}$ et $J^j_\ell(A_0, \underline{A}, \vec{A}, A)$ soit vrai. Définissons la relation binaire $\ll_\ell$ sur T_ℓ comme suit : $\langle j', A'_0, \underline{A}', \vec{A}', A'\rangle \ll_\ell \langle j, A_0, \underline{A}, \vec{A}, A\rangle$ si et seulement si :

soit $J^j_\ell \longrightarrow J^{j'}_\ell \quad \wedge\ s'_0 = s_0 \wedge \underline{s}' = \underline{s} \wedge \vec{s}' = \vec{s} \wedge \exists a \in A.\, t_a(s, s')$

ou $J^j_\ell \overset{\ell'}{\longrightarrow} J^{j'}_\ell \quad \wedge\ s'_0 = s_0 \wedge \underline{s}' = \underline{s} \wedge \vec{s}' = \vec{s} \wedge J^{\sigma_{\ell'}}_{\ell'}(s_0, s, s')$

ou $J^j_\ell \overset{\ell, (f_\ell, W_\ell, \prec_\ell)}{\longrightarrow} J^{j'}_\ell \quad \wedge\ s'_0 = s_0 \wedge \underline{s}' = \underline{s} \wedge \vec{s}' = \vec{s} \wedge J^{\sigma_\ell}_\ell(s_0, s, s')$

ou $((J^j_\ell \longrightarrow J^{j'}_\ell) \vee (J^j_\ell \circ\!\!\Longrightarrow J^{j'}_\ell) \vee (J^j_\ell \rightarrowtail\!\!\circ\!\!\Longrightarrow J^{j'}_\ell)) \wedge s'_0 = s_0 \wedge \underline{s}' = \underline{s} \wedge \vec{s}' = \vec{s} \wedge s' = s$

ou $J^j_\ell \circ\!\!\Longrightarrow J^{j'}_\ell \quad \wedge\ s'_0 = s_0 \wedge \underline{s}' = \underline{s} \wedge \vec{s}' = \vec{s}.s \wedge s' = s$

sinon $J^j_\ell \rightarrowtail\!\!\circ\!\!\Longrightarrow J^{j'}_\ell \wedge\ s'_0 = s_0 \wedge \underline{s}' = \underline{s} \wedge \vec{s} = \vec{s}'.s \wedge s' = s$

Supposons que $\langle j_k, A_{i_k}, \underline{A}_k, \vec{A}_k, A_k\rangle : k \geq 0\rangle$ est une séquence infinie strictement décroissante pour $\ll_\ell$. Il s'ensuit que $\langle j_k : k \geq 0\rangle$ est un chemin infini dans le graphe fini G_ℓ, c'est donc un cycle. Alors il existe un sommet j de G_ℓ (de type $J^i_\ell \rightarrowtail\!\!\circ$) tel que la séquence $\langle j, A_{o_{i_k}}, \underline{A}_{i_k}, \vec{A}_{i_k}, \tilde{A}_{i_k}, A_{i_k}\rangle : k \geq 0\rangle$ d'éléments de $\langle j_k, A_{i_k}, \underline{A}_k, \vec{A}_k, A_k\rangle : k \geq 0\rangle$ tels que $j_k = j$ est infinie. Ceci est en contradiction avec $\forall k \geq 0.\, (f(A_{o_{i_k}}, \underline{A}_{i_k}, \tilde{A}_{i_k}) \prec f(A_{o_{i_k}}, \underline{A}_{i_k}, \tilde{A}_{i_k}))$, $f \in (S^3 \to W)$ et $\underline{wf}(W, \prec)$. Par l'absurde, nous avons $\underline{wf}(T_\ell, \ll_\ell)$.

Nous choisissons $\Lambda_\ell = \Lambda$, $\varepsilon_{2_\ell}(A_0, A) = \bar{J}^{\varepsilon_\ell}_\ell(A_0, A, A)$, $\theta_{2_\ell}(A_0, \underline{A}, A) = \bar{J}^{\sigma_\ell}_\ell(A_0, \underline{A}, A)$,

$\Delta_2 = \sup^+ \{ rk(W_\ell, \prec_\ell) : \ell \in \Lambda \}$, $f_{2_\ell}(A_0, A) = rk(W_\ell, \prec_\ell)(f_\ell(A_0, A))$, $\pi_{2_\ell} = (rk(T_\ell, \ll_\ell) + 1)$, $\pi_2 = \pi$,

$\bar{I}^i_{2_\ell}(A_0, \underline{A}, A) = [\exists \underline{i} \in N_\ell,\ \vec{A} \in S^{e(j)}.\ (\bar{J}^j_\ell(A_0, \underline{A}, \vec{A}, A) \wedge \iota = rk(T_\ell, \ll_\ell)(\langle j, A_0, \underline{A}, \vec{A}, A\rangle))]$ quand $\iota < \pi_\ell$

$\bar{I}^{\pi_\ell}_{2_\ell}(A_0, \underline{A}, A) = \bar{I}^{\varepsilon_\ell}_\ell(A_0, \underline{A}, A)$.

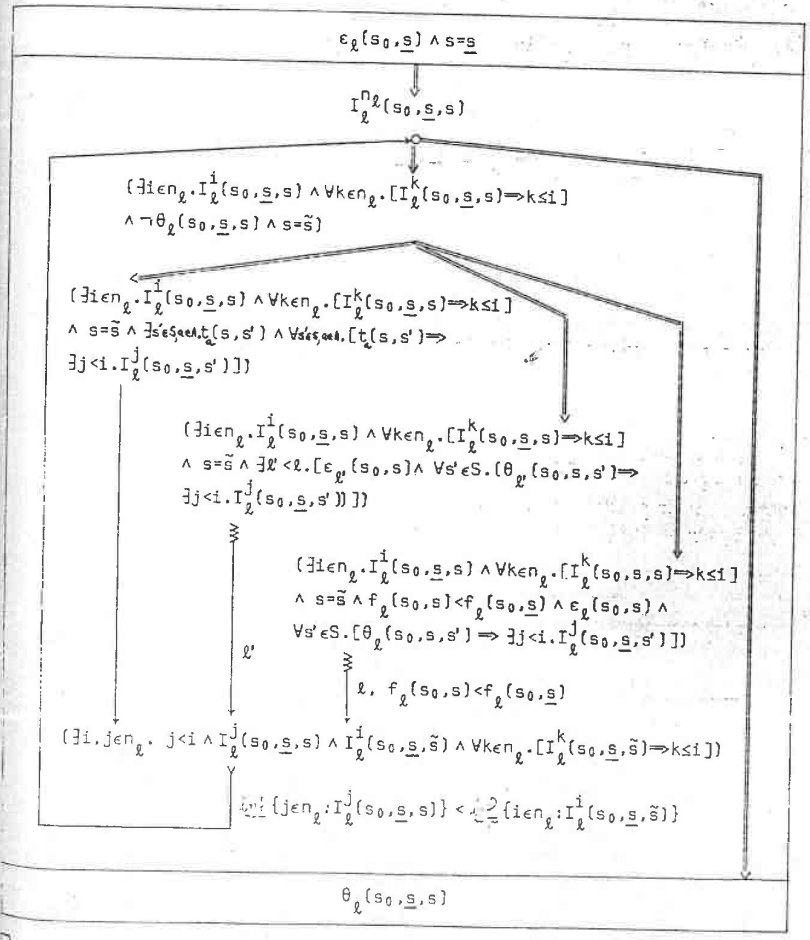
□

Théorème 5.5.2.N.2 (Complétude sémantique des preuves par chartes)

$(\Phi_2) \Longrightarrow (5.5.1.2)$

Démonstration

Nous pouvons représenter une preuve par (Φ_2) par la charte de preuve $\langle\langle \Lambda, \vdash\rangle, (G_\ell, (f_\ell, \underline{O}_{\ell'}, \prec)) : \ell \in \Lambda\rangle$ où chaque graphe G_ℓ, $\ell \in \Lambda$ est la charte suivante :



□

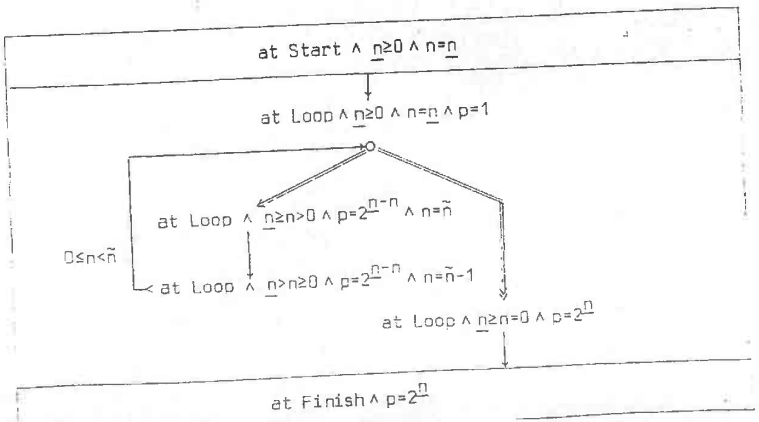
5.5.3 EXEMPLES DE PRESENTATION DE PREUVES PAR CHARTES

Les chartes permettent aussi bien de présenter des preuves par la méthode de Burstall que par la méthode de Floyd, généralisant les deux méthodes en les unifiant. Elles permettent également de traiter le cas des programmes parallèles.

5.5.3.1 Présentation de preuves "à la Floyd" par des chartes

Nous dirons qu'une preuve de fatalité par une charte est une preuve "à la Floyd" quand la charte de preuve a la forme de la charte (ou organigramme) du programme. Dans ce cas les assertions utilisées dans la charte de preuve sont des invariants au sens de Floyd

Par exemple, une preuve "à la Floyd" de correction totale du programme 5.3.1-1 peut également être présentée comme suit :



5.5.3.2 Preuve de propriétés de fatalité de programmes parallèles asynchrones

Puisque nous pouvons représenter un programme parallèle par un système de transition non-déterministe, les chartes de preuve peuvent également s'appliquer aux preuves de fatalité de programmes parallèles.

Exemple 5.5.3.2-1 (Correction totale d'un programme parallèle)

Considérons une version parallèle asynchrone du programme 5.3.1-1 qui calcule 2^m quand $m \geq 0$:

```
1:  N1:=0; N2:=N;
2:  [[ 11:  P1:=1
       12:  if N1+1 < N2 then
       13:        T1:=N1+1; P1:=2xP1;
       14:        N1:=T1;
       15:  fi;  goto 12;
       16:
    ||
       21:  P2:=1;
       22:  if N1+1 < N2 then
       23:        T2:=N2-1; P2:=2xP2;
       24:        N2:=T2;
       25:  fi;  goto 22;
       26:
    ]];
3:  P :=  if N1+1=N2 then 2xP1xP2 else P1xP2 fi;
4:
```

Nous écrivons at_j (respectivement at_{ij}) à la place de $c=i$ (respectivement $c_i=j$, où c (respectivement c_i) est le point de contrôle du programme (respectivement du numéro i lorsque le contrôle est dans une commande parallèle). Nous écrivons in_E pour $\vee\{at\ l : l \in E\}$ et nous simplifierons $\overset{P}{\underset{Q \vee R}{\downarrow}}$ en $\overset{P}{\underset{Q \quad R}{\swarrow \searrow}}$.

Dans la charte de preuve de correction totale

P: (at1 ∧ $m_0>0$ ⇝ at4 ∧ $p=2^{m_0}$), nous distinguons deux cas :

- Le cas $m_0 \leq 1$ se traite par le lemme

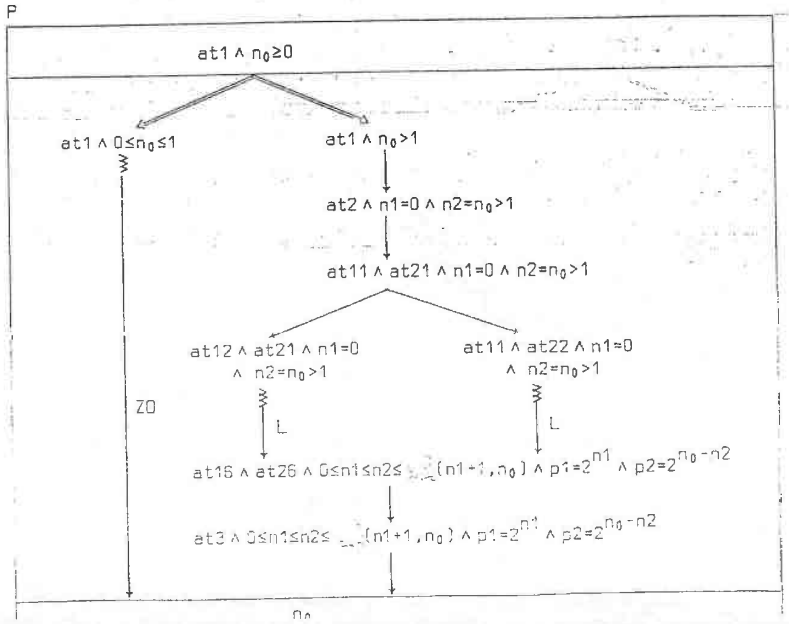
Z0: (at1 ∧ $0 \leq m_0 \leq 1$ ⇝ at4 ∧ $p=2^{m_0}$). Ce lemme peut être démontré par évaluation symbolique et nous laissons le lecteur faire la charte de preuve correspondante.

- Le cas principal $m_0>1$ se traite par le lemme

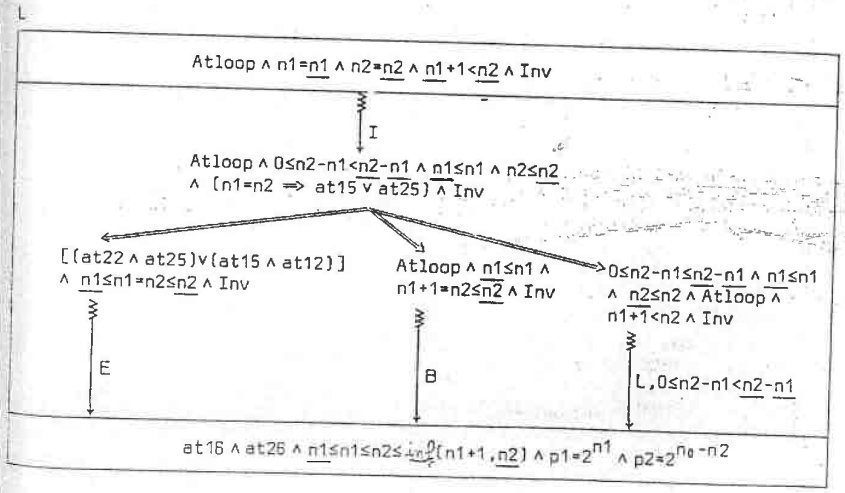
L: (Atloop ∧ $n1=\underline{n}1$ ∧ $m2=\underline{m}2$ ∧ $(\underline{n}1+1)<\underline{m}2$ ∧ Inv ⇝ at16 ∧ at26 ∧ $\underline{n}1 \leq n1 \leq m2 \leq \inf(m1+1, \underline{m}2)$ ∧ $p1=2^{n1}$ ∧ $p2=2^{m_0-m2}$) où Atloop remplace ([at12 ∧ in{21,...,25}] ∨ [in{11,...,15} ∧ at22]) et Inv est l'invariant suivant :

$$Inv = [(at11 \Rightarrow n1=0 \mid p1=2^{n1} \times (at14 \rightarrow 2\mid 1)) \wedge (at14 \Rightarrow t1=n1+1) \wedge$$
$$(at21 \Rightarrow m2=m_0 \mid p2=2^{m_0-m2} \times (at24 \rightarrow 2\mid 1)) \wedge (at24 \Rightarrow t2=m2-1)]$$

P est la charte de preuve suivante :



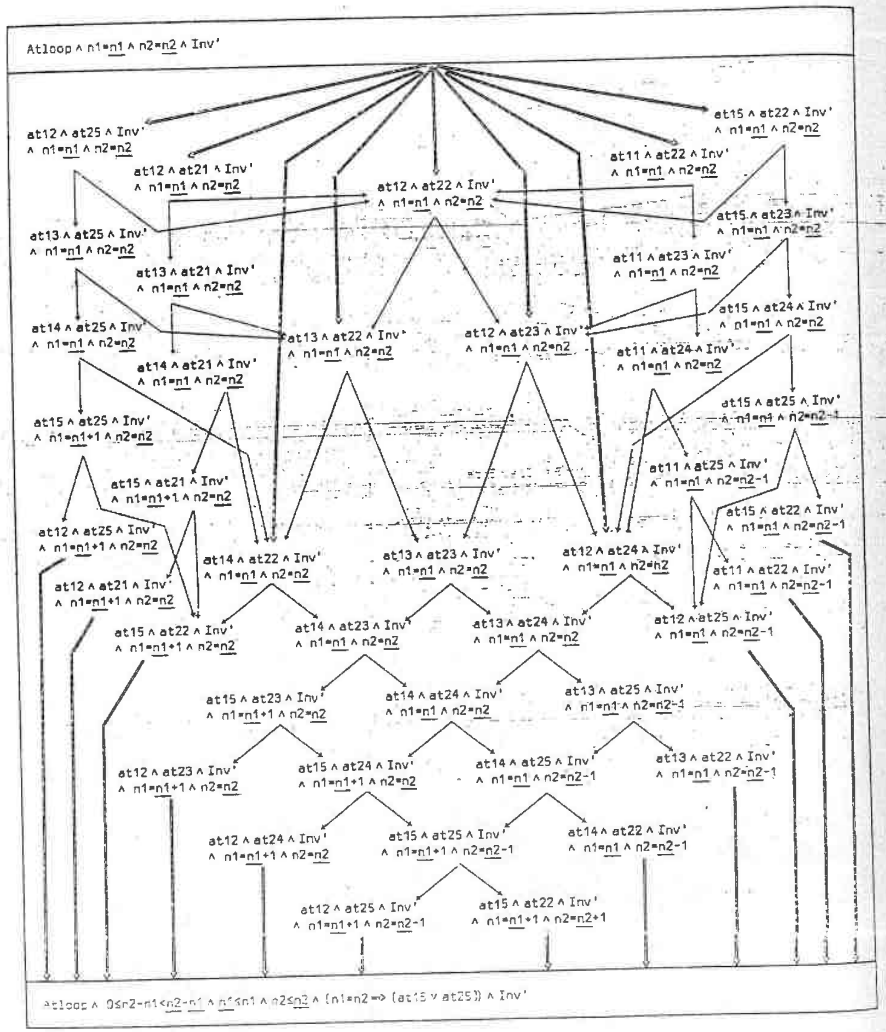
La preuve du lemme L se fait par induction sur $(m2-m1)$ qui décroit strictement à chaque itération de boucle dans l'un des deux processus. Cette itération est décrite par le lemme I: (Atloop ∧ $m1=\underline{m}1$ ∧ $m2=\underline{m}2$ ∧ $(\underline{n}1+1<m2)$ ∧ Inv ⇝ Atloop ∧ $0<(m2-m1)<(\underline{m}2-m1)$ ∧ $\underline{n}1 \leq m1$ ∧ $m2 \leq \underline{m}2$ ∧ $(\underline{n}1+1)<\underline{m}2$ ∧ Inv ∧ $(m1=m2 \Rightarrow (at15 \vee at25))$). Nous avons $m1 \leq m2 \leq (m1+1)$. Le cas $m1=m2$ se traite par le lemme E: (in{12,15,16} ∧ in{22,25,26} ∧ $n1=\underline{n}1$ ∧ $p1=\underline{p}1$ ∧ $m2=\underline{m}2$ ∧ $p2=\underline{p}2$ ∧ $(m1+1) \geq m2$ ⇝ at16 ∧ at26 ∧ $\underline{n}1=n1$ ∧ $\underline{p}1=p1$ ∧ $\underline{m}2=m2$ ∧ $\underline{p}2=p2$). Sa preuve est triviale par évaluation symbolique et nous laissons sa charte au lecteur. Le cas $m2=(m1+1)$ se traite par le lemme B: (Atloop ∧ $m1=\underline{m}1$ ∧ $(m1+1)=m2=\underline{m}2$ ∧ Inv ⇝ at16 ∧ at26 ∧ $\underline{n}1 \leq m1 \leq m2 \leq \inf(m1+1, \underline{m}2)$ ∧ $p1=2^{n1}$ ∧ $p2=2^{n_0-m2}$). La charte de preuve L est la suivante :



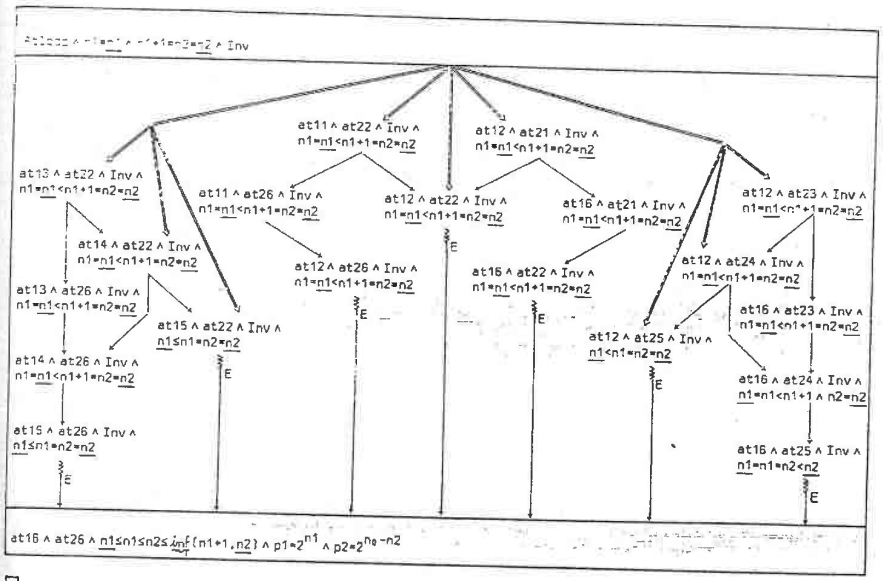
Les preuves des lemmes I et B ne présentent aucune difficulté et nous pouvons les faire entièrement par evaluation symbolique.

Nous prenons $Inv = (Inv \wedge (\underline{n}1+1<\underline{m}2)$ dans la charte de preuve I :

I


Atloop ∧ n1=n1 ∧ n2=n2 ∧ Inv'
at12 ∧ at25 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at12 ∧ at21 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at12 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at15 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at11 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at13 ∧ at25 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at13 ∧ at21 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at15 ∧ at23 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at11 ∧ at23 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at14 ∧ at25 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at14 ∧ at21 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at13 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at12 ∧ at23 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at15 ∧ at24 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at11 ∧ at24 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at15 ∧ at25 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at15 ∧ at21 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at11 ∧ at25 ∧ Inv' ∧ n1=n1 ∧ n2=n2-1
at12 ∧ at25 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at15 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2-1
at12 ∧ at21 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at14 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at13 ∧ at23 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at12 ∧ at24 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at11 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2-1
at15 ∧ at22 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at14 ∧ at23 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at13 ∧ at24 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at12 ∧ at25 ∧ Inv' ∧ n1=n1 ∧ n2=n2-1
at15 ∧ at23 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at14 ∧ at24 ∧ Inv' ∧ n1=n1 ∧ n2=n2
at13 ∧ at25 ∧ Inv' ∧ n1=n1 ∧ n2=n2-1
at12 ∧ at23 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at15 ∧ at24 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at14 ∧ at25 ∧ Inv' ∧ n1=n1 ∧ n2=n2-1
at13 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2-1
at12 ∧ at24 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2
at15 ∧ at25 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2-1
at14 ∧ at22 ∧ Inv' ∧ n1=n1 ∧ n2=n2-1
at12 ∧ at25 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2-1
at15 ∧ at22 ∧ Inv' ∧ n1=n1+1 ∧ n2=n2+1
Atloop ∧ 0≤n2-n1<n2-n1 ∧ n1≤n1 ∧ n2≤n2 ∧ (n1=n2 ⇒ (at15 ∨ at25)) ∧ Inv'

II


Atloop ∧ n1=n1 ∧ n2=n2 ∧ Inv
at11 ∧ at22 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at12 ∧ at21 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at13 ∧ at22 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at11 ∧ at26 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at12 ∧ at22 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at16 ∧ at21 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at12 ∧ at23 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at14 ∧ at22 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at12 ∧ at24 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at13 ∧ at26 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at12 ∧ at26 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at16 ∧ at22 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at16 ∧ at23 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at15 ∧ at22 ∧ Inv ∧ n1≤n1=n2=n2
at12 ∧ at25 ∧ Inv ∧ n1<n1=n2=n2
at14 ∧ at26 ∧ Inv ∧ n1=n1<n1+1=n2=n2
at16 ∧ at24 ∧ Inv ∧ n1=n1<n1+1 ∧ n2=n2
at15 ∧ at26 ∧ Inv ∧ n1≤n1=n2=n2
at16 ∧ at25 ∧ Inv ∧ n1=n1=n2<n2
E
at16 ∧ at26 ∧ n1≤n1≤n2≤inf{n1+1, n2} ∧ p1=2^n1 ∧ p2=2^n0-n2

5.5.3.3 Preuve de propriétés de fatalité de programmes parallèles faiblement équitables

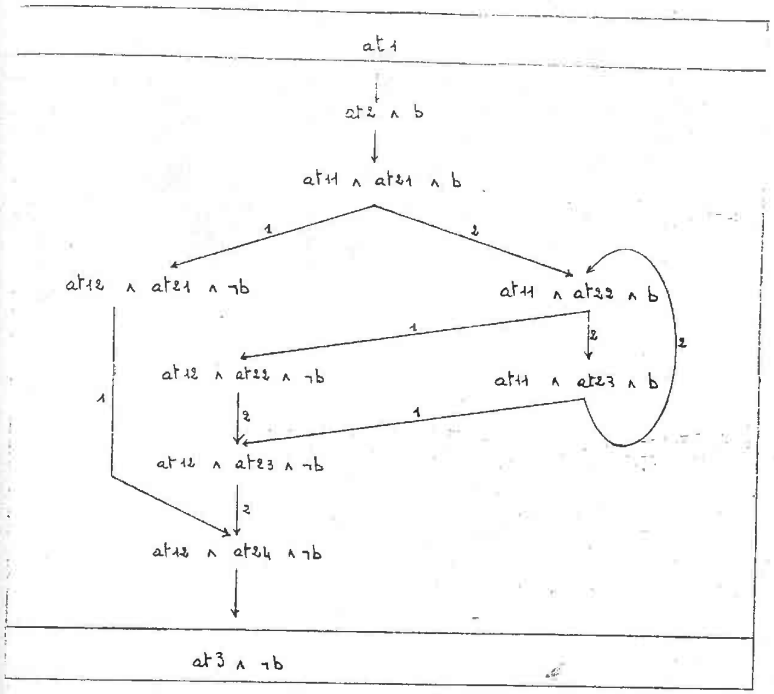
Les chartes de preuve peuvent également s'appliquer aux preuves de programmes parallèles faiblement équitables. Pour ce faire, nous mettons à sur chaque arc de la charte correspondant à l'évaluation symbolique d'un pas du processus Prw_i d'un programme parallèle faiblement équitable $[\![Prw_1]\!] \dots [\!] Prw_{m-1}]\!]$. Puisque la méthode soit complète, nous autorisons la présence de cycles dans la définition 5.5.1:1 pour lesquels il n'y a pas de preuve de terminaison à la Floyd par un triplet $\langle f, W, \prec \rangle$. Dans ce cas, il faudra simplement démontrer qu'il y a un processus toujours actif et jamais activé le long de ce cycle (si le cycle fait intervenir un lemme, ceci devra être démontré par une preuve d'invariance séparée). La terminaison découle alors de manière évidente de l'hypothèse d'équité faible. D'après le principe d'induction ($\mathcal{F}_{15}$), cette approche est complète.

Exemple 5.5.3.3-1

Pour l'exemple classique :

```
1:
    B: true ;
2:
    [[ 11:
           B:= false ;
       12:
    [] 21:
           while B do
       22:
               skip ;
       23:
           a := a+1
       24:
    ]] ,
3:
```

nous avons la charte de preuve suivante :



Le long du cycle, le processus 1 est toujours activable et jamais activé.

□

5.5.3.4 Preuve de propriétés de fatalité de programmes parallèles synchrones

En ce qui concerne les programmes parallèles synchrones, l'essentiel du travail a été fait au paragraphe 2.8.5. D'après 2.8.5.2.5 nous n'avons à considérer que les traces faiblement équitables engendrées par un système de transition. Par conséquent, la méthode des preuves par chartes du paragraphe précédent est directement applicable!

Exemple 5.5.3.4-1

Considérons le programme synchrone 2.8.5.4 qui réalise une section critique. Supposons que le deuxième processus soit en section critique. Nous démontrons que le premier processus entrera fatalement en section critique au moyen de la charte de preuve suivante :

5.6 REFERENCES

APT K.R., DE BAKKER J.W. [80], "Semantics of nondeterminism", Rapport de Recherche 80-01, LITP, (Paris), (Jan. 1980), 30p.

APT K.R., DE BAKKER J.W. [82], "Semantics of nondeterminism", Rapport de Recherche 82-70, LITP, (Paris), (Jan. 1983), 21p.

APT K.R., OLDEROG E.R. [82], "Proof rules dealing with fairness", (extended Abstract), Proc. Logics of Programs, Lect. notes in Comp. Sci. 131, Springer-Verlog, (1982), 1-8.

APT K.R., PLOTKIN G.D. [82], "Countable nondeterminism and random assignment" Research report 82-7, Dept. Comp. Sci., Edimburg U., (Feb. 1982), 40p.

BERG H.K., BOEBERT W.E., FRANTA W.R., MOHER T.G. [82], "Formal methods of program verification and specification", Prentice-Hall, (1982).

BURSTALL R.M. [74], "Program proving as hand simulation with a little induction", IFIP 74, North-Holland Pub. Co., (1974), 308-312.

COUSOT P. [81], "Semantic foundations of program analysis", in Program Flow Analysis, Theory and Applications, S.S. Muchnick - N.D. Jones (Eds), Prentice-Hall, (1981), 303-342.

COUSOT P., COUSOT R. [80], "Reasoning about program invariance proof methods", Rapport de Recherche CRIN-80-P050, (1980).

COUSOT P., COUSOT R. [83], "A la Floyd" induction principles for proving inevitability properties of programs", Rapport de Recherche CRIN-83-P3, a paraître dans "Paradigms of Programming" (M. Nivat & J. Reynolds, eds), Cambridge University Press.

COUSOT P., COUSOT R. [83a], "A la Floyd induction principles for proving inevitability properties of programs", (Nov. 1983).

LIVERCY C. [78], "Théorie des programmes", Dunod, (1978).

LUCKHAM D.C., SUZUKI N. [75], "Automatic program verification IV : proof of termination within a weak logic of programs", Comp. Sci. Report 522, Standford U., (1975).

MANNA Z., PNUELI A. [74], "Axiomatic approach to total correctness", Acta Informatica 3, (1974), 243-263.

MANNA Z., PNUELI A. [82], "Verification of concurrent programs : proving eventualities by well-founded ranking", STAN-CS-82-915, Comp. Sci. Dept., Standford U., (May 1982).

MANNA Z., PNUELI A. [83], "Verification of concurrent programs : a temporal proof system", stan-cs-83-967, Comp. Sci. Dept., Standford U., (June 1983).

MANNA Z., WALDINGER R.J. [78], "Is SOMETIME sometimes better than ALWAYS ?, intermittent assertions in proving program correctness", CACM 21, 2 (1978), 159-172.

NAUR P. [66], "Proof of algorithms by general snapshots", BIT 6, (1966), 310-316.

OWICKI S., GRIES D. [76], "An axiomatic proof technique for parallel programs I", Acta Informatica, 6 (1976), 319-340.

OWICKI S., LAMPORT L. [82], "Proving liveness properties of concurrent programs", TOPLAS 4, 3 (July 1982), 455-495.

PARK D. [81], "A predicate transformer for weak fair iteration", Proc. 6th [illegible] Symp. on Math. Foundations of Comp. Sci., Computation aspects of programs, Hakone, Japan (1981), 358-387.

PNUELI A. [77], "The temporal logic of programs", Proc. 18th Symp. on [illegible] Found. of Comp. Sci., Providence, RI (1977), 46-57.

SCHWARZ J. [76], "Event-based reasoning - a system for proving correct termination of programs", Proc. ICALP 3, Edinburgh, (1976) 131-146.

6. CONCLUSION

6. CONCLUSION

6.1 SEMANTIQUE OPERATIONNELLE

6.2 PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

6.3 PREUVES D'INVARIANCE ET DE FATALITE

6.4 REFERENCES

se retrouve cachée dans la condition $(\mathcal{F}_3)$-(d) puisque le but ne peut pas être satisfait par un état indéfini).

- Lamport [80] et Emerson [81] choisissent $\underset{\sim}{Suff}(\langle S, A, \Sigma\rangle)$ avec, dit le premier, l'idée que ce choix est nécessaire pour exprimer que "la façon dont le calcul évolue dans le futur ne dépend que de son état courant". Cet argument nous semble incomplet, la notion de sémantique close (cf. 2.6.8) étant mieux adaptée. Quand la sémantique est fermée par suffixes, il est possible d'exprimer certaines propriétés de la sémantique sans recourir aux indices pour désigner des états dans les traces et donc sans utiliser (directement) la notion de temps. Par exemple, l'absence de fatalité dans l'assertion $\Psi \in \mathcal{Y}(S \to \{\#, \#\!\#\})$ pour $\langle S, A, \Sigma\rangle$ appartient à $\underset{\sim}{Suff}(\langle S, A, \Sigma\rangle)$ peut s'exprimer par :

$$\forall p \in \Sigma'.\ \exists p' \in \Sigma'.\ (p' \twoheadrightarrow p \wedge \Psi(p'_\bullet))$$

L'inconvénient principal de la fermeture par suffixes nous semble être que la notion d'états initiaux (cf. 2.3:1) disparaît.

- Hoare [81] choisit $\underset{\sim}{Pref}^{<\omega}(\langle S, A, \Sigma\rangle)$ ce qui revient à considérer non seulement les calculs complets ou maximaux mais également des calculs en cours ou initiaux. La condition de fermeture par préfixes exprime à l'idée que le préfixe d'un calcul en cours est également un calcul en cours. De plus, l'idée de Hoare est qu'il est possible de raisonner sur des comportements infinis (c'est-à-dire $\underset{\sim}{Flim}(\underset{\sim}{Pref}^{<\omega}(\langle S, A, \Sigma\rangle))$) en n'utilisant que des traces finies (c'est-à-dire $\underset{\sim}{Pref}^{<\omega}(\langle S, A, \Sigma\rangle)$), ce qui, comme nous l'avons vu précédemment pour Pratt [82] n'est pas toujours possible (quand $\underset{\sim}{Flim}(\underset{\sim}{Pref}^{<\omega}(\langle S, A, \Sigma\rangle)) \neq \underset{\sim}{Pref}(\langle S, A, \Sigma\rangle)$). Fondamentalement, quand on raisonne sur la fermeture par préfixes d'une sémantique, il est plus possible d'exprimer, par exemple, qu'il peut survenir des pannes aléatoires car dans ce cas certains états peuvent être le dernier état d'une trace finie terminée (quand il y a une panne)

qui est également préfixe strict d'une autre trace (quand la panne n'a pas lieu). Avec le point de vue de Hoare, le pas d'arrêt sur panne doit se traiter par passage dans un état indéfini n'ayant pas de successeurs.

– Pour éviter les inconvénients inhérents aux propositions précédentes, Arnold-Nivat [82] choisissent de décrire l'ensemble de traces Σ par trois ensembles à savoir ce qu'ils appellent les "comportements initiaux" (c'est-à-dire les préfixes finis $\underset{\sim}{Pref}^{<\omega}(<S,A,\Sigma>)$ des traces de Σ), les "comportements finis terminés" (c'est-à-dire les traces finies $\Sigma \cap \Sigma^{<\omega}<S,A>$ de Σ) et les "comportements infinis" (c'est-à-dire les traces infinies $\Sigma \cap \Sigma^{\omega}<S,A>$ de Σ). Il faut alors ajouter un certain nombre de conditions de cohérence (les comportements initiaux sont fermés par préfixes, les comportements finis terminés sont des comportements initiaux, les préfixes finis des comportements infinis sont des comportements initiaux).

Par comparaison avec ces modèles de sémantique opérationnelle basés sur la notion de traces, notre choix s'avère plus simple et aussi expressif. En utilisant des opérateurs conventionnellement définis sur les sémantiques, nous pouvons retrouver tous les modèles précédemment cités comme cas particuliers.

Nous avons modélisé le parallélisme par le mélange non déterministe d'actions atomiques. En principe il est possible de déterminer l'ordre réel des actions et comme deux actions atomiques concurrentes ne peuvent avoir aucune influence l'une sur l'autre, nous pouvons supposer qu'elles ont été exécutées dans n'importe quel ordre. Pour des actions qui ne sont pas atomiques il est alors nécessaire de

les subdiviser en actions atomiques plus petites.

Ce point de vue est bien adapté à l'expression de l'exclusion mutuelle mais pas à celle de la simultanéité (deux actions s'exécutent en même temps). Pour ce faire, on peut considérer qu'une action pour un programme parallèle est un vecteur d'actions pour chacun de ses processus, une action spéciale pouvant (comme dans Arnold-Nivat [82]) désigner qu'un processus est inactif à l'instant. On peut évidemment avoir besoin de décrire non seulement des simultanéités mais également des recouvrements. Pour ce faire, on peut imaginer de marquer le début et la fin de chaque action par un événement $a \in A$ de la sémantique $<S,A,\Sigma>$.

Pour décrire l'ensemble de traces Σ, nous avons utilisé l'intermédiaire d'un système de transition. De manière plus générale, on peut imaginer (à la suite de Lamport [78]) de définir un ordre partiel sur des événements (marquant le début et la fin de chaque action) satisfaisant un certain nombre de contraintes d'ordonnancement (du genre les événements relatifs à un même processus sont totalement ordonnés, la fin de l'envoi d'un message précède le début de sa réception, etc...). On peut évidemment combiner ces différentes idées en décrivant un programme parallèle à l'aide d'un système de transition pour chaque processus, d'un ordre partiel sur des événements exprimant des conditions de synchronisation et d'une condition globale sur les traces exprimant les hypothèses d'équité.

6.2 PROPRIETES D'INVARIANCE ET DE FATALITE DES PROGRAMMES

Nous avons étudié les propriétés d'invariance conditionnelle et de fatalité sous invariance à cause de leur importance et de leur simplicité. En combinant ces deux types de propriétés, il est possible d'exprimer la plupart des propriétés relatives à la correction des programmes.

Il aurait cependant été aisé de les généraliser mais ce, au prix de complications qui nous ont semblées inutiles.

Par exemple, nous aurions pu prendre comme état initial non pas l'état de départ de la trace mais un état intermédiaire quelconque.

Pour l'invariance relationnelle, nous aurions alors la définition :

$$\forall p \in \Sigma.\ \forall i \in |p|.\ \forall j \in |p|.\ (j > i \Rightarrow \psi(p_i, p_j))$$

tandis que la fatalité relationnelle serait définie par :

$$\forall p \in \Sigma.\ \forall i \in |p|.\ \exists j \in |p|.\ (j > i \wedge \psi(p_i, p_j))$$

mais ce type de formule revient à considérer une propriété d'invariance ou de fatalité telles que nous les avons définies précédemment sur un ensemble de suffixes $\underline{\text{Suff}}(\langle S, A, \Sigma \rangle)$.

Il est également possible de considérer des propriétés relatives au passé plutôt qu'au futur comme par exemple (Clarke-Pnueli [81]) :

$$\forall p \in \Sigma,\ i \in |p|.\ \exists k < |p|.\ [k \leq i \wedge \psi(p_k, p_i) \wedge \forall j \in \omega.\ [k < j \leq i \Rightarrow \phi(p_j, p_i)]]$$

mais là encore il suffit de raisonner sur une sémantique transformée pour adapter les principes d'induction.

Les logiques temporelles (Lamport [80]) du type "linear type logic" comprennent des formules du genre

$$\models \Box\psi \quad \text{et} \quad \models \Diamond\psi$$

qui s'interprètent, pour une sémantique donnée $\langle S, A, \Sigma \rangle$, respectivement par

$$\forall p \in \Sigma.\ \forall i \in |p|.\ \psi(p^{\geq i}) \quad \text{et} \quad \forall p \in \Sigma.\ \exists i \in |p|.\ \psi(p^{\geq i})$$

Les logiques temporelles (Lamport [80]) du type "branching type logic" comprennent des formules du genre :

$$\models \text{ALL}\,\psi \qquad \models \text{SOME}\,\psi \qquad \models \text{POT}\,\psi \quad \text{et} \quad \models \text{INEV}\,\psi$$

qui s'interprètent respectivement par les formules :

$$\forall a \in S.\ \forall p \in \Sigma.\ \forall i \in |p|.\ [(p_i = a) \Rightarrow \forall j \in |p|.\ [(j \geq i) \Rightarrow \psi(p_j)]]$$

$$\forall a \in S.\ \exists p \in \Sigma.\ \exists i \in |p|.\ [(p_i = a) \wedge \forall j \in |p|.\ [(j \geq i) \Rightarrow \psi(p_j)]]$$

$$\forall a \in S.\ \exists p \in \Sigma.\ \exists i \in |p|.\ [(p_i = a) \wedge \exists j \in |p|.\ [(j \geq i) \wedge \psi(p_j)]]$$

$$\forall a \in S.\ \forall p \in \Sigma.\ \forall i \in |p|.\ [(p_i = a) \Rightarrow \exists j \in |p|.\ [(j \geq i) \wedge \psi(p_j)]]$$

Ces logiques sont utilisées pour faire des spécifications et faire des preuves. En ce qui concerne les spécifications il s'agit d'exprimer des propriétés des traces. Ceci peut se faire avec la logique ordinaire qui permet d'exprimer des assertions arbitraires sur les traces d'exécution. Les défenseurs des logiques temporelles pensent que cette généralité n'est pas nécessaire et qu'on peut se restreindre à quelques modalités bien choisies. Sans entrer dans l'énumération de toutes les modalités possibles, nous nous contentons d'une comparaison avec les logiques linéaire et arborescente de Lamport [80], brièvement décrites ci-dessus :

- Remarquons qu'il n'est pas possible de lier l'état courant p_i à l'état initial p_0 sauf en utilisant des variables auxiliaires comme dans la preuve du théorème 4.3.1.4 (2). Ceci nous amène

à des discussions ultérieures (cf. 4.3.2 à 4.5 et 5.3.1.6) et à notre préférence pour l'utilisation explicite de variables auxiliaires dans la preuve plutôt qu'à l'utilisation implicite de variables auxiliaires dans un programme transformé.

- Les propriétés universelles de la "linear type logic" sont bien adaptées pour exprimer des propriétés de correction de programmes non déterministes exécutés en profondeur (une alternative est choisie puis menée à son terme) ou de programmes parallèles. C'est ce genre de propriétés que nous avons retenues (cf. ch.3). En s'inspirant des logiques du type "branching time", il est également possible de considérer des propriétés de la forme :

$$\exists p \in \Sigma.\ \forall i \in |p|.\ \psi(p_0, p_i)$$
$$\exists p \in \Sigma.\ \exists i \in |p|.\ \psi(p_0, p_i)$$

Nous n'avons pas étudié ces propriétés existentielles car il est très facile d'adapter ce que nous avons fait pour les propriétés universelles correspondantes et nous avons voulu éviter ce qui peut paraître comme d'évidentes redites.

- Il est bien connu qu'il existe des propriétés des programmes qu'on peut spécifier avec une logique de type "linear time" et pas avec une logique de type "branching time" et inversement (Lamport [80], Gruj [84]). Il existe également des propriétés de programmes qui ne peuvent s'exprimer ni par l'une ni par l'autre de ces logiques (Clarke-al [81]). Ceci explique la multiplication de propositions concernant l'introduction de nouvelles modalités et montre que le choix des spécifications modalités auxquelles on peut se restreindre n'est pas du tout évident. C'est pourquoi nous avons utilisé au chapitre 2 une méthode de spécification qui nous semble tout à fait satisfaisante puisqu'elle utilise directement l'ensemble des traces p de la sémantique en utilisant des indices $i \in |p|$ pour désigner l'état p_i du calcul à un instant i quelconque.

6.3 PREUVES D'INVARIANCE ET DE FATALITE

On peut également utiliser ces logiques pour faire des preuves et ceci nous amène à une comparaison avec les chapitres 4 et 5. En logique, les preuves se font à l'aide de systèmes formels comportant des axiomes (dont certains expriment la sémantique du programme) et des règles d'inférence.

- Il semble difficile d'utiliser cette méthode pour formaliser des méthodes de preuve de programmes indépendamment du langage de programmation utilisé. Par exemple Apt-Delporte [83] ont essayé de formaliser la méthode des assertions intermittentes de Burstall à l'aide d'une logique temporelle pour en démontrer la correction et la complétude arithmétique. Cette étude est faite pour la correction totale de programmes itératifs (du style de ceux considérés en 2.8.1) et on ne voit pas, par exemple, comment généraliser à d'autres propriétés du même genre ou au cas des programmes parallèles. La formalisation de méthodes de preuve à l'aide de nos principes d'induction nous semble donc plus abstraite et générale.

- Pour formaliser l'induction utilisée dans les preuves de programmes, les règles d'inférence pour ces logiques d'ordre supérieur sont une sorte de généralisation naturelle de celle de Peano (Manna [74]) :

$$[[\psi(0) \wedge \forall n \in \omega.\ [\psi(n) \Rightarrow \psi(n+1)]] \Rightarrow \forall n \in \omega.\ \psi(n)]$$

ou même l'induction transfinie sur les ordinaux (qui est plus générale)

$$[\forall x \in \delta.((\forall y \in x.\ \psi(y)) \Rightarrow \psi(x))] \Rightarrow \forall x \in \delta.\ \psi(x)$$

Les autres niveaux d'induction peuvent évidemment se déduire (puisque c'est ce principe d'induction que nous avons utilisé pour démontrer la correction de tous nos principes d'induction). Cette forme de généralité permet par exemple à Manna-Pnueli [81a,b] de formaliser les méthodes de Floyd et Burstall en ajoutant (puisque exclusivement) que les assertions considérées (pour les programmes séquentiels) sont de la forme :

$$\vdash (at\ l \wedge \phi) \Rightarrow \Diamond (at\ l' \wedge \psi)$$

A notre avis, cette forme de généralités n'apprend pas grand chose pour mieux comprendre ces méthodes alors que notre formalisation du chapitre 5 est beaucoup plus précise. De plus pour arriver à un système de déduction utilisable en pratique, il est nécessaire de déduire des axiomes et règles d'inférence de base, un grand nombre de règles dérivées (il y a 24 axiomes, 4 règles d'inférence de base et 62 règles dérivées dans Manna [81]). Cette profusion de règles est à contraster avec la concision des principes d'induction du chapitre 5.

- Il est bien évident qu'en pratique, on s'intéresse à la preuve simultanée de plusieurs propriétés d'un programme. Dans ce cas, on s'efforce d'éviter les répétitions en combinant autant que faire se peut les différentes preuves. Nous avons abordé à plusieurs reprises ce problème (cf. par exemple la discussion en 4.3.2.2.6) qui semble être ignoré dans la démarche des logiques temporelles (qui ne permettent guère de démontrer $\vdash \Box\phi \wedge \Diamond\psi$ si on ne démontre $\vdash \Box\phi$ et $\vdash \Diamond\psi$ séparément).

- Par contre, les logiques temporelles permettent d'exprimer aisément des propriétés du type $\vdash \Box\Diamond\psi$ combinant plusieurs modalités, (d'où l'avantage de faire porter ψ (aussi bien que $\Diamond\psi$) sur des (suffixes de) traces plutôt que sur des (paires d') états). Malheureusement, il n'est en général pas prévu d'axiomes ou de règles d'inférence pour les preuves comportant de telles combinaisons de modalités. Nous devons dire que nous n'avons pas nous non plus, abordé ce problème.

- Ceci nous amène enfin au problème de la structuration des preuves en particulier pour les gros programmes. Nous avons proposé diverses méthodes de décompositions des preuves, basées sur la sémantique des programmes (cf. par exemple 4.3.2.4 et 5.2.7) et généralisé l'idée de Burstall, classique en mathématiques, de décomposition de la preuve d'un théorème à l'aide de lemmes. Cette étude devrait pouvoir être approfondie notamment dans le cas des programmes distribués (pour éviter les phénomènes d'interférence) et des programmes modulaires (pour faire correspondre les lemmes aux modules). Pour explorer cette voie, on pourrait s'inspirer de Jones [85].

6.4 REFERENCES

ARNOLD A. [83], "An axiomatization of the intermittent assertion method", Rapport de Recherche 83-10, LITP, Paris, (Jan. 1983), 31p.

ARNOLD A., NIVAT M. [82], "Comportements de processus", Rapport de Recherche 82-12, LITP, Paris, (Février 1982), 34p.

CLARKE Ed., FRANCEZ N., GUREVICH Y., SISTLA A. [81], "Can message buffers be characterized in linear temporal logic", Rapport de Recherche, Harvard U., (1981), 14p.

EMERSON E.A. [81], "Alternative semantics for temporal logics", Rapport de Recherche TR-182, Texas U., Austin, (Oct. 1981).

GRAF S. [84], "On Lamport's comparison between linear and branching time temporal logic", RAIRO Inf. Théorique, 18, 4 (1984), 345-353.

HOARE C.A.R. [81], "A calculus of total correctness for communicating processes", SCP 1 (1981), 49-72.

JONES C.B. [85], "The role of proof obligations in software design", TAPSOFT 85, Lect. Notes in Comp. sci. 186, (1985), 27-41.

LAMPORT L. [78], "Time, clocks and the ordering of events in a distributed system", CACM 21, 7 (July 1978), 558-565.

LAMPORT L. [80], "'Sometime' is sometimes 'not never'", 7th annual ACM Symp. on principles of programming languages, (1980), 174-185.

MANNA Z. [81], "Verification of sequential programs: Temporal axiomatization", Rapport de Recherche STAN-CS-81-877, Dept. of Comp. Sci. Stanford U., (Sept 1981), 45p.

MANNA Z., PNUELI A. [81a], "Verification of concurrent programs, part I: the temporal logic framework", Rapport de Recherche STAN-CS-81-836, Dept. of Comp. Sci., Stanford U., (1981), 62p.

MANNA Z., PNUELI A. [81b], "Verification of concurrent programs, part II: temporal proof principles", Rapport de Recherche STAN-CS-81-843, Dept. of Comp. Sci., Stanford U., (1981), 51p.

PRATT V.R. [82], "On the composition of processes", 9th ACM annual Symp. on principles of programming languages, (1982), 213-223.

ANNEXE I :

NOTATIONS MATHEMATIQUES

ANNEXE I :

NOTATIONS MATHEMATIQUES

ANNEXE I :

NOTATIONS MATHEMATIQUES

I.1 NOTATIONS DE LOGIQUE

Nous utilisons la logique du premier ordre avec égalité de manière intuitive. Les variables logiques sont notées $x, z, \ldots$, les prédicats $P, Q, \ldots$ et nous utilisons la convention habituelle que $P(x)$ signifie que x peut apparaître comme variable libre dans P. Nous utilisons les symboles logiques $P \vee Q$ (disjonction), $P \wedge Q$ (conjonction), $\neg P$ (négation), $P \Rightarrow Q$ (implication), $P \Leftrightarrow Q$ (équivalence), $x = y$ (égalité), $\forall x.P(x)$ (quantification universelle) $\exists x.P(x)$ (quantification existentielle) et diverses formes de parenthèses $(,)$, $[,]$, $\ldots$. Nous notons $P \Leftarrow Q$, $P \not\Rightarrow Q$, $P \not\Leftarrow Q$, $x \neq y$, ... respectivement pour $Q \Rightarrow P$, $\neg(P \Rightarrow Q)$, $\neg(Q \Rightarrow P)$, $\neg(x = y)$, ... $(P \Rightarrow Q | R)$ est l'abréviation de $((P \wedge Q) \vee (\neg P \wedge R))$, $x = (P \rightarrow y | z)$ celle de $(P \Rightarrow x = y | x = z)$, $\exists! x.P(x)$ celle de $\exists y.\forall x.(x = y \Leftrightarrow P(x))$ où y n'est pas libre dans P. Nous écrivons $\forall x_0 \in X_0, \ldots, x_i \in X_i, \ldots \bullet P(x_0, \ldots, x_i, \ldots)$ pour $\forall x_0 \bullet \ldots \forall x_i \bullet \ldots (x_0 \in X_0 \Rightarrow \ldots (x_i \in X_i \Rightarrow \ldots P(x_0, \ldots, x_i, \ldots) \ldots) \ldots)$ et de même avec les quantificateurs existentiels. La valeur de vérité vraie $(x = x)$ est notée tt tandis que la valeur de vérité fausse $(x \neq x)$ est notée ff.

Par abus de notation les quantificateurs universels sont parfois omis. Les valeurs de vérité tt et ff sont parfois confondues avec leurs interprétations qui sont donc également notées tt et ff. Les prédicats sont également souvent confondus avec leurs interprétations, c'est-à-dire qu'ils sont compris comme des fonctions des domaines de valeurs de leurs variables libres dans l'ensemble $\{tt, ff\}$ des valeurs de vérité. Dans ce cas nous notons $P(x_0, \ldots, x_{i-1})$ où $x_0, x_1, \ldots$ sont les seules variables qui peuvent apparaître libres dans P et dans certains cas l'expression $P(x)$ est notée P_x.

I.2 NOTATIONS ENSEMBLISTES ELEMENTAIRES

Nous utilisons l'axiomatisation de la théorie des ensembles de Zermelo-Fraenkel et admettons l'axiome du choix. Les prédicats de cette théorie sont ceux d'une logique avec egalité où les notions de classes $X, X', X_0, \dots, X_i, \dots, x, x', x_0, \dots, x_i, \dots$ et d'appartenance $\in$ sont considerées comme primitives. X est un ensemble si et seulement si il existe Y tel que $X \in Y$. $\mathrm{set}(X)$ est l'abréviation de $\exists Y.(X \in Y)$.

Si $P(x)$ est un prédicat et X n'est pas libre dans P, alors $\{x: P(x)\}$ est l'unique classe X telle que $\forall x.(x \in X \leftrightarrow (\mathrm{set}(x) \wedge P(x)))$ c'est-à-dire la classe de tous les ensembles x tels que $P(x)$. Nous écrivons $\{x_0, \dots, x_i, \dots\}$ pour $\{x: x = x_0 \vee \dots \vee x = x_i \vee \dots\}$. Plus généralement si $\tau(x_0, \dots, x_i, \dots)$ est un terme et $P(x_0, \dots, x_i, \dots)$ un prédicat de la théorie des ensembles dans lesquels X n'apparaît pas, nous définissons $\{\tau(x_0, \dots, x_i, \dots): P(x_0, \dots, x_i, \dots)\} = \{X: \exists x_0, \dots, x_i, \dots .(P(x_0, \dots, x_i, \dots) \wedge X = \tau(x_0, \dots, x_i, \dots))\}$. $\{\tau \in Y: P\}$ est l'abréviation de $\{\tau: \tau \in Y \wedge P\}$.

L'inclusion est définie par $X \subseteq Y \iff \forall z.(z \in X \Rightarrow z \in Y)$, l'inclusion stricte par $X \subset Y \iff (X \subseteq Y \wedge X \neq Y)$. Nous écrivons également $X \supseteq Y$ pour $Y \subseteq X$, $X \not\subseteq Y$ pour $\neg(X \subseteq Y)$, etc...

L'ensemble vide zéro noté 0 ou ϕ est $\{x: x \neq x\}$.

Nous définissons l'union $X \cup Y = \{x: x \in X \vee x \in Y\}$, l'union infinie $\bigcup X = \{x: \exists y \in X.(x \in y)\}$ et $\bigcup_{i \in I} A_i = \bigcup\{A_i: i \in I\}$ quand A est une famille indexée d'ensembles. De même pour l'intersection $X \cap Y = \{x: x \in X \wedge x \in Y\}$, $\bigcap X = \{x: \forall y \in X.(x \in y)\}$ et $\bigcap_{i \in I} A_i = \bigcap\{A_i: i \in I\}$. La différence de classes est $X \setminus Y = \{x: x \in X \wedge x \notin Y\}$ avec la notation particulière $X \sim x = X \setminus \{x\}$ pour la différence avec un singleton. La puissance est $\mathcal{P}X = \{x: x \subseteq X\}$. Si x est la paire ordonnée $\langle x, y\rangle = \{\{x\}, \{x, y\}\}$ nous notons $x_1 = x(0) = x$ et $x_2 = x(1) = y$. Ceci permet de définir le produit cartesien

Une classe r est une relation (binaire) si tout membre x de r est une paire ordonnée c'est-à-dire que $\forall x.[(x \in r) \Rightarrow (\exists y, z.(x = \langle y, z\rangle))]$. Le domaine d'une relation r est $\mathrm{dom}(r) = \{x: \exists y.(\langle x, y\rangle \in r)\}$, son co-domaine est $\mathrm{rng}(r) = \{y: \exists x.(\langle x, y\rangle \in r)\}$, son champ est $\mathrm{fld}(r) = \mathrm{dom}(r) \cup \mathrm{rng}(r)$. Une fonction f est une relation telle que $\forall x, y, z.((\langle x, y\rangle \in f \wedge \langle x, z\rangle \in f) \Rightarrow (y = z))$ ce qui justifie l'emploi de la notation fonctionnelle $f(x)$, fx ou f_x pour désigner l'unique y, s'il existe, tel que $\langle x, y\rangle \in f$. Nous notons $X \to Y$ la classe des fonctions partielles de X dans Y c'est-à-dire $\{f: f \text{ est une fonction} \wedge \mathrm{dom}(f) \subseteq X \wedge \mathrm{rng}(f) \subseteq Y\}$ et $X \to Y$ la classe des fonctions totales de X dans Y c'est-à-dire $\{f: f \in (X \to Y) \wedge \mathrm{dom}(f) = X\}$. La composition fonctionnelle est définie par $f \circ g(x) = f(g(x))$.
Si $f \in (X_0 \times \dots \times X_m \to Y)$ nous écrivons $f(x_0, \dots, x_m)$ au lieu de $f(\langle \dots \langle\langle x_0, x_1\rangle, x_2\rangle, \dots, x_m\rangle)$. $f \in (X \to Y)$ est injective (respectivement surjective, bijective) si et seulement si $\forall x, y \in \mathrm{dom}(f).[(x \neq y) \Rightarrow (f(x) \neq f(y))]$ (respectivement $\mathrm{rng}(f) = Y$, f est injective et surjective). La restriction $f \restriction X$ de la fonction f à X est $f \cap (X \times \mathrm{rng}(f))$. Si $\tau(x)$ est un terme ensembliste (désignant une classe) et $P(x)$ un prédicat alors $\langle \tau(x): P(x)\rangle$ dénote la fonction $\{\langle x, \tau(x)\rangle : P(x)\}$.

Dans la suite nous décrirons souvent les ensembles X au moyen de leurs fonctions caractéristiques également notées X telle que $X \in (\mathcal{U} \to \{tt, ff\})$, $\mathcal{U} = \{x: tt\}$ est l'univers et $\forall x.(X(x) = (x \in X))$. Les opérateurs logiques sont étendus point par point : $X \wedge Y\,(x) = X(x) \wedge Y(x)$, etc. En particulier pour les relations nous définissons $\mathrm{rel}(X, r) = [r \in (X \times X \to \{tt, ff\})]$ et écrirons indifféremment $\langle x, y\rangle \in r$, $x\,r\,y$ et $r(x, y)$. La restriction (respectivement gauche, droite) de la relation r à X est $r \restriction X\,(x, y) = [x \in X \wedge r(x, y) \wedge y \in X]$ (respectivement $r \restriction X\,(x, y) = [x \in X \wedge r(x, y)]$, $r \restriction X\,(x, y) = [r(x, y) \wedge y \in X]$). La composition de relations est $r \circ s\,(x, y) = \exists z.[r(x, z) \wedge s(z, y)]$. La relation identité est $\underline{1}(x, y) = [x = y]$. Ayant introduit l'ensemble ω des entiers naturels, nous définissons $r^0 = \underline{1}$, $r^{m+1} = r \circ r^m$, la fermeture transitive $r^+ = \exists m \in (\omega \setminus 0).r^m$ et la fermeture transitive

réflexive $r^* = \underline{1} \vee r^+$. L'inverse d'une relation est $r^{-1}(x,y) = r(y,x)$. L'image $r[X]$ de X par r est $\{y : \exists x \in X.\ (x\,r\,y)\}$. Un morphisme f de la relation r dans la relation r' est tel que $[f \in (\mathrm{fld}(r) \to \mathrm{fld}(r')) \wedge \forall a,b \in \mathrm{fld}(r).\ [a\,r\,b = (fa)\,r'\,(fb)]]$. C'est un isomorphisme si f est bijective, épimorphisme si f est surjective, monomorphisme si f est injective, un endomorphisme si $\mathrm{fld}(r) = \mathrm{fld}(r')$ et un automorphisme si c'est un endomorphisme et un isomorphisme.

r est une relation d'équivalence sur X si elle est réflexive (i.e. $\forall x \in X.\ x\,r\,x$), symétrique (i.e. $\forall x,y \in X.\ x\,r\,y \Rightarrow y\,r\,x$) et transitive (i.e. $\forall x,y,z \in X.\ (x\,r\,y \wedge y\,r\,z) \Rightarrow x\,r\,z$). r induit une partition (i.e. tout élément de X appartient à exactement un seul élément de la partition) de X en classes d'équivalence $[x]_r = \{y : y \in X \wedge x\,r\,y\}$. L'ensemble des classes d'équivalence de X pour r est appelé l'ensemble quotient de X modulo r et est noté X/r.

I.3 ORDRES

Une relation d'ordre (partiel) strict est irréflexive et transitive $\mathrm{apo}(W,\prec) = (\mathrm{rel}(W,\prec) \wedge [\forall x \in W.\ \neg(x \prec x)] \wedge [\forall x,y,z \in W.\ (x \prec y \wedge y \prec z) \Rightarrow x \prec z])$. La relation d'ordre réflexif $\preccurlyeq$ correspondant à $\prec$ est $x \preccurlyeq y = [x \prec y \vee x = y]$. Une relation d'ordre réflexif est réflexive, transitive et antisymétrique $\mathrm{rpo}(W,\preccurlyeq) = (\mathrm{rel}(W,\preccurlyeq) \wedge [\forall x \in W.\ x \preccurlyeq x] \wedge [\forall x,y,z \in W.\ (x \preccurlyeq y \wedge y \preccurlyeq z) \Rightarrow x \preccurlyeq z] \wedge [\forall x,y,z \in W.\ (x \preccurlyeq y \wedge y \preccurlyeq x) \Rightarrow x = y])$. Nous notons $\succ$ (respectivement $\succcurlyeq$) l'inverse de $\prec$ (respectivement $\preccurlyeq$) et $\mathrm{po}(W,r) = \mathrm{apo}(W,r) \vee \mathrm{rpo}(W,r)$.

Si $\mathrm{rpo}(W,\preccurlyeq)$, $X \subseteq W$ et $a \in W$ alors a est un majorant (respectivement majorant strict, minorant, minorant strict) de X pour $\preccurlyeq$ si et seulement si $\forall x \in X.\ x \preccurlyeq a$ (respectivement $\forall x \in X.\ x \prec a$, $\forall x \in X.\ a \prec x$, $\forall x \in X.\ a \preccurlyeq x$). a est le plus grand (respectivement plus petit) élément de X pour $\preccurlyeq$ si c'est un majorant (respectivement minorant) de X pour $\preccurlyeq$ et $a \in X$. a est un élément maximal (respectivement minimal) de X pour $\preccurlyeq$ si $a \in X \wedge \forall x \in X.\ \neg(a \prec x)$ (respectivement $\forall x \in X.\ \neg(x \prec a)$). a est la borne supérieure (respectivement inférieure) de X pour $\preccurlyeq$ si a est le plus petit (respectivement grand) des majorants (respectivement minorants) de X pour $\preccurlyeq$ c'est-à-dire $[\forall x \in X.\ x \preccurlyeq a \wedge \forall y \in W.\ (\forall x \in X.\ x \preccurlyeq y) \Rightarrow (a \preccurlyeq y)]$ (respectivement $[\forall x \in X.\ a \preccurlyeq x \wedge \forall y \in W.\ (\forall x \in X.\ y \preccurlyeq x) \Rightarrow (y \preccurlyeq a)]$). Si elle existe la borne supérieure ou suprémum de X pour $\preccurlyeq$ est notée $\sup(W,\preccurlyeq)X$ (ou $\sqcup X$). La borne inférieure ou infimum est notée $\inf(W,\preccurlyeq)X$ (ou $\sqcap X$). La borne supérieure (respectivement inférieure) stricte de X pour $\prec$ est notée $\sup^+(W,\prec)X$ (respectivement $\inf^+(W,\prec)X$) si elle existe. C'est le plus petit (respectivement grand) des majorants (respectivement minorants) stricts de X pour $\prec$.

Un ordre r (strict ou réflexif) sur W est linéaire ou total si et seulement si deux éléments quelconques distincts sont comparables c'est-à-dire $\mathrm{lo}(W,r) = [\mathrm{po}(W,r) \wedge \forall x,y \in W.\ [(x \neq y) \Rightarrow (x\,r\,y \vee y\,r\,x)]]$. Une relation $\prec$ sur W est bien-fondée si et seulement si toute partie non vide a un élément minimal. $\mathrm{wf}(W,\prec) = [\mathrm{rel}(W,\prec) \wedge \forall X \subseteq W.\ [X \neq 0 \Rightarrow \exists y \in X.\ (\neg \exists z \in X.\ z \prec y)]]$. Ayant admis

l'axiome du choix, ceci est équivalent à l'assertion que toute chaîne strictement décroissante est finie. Nous écrivons $\mathrm{wfi}(W, \prec, \mu)$ quand $\prec$ est une relation bien fondée sur W avec un élément minimal μ, $\mathrm{wfi}(W, \prec, \mu) = [\mathrm{wf}(W, \prec) \wedge \mu \in W \wedge \forall x \in W . \neg (x \prec \mu)]$. Une relation $\prec$ de bon-ordre sur W est linéaire et bien-fondée $\mathrm{wo}(W, \prec) = [\mathrm{lo}(W, \prec) \wedge \mathrm{wf}(W, \prec)]$.

Une *classe partiellement ordonnée* est une paire $\langle W, \prec \rangle$ telle que $\mathrm{po}(W, \prec)$. L'*addition* de classes partiellement ordonnées est définie par $\langle W_0, \prec_0 \rangle \oplus \langle W_1, \prec_1 \rangle = \langle W, \prec \rangle$ si et seulement si $W = \{\langle 0, w_0 \rangle : w_0 \in W_0\} \cup \{\langle 1, w_1 \rangle : w_1 \in W_1\}$ et $\langle i, x \rangle \prec \langle j, y \rangle$ si et seulement si $[(i = j = 0 \wedge x \prec_0 y) \vee (i < j) \vee (i = j = 1 \wedge x \prec_1 y)]$ (c'est-à-dire que dans $\langle W, \prec \rangle$ les éléments de W_0 sont ordonnés comme dans $\langle W_0, \prec_0 \rangle$, tous les membres de W_0 précèdent ceux de W_1 et les éléments de W_1 sont ordonnés comme dans $\langle W_1, \prec_1 \rangle$). La *multiplication* de classes ordonnées est définie par $\langle W_0, \prec_0 \rangle \otimes \langle W_1, \prec_1 \rangle = \langle W, \prec \rangle$ si et seulement si $W = W_0 \times W_1$ et $\prec$ est l'*ordre lexicographique droit*, $\langle x, y \rangle \prec \langle x', y' \rangle$ si et seulement si $[y \prec_1 y' \vee (y = y' \wedge x \prec_0 x')]$.

Soit $\langle W, \preccurlyeq \rangle$ une classe partiellement ordonnée. $X \subseteq W$ est *dirigé* si et seulement si $\forall x, x' \in X . \exists x'' \in X . (x \preccurlyeq x'' \wedge x' \preccurlyeq x'')$. $\langle W, \preccurlyeq \rangle$ est un cpo (*ordre partiel complet* ou encore ensemble inductif) si W est un ensemble possédant un plus petit élément et tout ensemble dirigé $X \subseteq W$ admet une borne supérieure. $\langle W, \preccurlyeq, \wedge, \vee, \bot, \top \rangle$ est un *treillis complet* si pour toute partie non vide $X \subseteq W$, sa borne supérieure $\mathrm{sup}(W, \preccurlyeq) X$ (notée $\vee X$) et sa borne inférieure $\mathrm{inf}(W, \preccurlyeq) X$ (notée $\wedge X$) existent. Ceci entraîne en particulier que W admet un plus petit élément $\bot = \mathrm{inf}(W, \preccurlyeq) W$ et un plus grand élément $\top = \mathrm{sup}(W, \preccurlyeq) W$. Un *treillis booléen complet* $\langle W, \preccurlyeq, \wedge, \vee, \bot, \top, \neg \rangle$ est un treillis distributif complémenté ($\neg x$ désigne un complément de x) complet.

Soient $\langle W, \preccurlyeq \rangle$ une classe partiellement ordonnée et $\rho \in (W \to W)$. ρ est une *rétraction* sur W s'il est *monotone* (i.e. $\forall x, y \in W . (x \preccurlyeq y \Rightarrow \rho(x) \preccurlyeq \rho(y))$) et *idempotent* (i.e. $\rho = \rho \circ \rho$ c'est-à-dire $\forall x \in W . \rho(x) = \rho(\rho(x))$). ρ est une *préfermeture supérieure* (duellement inférieure) sur W s'il est monotone idempotent et satisfait

l'*axiome de connectivité supérieure* (i.e. $\forall x \in W . \rho(\mathrm{sup}(W, \preccurlyeq)\{x, \rho(x)\}) = \rho(x)$) (duellement l'*axiome de connectivité inférieure* (i.e. $\forall x \in W . \rho(\mathrm{inf}(W, \preccurlyeq)\{x, \rho(x)\}) = \rho(x)$)). ρ est une *fermeture supérieure* (duellement *inférieure*) sur W s'il est monotone, idempotent et *extensif* (i.e. $\forall x \in W . x \preccurlyeq \rho(x)$) (duellement *réductif* (i.e. $\forall x \in W . \rho(x) \preccurlyeq x$)).

Soient $\langle W_1, \preccurlyeq_1 \rangle$ et $\langle W_2, \preccurlyeq_2 \rangle$ deux classes partiellement ordonnées et (α, γ) une paire de fonctions monotones $\alpha \in (W_1 \to W_2)$ et $\gamma \in (W_2 \to W_1)$ et telles que $\alpha \circ \gamma \preccurlyeq_1 \underline{1}$ et $\underline{1} \preccurlyeq_2 \gamma \circ \alpha$. (α, γ) est une *correspondance de Galois*.
En définissant les fonctions partielles $\partial_1 \in ((W_2 \to W_1) \to (W_1 \to W_2))$ et $\partial_2 \in ((W_1 \to W_2) \to (W_2 \to W_1))$ comme suit : $\partial_1(\gamma)(x) = \wedge\{y \in W_2 : x \preccurlyeq_1 \gamma(y)\}$, $\partial_2(\alpha)(y) = \vee\{x \in W_1 : \alpha(x) \preccurlyeq_2 y\}$, nous avons les résultats suivants : $[\alpha \circ \gamma \preccurlyeq_1 \underline{1}$ et $\underline{1} \preccurlyeq_2 \gamma \circ \alpha] \Leftrightarrow [\alpha$ est un $\vee$-morphisme complet et $\gamma = \partial_2(\alpha)] \Leftrightarrow [\forall x \in W_1, y \in W_2 . (\alpha(x) \preccurlyeq_2 y) \Leftrightarrow (x \preccurlyeq_1 \gamma(y))] \Leftrightarrow [\partial_1(\gamma) = \alpha$ et $\partial_2(\alpha) = \gamma] \Leftrightarrow [\alpha \preccurlyeq_2 \partial_1(\gamma)$ et $\partial_2(\alpha) \preccurlyeq_1 \gamma] \Leftrightarrow [\gamma$ est un $\wedge$-morphisme complet et $\alpha = \partial_1(\gamma)]$.

Si (α, γ) est une correspondance de Galois entre $\langle W_1, \preccurlyeq_1 \rangle$ et $\langle W_2, \preccurlyeq_2 \rangle$ alors nous avons $[\alpha \circ \gamma = \underline{1}] \Leftrightarrow [\alpha$ est surjective$] \Leftrightarrow [\gamma$ est injective$]$ et aussi $[\gamma \circ \alpha = \underline{1}] \Leftrightarrow [\alpha$ est injective$] \Leftrightarrow [\gamma$ est surjective$]$.

Soient $\langle W_1, \preccurlyeq_1, \wedge_1, \vee_1, \bot_1, \top_1, \neg_1 \rangle$ et $\langle W_2, \preccurlyeq_2, \wedge_2, \vee_2, \bot_2, \top_2, \neg_2 \rangle$ deux treillis booléens. (Si f est une fonction d'un treillis booléen dans un treillis booléen, nous définissons $\tilde{f}$ telle que $\tilde{f}(x) = \neg(f(\neg x))$). Alors $[(\alpha, \gamma)$ est une correspondance de Galois entre W_1 et W_2, si et seulement si $(\tilde{\gamma}, \tilde{\alpha})$ est une correspondance de Galois entre W_2 et $W_1]$ et $[f$ est surjective (respectivement injective) si et seulement si $\tilde{f}$ est surjective (respectivement injective)$]$.

I.4 ORDINAUX

Les ordinaux constituent une extension dans l'infini de l'ensemble ω des entiers naturels muni de l'ordre naturel $<$:

$0, 1, 2, \dots, \omega, \omega+1, \omega+2, \dots, \omega+\omega = \omega\times 2, \omega\times 2+1, \omega\times 2+2, \dots, \omega\times 2+\omega = \omega\times 3, \omega\times 3+1, \dots,$
$\omega\times 4, \dots, \omega\times\omega = \omega\uparrow 2, \dots, \omega\uparrow 3, \dots, \omega\uparrow\omega, \dots, \varepsilon_0 = \underbrace{\omega\uparrow\omega\uparrow\omega\dots}_{\omega \text{ fois}}, \dots, \omega_1^{c\alpha}, \dots, \omega_1, \dots, \aleph_1, \dots, \aleph_\omega \dots$

Nous utilisons la définition de Zermelo-Von Neumann des ordinaux. Une classe x est transitive si tout membre d'un membre de x est membre de x, $\mathrm{tran}(x) = [\forall y\in x, z\in y.\ z\in x] = [\cup x \subseteq x]$. x est un ordinal si et seulement si x est transitif et tout membre de x est transitif : $\mathrm{ord}(x) = [\mathrm{tran}(x) \wedge \forall z\in x.\ \mathrm{tran}(z)]$. Nous notons également $\mathrm{ord} = \{x : \mathrm{ord}(x)\}$ la classe des ordinaux et nous utilisons le plus souvent des lettres grecques $\alpha, \delta, \lambda, \dots$ pour désigner des ordinaux (α mais plutôt des lettres latines $n, m, \dots$ pour des entiers). La relation $\alpha<\beta = (\mathrm{ord}(\alpha) \wedge \mathrm{ord}(\beta) \wedge \alpha\in\beta) = (\mathrm{ord}(\alpha) \wedge \mathrm{ord}(\beta) \wedge \alpha\subset\beta)$ est une relation de bon-ordre pour ord dont l'infimum est zéro, l'ensemble vide, noté 0. Nous définissons la successeur $\mathcal{S}\alpha$ d'un ordinal α comme l'ordinal $\mathcal{S}\alpha = \alpha\cup\{\alpha\} = \alpha+1$. Nous posons comme d'habitude $1 = \mathcal{S}0 = \{0\}$, $2 = \mathcal{S}1 = \{0,1\} = \{0,\{0\}\}$, $3 = \mathcal{S}2 = \{0,1,2\}$ ... et plus généralement $\alpha = \{\beta\in\mathrm{ord} : \beta<\alpha\}$. α est un ordinal successeur si et seulement si $\mathrm{succ}(\alpha) = [\alpha\in\mathrm{ord} \wedge \exists\beta\in\mathrm{ord}.\ \alpha = \mathcal{S}\beta] = [\alpha\in\mathrm{ord} \wedge \cup\alpha<\alpha]$. Nous notons $\alpha-1$ le prédecesseur de α. Nous avons $\forall\alpha,\beta\in\mathrm{ord}.\ \neg(\alpha<\beta<\mathcal{S}\alpha)$. Si α n'est pas un ordinal successeur, c'est un ordinal limite et nous posons $\alpha-1=\alpha$. Les ordinaux limites sont caractérisés par $\mathrm{limit}(\alpha) = [\mathrm{ord}(\alpha) \wedge \forall\beta<\alpha.\ \mathcal{S}\beta<\alpha] = [\mathrm{ord}(\alpha) \wedge \cup\alpha=\alpha] = [\mathrm{ord}(\alpha) \wedge \forall\beta<\alpha.\ \exists\gamma.\ \beta<\gamma<\alpha]$. Nous posons $\alpha\leqslant\beta = [\alpha<\beta \vee \alpha=\beta] = [\alpha<\mathcal{S}\beta] = [\alpha\subseteq\beta]$. Pour tout $X\subseteq\mathrm{ord}$, $\sup X = \sup(\mathrm{ord},\leqslant)X = \cup X$ (respectivement $\sup^+ X = \sup^+(\mathrm{ord},\leqslant)X$) est le suprémum (respectivement strict) de X pour $\leqslant$. Si X n'a pas de plus grand élément alors c'est un ordinal limite et $\sup^+ X = \sup X$. Si X a un plus grand élément α alors $\sup X = \alpha$ et $\sup^+ X = \mathcal{S}\alpha$. Pour tout ensemble non vide $X\subseteq\mathrm{ord}$, $\cap X$ est le plus petit élément de X pour $\leqslant$. L'ordinal $\mathrm{nat}(x) =$

$[\exists!\lambda(\alpha) \wedge (\alpha\neq 0 \Rightarrow (\neg\mathrm{limit}(\alpha) \wedge \forall\beta\in(\alpha\vee 0).\ \neg\mathrm{limit}(\beta)))]$, $\omega = \cap\{\lambda\in\mathrm{ord} : \lambda\neq 0 \wedge \mathrm{limit}(\lambda)\}$. L'ensemble des entiers est $\mathbb{Z} = \omega\cup\{-m : 0<m\in\omega\}$ (où $-m$ est la paire $\langle 0,m\rangle$).

Si $\mathrm{wf}(W,\prec)$ alors les preuves par induction transfinie sur $\prec$ sont de la forme $[\forall x\in W.\ ((\forall\beta\prec\alpha.\ P(\beta)) \Rightarrow P(\alpha))] \Rightarrow [\forall\alpha\in W.\ P(\alpha)]$. En particulier pour les ordinaux nous avons $[P(0) \wedge \forall\alpha\in\mathrm{ord}.[P(\alpha)\Rightarrow P(\mathcal{S}\alpha)] \wedge \forall\alpha\in\mathrm{ord}.[(\mathrm{limit}(\alpha) \wedge \forall\beta<\alpha.\ P(\beta)) \Rightarrow P(\alpha)]] \Rightarrow [\forall\alpha\in\mathrm{ord}.\ P(\alpha)]$ tandis que pour les entiers nous avons $\forall\alpha\in\omega.\ [P(0) \wedge \forall\alpha\in\omega.[P(\alpha) \Rightarrow P(\alpha+1)]] \Rightarrow [\forall\alpha\in\omega.\ P(\alpha)]$.

Si $\mathrm{wf}(W,\prec)$ alors les définitions par récurrence transfinie sont de la forme $F(x_1,\dots,x_m,w) = G(x_1,\dots,x_m,w,\{\langle z, F(x_1,\dots,x_m,z)\rangle : z\in W \wedge z\prec w\})$. En particulier pour les ordinaux ces définitions prennent souvent la forme $F(x_1,\dots,x_m,0) = f(x_1,\dots,x_m)$, $F(x_1,\dots,x_m,\mathcal{S}\alpha) = G(x_1,\dots,x_m,\alpha,F(x_1,\dots,x_m,\alpha))$ et $\mathrm{limit}(\alpha) \Rightarrow [F(x_1,\dots,x_m,\alpha) = H(x_1,\dots,x_m,\alpha,\{F(x_1,\dots,x_m,\beta) : \beta<\alpha\})]$.

Nous utilisons les ordinaux comme modèles des ordres bien fondés. Si $\mathrm{wf}(W,\prec)$ alors nous définissons le rang de $x\in W$ par $\mathrm{rk}(W,\prec)(x) = \sup^+\{\mathrm{rk}(W,\prec)(y) : y\in W \wedge y\prec x\}$ et le rang de $\langle W,\prec\rangle$ par $\mathrm{rk}[W,\prec] = \sup^+\{\mathrm{rk}(W,\prec)(x) : x\in W\}$. Nous avons $\mathrm{rk}(W,\prec)\in(W\to\mathrm{rk}[W,\prec])$, $\forall x,y\in W.\ [(x\prec y) \Rightarrow \mathrm{rk}(W,\prec)(x) \prec \mathrm{rk}(W,\prec)(y)]$ et $\mathrm{rk}(W,\prec)$ est un isomorphisme de W sur $\mathrm{rk}[W,\prec]$ quand $\mathrm{wo}(W,\prec)$.

L'addition d'ordinaux est définie par $\alpha+0=\alpha$, $\alpha+\mathcal{S}\beta = \mathcal{S}(\alpha+\beta)$ et $\alpha+\gamma = \bigcup_{\delta<\gamma}(\alpha+\delta)$ quand γ est un ordinal limite non nul. Si $\mathrm{wo}(W_0,\prec_0)$ et $\mathrm{wo}(W_1,\prec_1)$ alors i_+ défini par $i_+(\langle 0,w\rangle) = \mathrm{rk}(W_0,\prec_0)(w)$, $i_+(\langle 1,w\rangle) = \mathrm{rk}[W_0,\prec_0] + \mathrm{rk}(W_1,\prec_1)(w)$ est l'unique isomorphisme de $\langle W_0,\prec_0\rangle \oplus \langle W_1,\prec_1\rangle$ sur $\mathrm{rk}[W_0,\prec_0] + \mathrm{rk}[W_1,\prec_1] = \mathrm{rk}[\langle W_0,\prec_0\rangle \oplus \langle W_1,\prec_1\rangle]$.

La multiplication d'ordinaux est définie par $\alpha\times 0 = 0$, $\alpha\times\mathcal{S}\beta = (\alpha\times\beta)+\alpha$ et $\alpha\times\gamma = \bigcup_{\delta<\gamma}(\alpha\times\delta)$ quand γ est un ordinal limite non nul. Si $\mathrm{wo}(W_0,\prec_0)$ et $\mathrm{wo}(W_1,\prec_1)$ alors $i_\times$ défini par $i_\times(\langle x,y\rangle) = (\mathrm{rk}[W_0,\prec_0]\times\mathrm{rk}(W_1,\prec_1)(y)) + \mathrm{rk}[W_1,\prec_1]$ est l'unique isomorphisme de $\langle W_0,\prec_0\rangle \otimes \langle W_1,\prec_1\rangle$ sur $\mathrm{rk}[W_0,\prec_0]\times\mathrm{rk}[W_1,\prec_1] = \mathrm{rk}[\langle W_0,\prec_0\rangle \otimes \langle W_1,\prec_1\rangle]$.

L'exponentiation d'ordinaux est définie par $\alpha\uparrow 0 = 1$, $\alpha\uparrow\beta^{+} = (\alpha\uparrow\beta)\times\alpha$ et $\alpha\uparrow\gamma = \bigcup_{\delta<\gamma} \alpha\uparrow\delta$ quand γ est un ordinal limite non nul.

I.5 SEQUENCES

Une séquence sur A est une fonction f telle que $\mathrm{dom}(f)$ est un ordinal et $\mathrm{rng}(f) = A$. Nous appelons A l'alphabet de la séquence f. La longueur $\mathrm{dom}(f)$ de la séquence f est également notée $|f|$. Nous notons $A^{\leq\lambda} = \bigcup\{(\alpha\to A) : \alpha\in(\lambda+1)\}$ (respectivement $A^{<\lambda} = \bigcup\{(\alpha\to A) : \alpha\in\lambda\}$) la classe des séquences non vides de longueur inférieure (respectivement ou égale) à λ et $A^{<\lambda} = \bigcup\{(\alpha\to A) : \alpha\in\lambda\}$ (respectivement $A^{\leq\lambda} = \bigcup\{(\alpha\to A) : \alpha\in(\lambda+1)\}$) la classe des séquences de longueur inférieure (respectivement ou égale) à λ.

Les séquences $f\in A^{<\omega}$ sont dites finies car $|f|<\omega$. La séquence vide c'est-à-dire la séquence finie de longueur 0 est 0 également notée $\langle\rangle$. Si $k\in\omega$ et $a,b,\dots,t$ sont k termes ensemblistes, alors le k-tuple $\langle a,b,\dots,t\rangle$ dénote la séquence finie $\{\langle 0,a\rangle,\langle 1,b\rangle,\dots,\langle k-1,t\rangle\}$ de longueur k. Les notions de paire ordonnée $\langle x,y\rangle = \{\{x\},\{x,y\}\}$ et de séquence de longueur 2 $\langle\alpha,\gamma\rangle = \{\langle 0,x\rangle,\langle 1,y\rangle\}$ sont confondues parceque nous n'utilisons que leur propriété commune $(\langle x,y\rangle = \langle u,v\rangle) \Rightarrow (x=u \wedge y=v)$. Les séquences infinies de longueur ω sont notées $\langle f_i : i\in\omega\rangle$ ou $\langle f_0,\dots,f_i,\dots\rangle$ tandis que les séquences transfinies de longueur $\lambda>\omega$ sont notées $\langle f_i : i\in\lambda\rangle$ ou $\langle f_0,\dots,f_i,\dots\rangle_{i\in\lambda}$.

Le préfixe $f^{<\pi}$ (respectivement $f^{\leq\pi}$) d'une séquence $f\in(\alpha\to A)$ est la séquence f quand $\pi\geq\alpha$ (respectivement $\pi+1\geq\alpha$) sinon c'est la séquence $f'\in(\pi\to A)$ telle que $\forall i\in\pi.\ f'_i = f_i$ (respectivement $f'\in(\pi+1\to A)$ telle que $\forall i\in(\pi+1).\ f'_i = f_i$). Le suffixe $f^{>\pi}$ (respectivement $f^{\geq\pi}$) d'une séquence $f\in(\alpha\to A)$ est la séquence vide si $\pi+1\geq\alpha$ (respectivement $\pi\geq\alpha$) sinon $\pi+1<\alpha$ (respectivement $\pi<\alpha$) et c'est la séquence $f'\in(\beta\to A)$ telle que β est l'unique ordinal pour lequel $(\pi+1)+\beta=\alpha$ que nous notons $\beta=\alpha-(\pi+1)$ et $\forall i\in\beta.\ f'_i = f_{(\pi+1)+i}$ (respectivement $\pi+\beta=\alpha$ que nous notons $\beta=\alpha-\pi$ et $\forall i\in\beta.\ f'_i = f_{\pi+i}$). La tranche $f^{<\pi,\beta>}$ (respectivement $f^{\leq\pi,\beta>}$, $f^{<\pi,\beta\geq}$ et $f^{\leq\pi,\beta\geq}$) est $(f^{<\pi})^{>\alpha}$ (respectivement $(f^{\leq\pi})^{>\alpha}$, $(f^{<\pi})^{\geq\alpha}$, $(f^{\leq\pi})^{\geq\alpha}$).

Nous désignons l'opération de concaténation $^\frown$ sur $A^{<\omega}$ par $f^\frown g = f$ si $|f|=\omega$ sinon $f^\frown g = f\cup\{\langle |f|+i, g(i)\rangle : i\in|g|\}$.

I.6 CARDINAUX

X est équipotent avec Y noté $X \approx Y$ si et seulement si il existe une bijection entre X et Y. $\underline{m}$ est un cardinal si et seulement si $[\underline{m} \in \underline{ord} \wedge \forall \beta \in \underline{m}. \neg(\beta \approx \underline{m})]$. Nous notons $\underline{card}(X)$ le cardinal de l'ensemble X c'est-à-dire l'unique cardinal $\underline{m}$ équipotent à X. Pour tout ordinal α, α^+ est le plus petit cardinal strictement supérieur à α. Un ensemble est fini si $\underline{card}(X) < \omega$, dénombrable si $\underline{card}(X) \leq \omega$, infini si $card(X) \geq \omega$.

Un cardinal $\underline{m}$ est dit régulier si $\forall \Gamma \subseteq \underline{m}$, si $\underline{card}(\Gamma) < \underline{m}$ alors $\cup \Gamma < \underline{m}$, sinon il est dit singulier.

$\underline{m}^+ = \omega$ si $\underline{m} < \omega$, $\underline{m}^+ = \underline{m}$ si $\underline{m}$ est un cardinal infini régulier et $\underline{m}^+ = \underline{m}^+$ si $\underline{m}$ est un cardinal infini singulier. (Pour tout cardinal $\underline{m}$, m^+ est régulier (puisque ω est régulier et supposant l'axiome du choix, pour tout cardinal infini $\underline{m}$, m^+ est régulier)).

ANNEXE II :

INDEX DES NOTATIONS MATHEMATIQUES

ANNEXE II :

INDEX DES NOTATIONS MATHEMATIQUES

ANNEXE II :

INDEX DES NOTATIONS MATHEMATIQUES

Les notations ci-dessous sont introduites et définies dans l'annexe I.

LOGIQUE

$=$	égalité
$\neq$	différent
$\neg$	non logique
$\Rightarrow$	implication logique
$\Leftarrow$	implication logique inverse
$\not\Rightarrow$	négation de l'implication logique
$\not\Leftarrow$	négation de l'implication logique inverse
$\Leftrightarrow$	équivalence logique
$\vee$	ou (inclusif) logique
$\wedge$	et logique
$\mathbb{F}$	valeur de vérité fausse
$\mathbb{T}$	valeur de vérité vraie
$\mathbb{U}$	univers ($\mathbb{U} = \{x : \#\}$)
$x, x', x_i, x_{i,m}, z, z', z_i, z_{i,m}$	variables logiques
P, Q, R	Prédicats
$P(x_0, \ldots, x_i, \ldots)$	un prédicat avec $x_0, \ldots, x_i \ldots$ comme (seules) variables libres possibles
$(P \Rightarrow Q \mid R)$	si P alors Q sinon R, $((P \wedge Q) \vee (\neg P \wedge R))$
$(P \to a \mid b)$	désigne la valeur a si P est vrai, sinon b
$\forall$	pour tout
$\forall x_0, \ldots x_i \ldots P$	récursivation de $\forall x_0 . (\ldots (\forall x_i . (\ldots P \ldots)) \ldots)$
$\forall x_0 \in X_0, \ldots \forall x_i \in X_i \ldots P$	abbréviation de $\forall x_0, \ldots x_i \ldots ((x_0 \in X_0 \wedge \ldots \wedge x_i \in X_i \wedge \ldots) \Rightarrow P)$
$\wedge \ldots \wedge$	

$\exists!$	il existe un unique
$\exists x_0,\dots,x_i,\dots . P$	abbréviation de $\exists x_0.(\dots(\exists x_i.(\dots . P\dots))\dots)$
$\exists x_0\in X_0,\dots,x_i\in X_i,\dots . P$	abbréviation de $\exists x_0,\dots,x_i,\dots .(x_0\in X_0\wedge\dots\wedge x_i\in X_i\wedge\dots\wedge P)$

ENSEMBLES

$\in$	est membre de
$\notin$	n'est pas membre de
$\cup$	
$X\cup Y$	union binaire
$\cup X$	union infinie de tous les membres de X
$\bigcup_{i\in I} A_i$	$=\cup\,\mathrm{rng}(A)$ où $A\in(I\to\mathrm{rng}(A))$
$\cap$	
$X\cap Y$	intersection binaire
$\cap X$	intersection infinie de tous les membres de X
$\bigcap_{i\in I} A_i$	$=\cap\,\mathrm{rng}(A)$ où $A\in(I\to\mathrm{rng}(A))$
$\subseteq$	inclusion large
$\subset$	inclusion stricte
$\supseteq$	$X\supseteq Y \iff Y\subseteq X$
$\supset$	$X\supset Y \iff Y\subset X$
$\nsubseteq$	$X\nsubseteq Y \iff \neg(X\subseteq Y)$
$\not\subset$	$X\not\subset Y \iff \neg(X\subset Y)$
$\nsupseteq$	$X\nsupseteq Y \iff \neg(X\supseteq Y)$
$\not\supset$	$X\not\supset Y \iff \neg(X\supset Y)$
$0,\varnothing$	zéro ou ensemble vide
$\setminus$	différence de classes
$\sim$	différence avec un singleton, $X\sim x = X\setminus\{x\}$
$\times$	produit cartésien
$.$	tel que (dans $\exists x.P(x)$), on a (dans $\forall x.P(x)$)
$:$	tel que (dans $\{x:P(x)\}$)
$\wp X$	puissance de X
$X/_r$	ensemble quotient de X modulo r
$[x]_r$	classe d'équivalence de x pour r
(x,y)	paire ordonnée

$\forall x.\, P(x)$	pour tout x, on a $P(x)$
$\exists x.\, P(x)$	il existe x tel que $P(x)$
$\{x : P(x)\}$	la classe de tous les x tels que $P(x)$
$\{r(x_0,\dots,x_i,\dots) : P(x_0,\dots,x_i,\dots)\}$	la classe de tous les $r(x_0,\dots,x_i,\dots)$ tels que $P(x_0,\dots,x_i,\dots)$
$\{x \in X : P\}$	abbréviation de $\{x : x \in X \wedge P\}$
$\underline{it}$	l'unique isomorphisme de l'addition $\langle W_0, \prec_0\rangle \oplus \langle W_1, \prec_1\rangle$ de classes bien fondées sur $\underline{rk}[W_0,\prec_0] + \underline{rk}[W_1,\prec_1]$
$\underline{ix}$	l'unique isomorphisme de la multiplication $\langle W_0, \prec_0\rangle \otimes \langle W_1, \prec_1\rangle$ de classes bien fondées sur $\underline{rk}[W_0,\prec_0] \times \underline{rk}[W_1,\prec_1]$
$\underline{set}(X)$	X est un ensemble
$\underline{tran}(X)$	X est une classe transitive, $[\forall y \in X, x \in y.\, x \in X]$

RELATIONS

$\neg$	négation de relation $(\neg r)(x,y) = \neg r(x,y)$
$\Rightarrow$	implication de relations $(r \Rightarrow r')(x,y) = (r(x,y) \Rightarrow r'(x,y))$
$\vee$	union de relations $(r \vee r')(x,y) = r(x,y) \vee r'(x,y)$
$\wedge$	intersection de relations $(r \wedge r')(x,y) = r(x,y) \wedge r'(x,y)$
$\underline{1}$	relation identité
$x\, r\, y$	x est en relation avec y selon la relation binaire r
$r(x,y)$	"
$\langle x,y\rangle \in r$	"
$r \upharpoonright X$	restriction gauche de la relation r, $r \cap (X \times \underline{rng}(r))$
$r \upharpoonright X$	restriction droite de la relation r, $r \cap (\underline{dom}(r) \times X)$
$r \upharpoonright X$	restriction de la relation r, $r \cap (X \times X)$
$r \underline{\circ} s$	composition de relations, $r \circ s\,(x,y) = \exists y.\, [r(x,y) \wedge s(y,y)]$
r^0	relation identité, $\underline{1}$
r^m	$r \underline{\circ} r \underline{\circ} \dots \underline{\circ} r$, m fois
r^+	fermeture transitive d'une relation, $\exists m \in (\omega \setminus 0).\, r^m$
r^*	fermeture transitive reflexive, $\exists m \in \omega.\, r^m$
r^{-1}	inverse de la relation r, $x\, r^{-1}\, y = y\, r\, x$
$r[X]$	image de X par r, $\{y : \exists x \in X.\, r(x,y)\}$
$\underline{dom}(r)$	domaine de la relation r, $\{x : \exists y.\, x\, r\, y\}$
$\underline{fld}(r)$	champ de la relation r, $\underline{dom}(r) \cup \underline{rng}(r)$
$\underline{rng}(r)$	co-domaine de la relation r, $\{y.\, \exists x.\, x\, r\, y\}$
$\underline{rel}(X,r)$	r est une relation sur X, $r \in (X \times X \to \{tt, ff\})$

FONCTIONS

$\circ$	composition fonctionnelle, $f \circ g(x) = f(g(x))$
$(X \to Y)$	classe des fonctions partielles de X dans Y
$(X \to Y)$	classe des fonctions totales de X dans Y
$\langle \tau(x) : x \in I \rangle$	la fonction f sur I telle que $\forall x.\ f(x) = \tau(x)$
$f \restriction X$	restriction de f à X, $f \cap (X \times \mathrm{rng}(f))$
$f(x)$	notation fonctionnelle de y tel que $\langle x,y \rangle \in f$, s'il existe
f_x	"
f_x	"
$f(x_0, \dots, x_{m-1})$	$f(\langle x_0, \dots, x_{m-1} \rangle)$
$f[x\|y]$	la fonction f' telle que $f'(x) = y$ et $f'(z) = f(z)$ si $z \neq x$

ORDRES

$\prec$	une relation d'ordre strict
$\preccurlyeq$	la relation d'ordre reflexif correspondant à $\prec$
$\succ$	la relation inverse de $\prec$
$\succcurlyeq$	la relation inverse de $\preccurlyeq$
$\oplus$	addition de classes partiellement ordonnées
$\otimes$	multiplication de classes partiellement ordonnées
$\bot$	infimum d'un treillis complet
$\top$	supremum d'un treillis complet
$\langle W, \preccurlyeq \rangle$	classe partiellement ordonnée
$\langle W, \preccurlyeq, \wedge, \vee, \bot, \top \rangle$	treillis complet
$\langle W, \preccurlyeq, \wedge, \vee, \bot, \top, \neg \rangle$	treillis booléen complet
(α, β)	demi-correspondance, quasi-correspondance ou correspondance de Galois
$\underset{\sim}{cpo}$	ordre partiel complet
$\underset{\sim}{lo}(W, r)$	la relation r est un ordre linéaire ou total sur W
$\underset{\sim}{po}(W, r)$	la relation r est un ordre partiel (strict ou reflexif) sur W
$\underset{\sim}{rk}[W, \prec]$	rang de $\langle W, \prec \rangle$ pour la relation bien fondée $\prec$ sur W, $\underset{\sim}{sup}^+\{\underset{\sim}{rk}(W,\prec)(x) : x \in W\}$
$\underset{\sim}{rk}(W, \prec)(x)$	rang de x pour la relation bien fondée $\prec$ sur W, $\underset{\sim}{sup}^+\{\underset{\sim}{rk}(W,\prec)(y) : y \in W \wedge y \prec x\}$
$\underset{\sim}{rpo}(W, \preccurlyeq)$	la relation $\preccurlyeq$ est un ordre partiel reflexif sur W
$\underset{\sim}{spo}(W, \prec)$	la relation $\prec$ est un ordre partiel strict sur W
$\underset{\sim}{sup}(W, \preccurlyeq)X$	borne supérieure de X pour $\preccurlyeq$ sur W
$\underset{\sim}{sup}^+(W, \preccurlyeq)X$	borne supérieure stricte de X pour $\preccurlyeq$ sur W
$\underset{\sim}{wf}(W, \prec)$	la relation $\prec$ est bien fondée sur W, $[\mathrm{rel}(W,\prec) \wedge \forall X \subseteq W.\ [X \neq 0 \Rightarrow \exists y \in X.\ (\forall z \in X.\ \neg z \prec y)]]$
$\underset{\sim}{wf}(N, \prec, \mu)$	la relation $\prec$ est bien fondée sur W avec un element minimal μ, $[\omega f(W,\prec) \wedge \mu \in W \wedge \forall z \in W.\ \neg(z \prec \mu)]$
$\underset{\sim}{wo}(W, \prec)$	la relation $\prec$ est un bon-ordre sur W, $[\underset{\sim}{lo}(W,\prec) \wedge \underset{\sim}{wf}(W,\prec)]$

ORDINAUX

$<$	inférieur strict sur les ordinaux, $\alpha<\beta = (\underline{ord}(\alpha) \wedge \underline{ord}(\beta) \wedge \alpha\in\beta)$
$\leqslant$	inférieur ou égal sur les ordinaux, $\alpha\leqslant\beta = (\alpha<\beta)\vee(\alpha=\beta)$
$>$	inverse de $<$
$\geqslant$	inverse de $\leqslant$
$\not<$	négation de $<$
$\not\leqslant$	négation de $\leqslant$
$\not>$	négation de $>$
$\not\geqslant$	négation de $\geqslant$
1	$=\underline{\mathcal{P}}0 = \{0\}$
2	$=\underline{\mathcal{P}}1 = \{0,1\} = \{0,\{0\}\}$
3	$=\underline{\mathcal{P}}2 = \{0,1,2\} = \{0,\{0\},\{0,\{0\}\}\}$
...	...
$\alpha-1$	ordinal predecesseur de α, c'est α si $\underline{limit}(\alpha)$ sinon β tel que $\underline{\mathcal{P}}\beta=\alpha$
$\underline{\mathcal{P}}\alpha$	ordinal successeur de α, $\underline{\mathcal{P}}\alpha = \alpha\cup\{\alpha\} = \alpha+1$
$\cup X$	supremum de $X\subseteq\underline{ord}$ pour $\leqslant$
$\cap X$	plus petit élément de $X\subseteq\underline{ord}$ pour $\leqslant$ quand $X\neq 0$
ω	ensemble des entiers naturels
$+,\dot{+}$	addition d'ordinaux, $\alpha+\beta = \alpha\cup\underline{sup}\{(\alpha+\gamma)+1 : \gamma<\beta\}$
$\times,\dot{\times}$	multiplication d'ordinaux, $\alpha\times\beta = \underline{sup}\{(\alpha\times\gamma)+\alpha : \gamma<\beta\}$
$\uparrow$	exponentiation d'ordinaux
$\underline{limit}(\alpha)$	caractérise un ordinal limite
$\underline{limit}$	$=\{\alpha : \underline{limit}(\alpha)\}$
$\underline{nat}(\alpha)$	caractérise les entiers naturels
$\underline{ord}(\alpha)$	caractérise les ordinaux, $[\underline{trans}(\alpha)\wedge\forall\beta\in\alpha.\ \underline{trans}(\beta)]$
$\underline{ord}$	classe des ordinaux, $\{\alpha : \underline{ord}(\alpha)\}$
$\underline{succ}(\alpha)$	caractérise un ordinal successeur, $[\underline{ord}(\alpha)\wedge\exists\beta.\alpha=\underline{\mathcal{P}}\beta]$
$\underline{succ}$	$=\{\alpha : \underline{succ}(\alpha)\}$
$\underline{sup}\,X$	supremum d'une classe d'ordinaux, $\underline{sup}(\underline{ord},\leqslant)X = \cup X$

SEQUENCES

$\langle\rangle$	séquence vide 0
$\langle f_0,\dots,f_{n-1}\rangle$	séquence finie de longueur n
$\langle f_0,\dots,f_i,\dots\rangle$	séquence infinie de longueur ω
$\langle f_i : i\in\lambda\rangle$	séquence transfinie de longueur λ
$\langle f_0,\dots,f_i,\dots\rangle_{i\in\lambda}$	"
$\lvert f\rvert$	longueur de la séquence f, $\lvert f\rvert = \underline{dom}(f)$
$\hat{}$	concaténation de séquences
$A^{<\lambda}$	classe des séquences non vides sur A de longueur inférieure à λ, $\cup\{(\alpha\to A) : \alpha\in(\lambda\setminus 0)\}$
$A^{\leqslant\lambda}$	classe des séquences non vides sur A de longueur inférieure ou égale à λ, $\cup\{(\alpha\to A) : \alpha\in(\lambda+1\setminus 0)\}$
$A^{\leqslant\lambda}$	classe des séquences sur A de longueur inférieure à λ, $\cup\{(\alpha\to A) : \alpha\in\lambda\}$
$A^{\leqslant\lambda}$	classe des séquences sur A de longueur inférieure ou égale à λ, $\cup\{(\alpha\to A) : \alpha\in\lambda+1\}$
$f^{<\pi}$	préfixe $\langle f_0,\dots,f_{\pi-1}\rangle$ d'une séquence f (si $\pi<\lvert f\rvert$ sinon f)
$f^{\leqslant\pi}$	préfixe $\langle f_0,\dots,f_\pi\rangle$ d'une séquence f (si $\pi+1<\lvert f\rvert$ sinon f)
$f^{>\pi}$	suffixe $\langle f_{\pi+1},\dots\rangle$ d'une séquence f (si $\pi+1<\lvert f\rvert$ sinon $\langle\rangle$)
$f^{\geqslant\pi}$	suffixe $\langle f_\pi,\dots\rangle$ d'une séquence f (si $\pi<\lvert f\rvert$ sinon $\langle\rangle$)
$f^{<\alpha,\beta>}$	tranche d'une séquence f, $(f^{<\beta})^{>\alpha}$
$f^{\leqslant\alpha,\beta\geqslant}$	tranche d'une séquence f, $(f^{\leqslant\beta})^{\geqslant\alpha}$
$f^{\leqslant\alpha,\beta>}$	tranche d'une séquence f, $(f^{<\beta})^{\geqslant\alpha}$
$f^{<\alpha,\beta\geqslant}$	tranche d'une séquence f, $(f^{\leqslant\beta})^{>\alpha}$

CARDINAUX

$\approx$	équipotence
α^+	le plus petit cardinal strictement supérieur à α
$\underline{m}^+$	$\underline{m}^+ = \omega$ si $\underline{m} < \omega$, $\underline{m}^+ = \underline{m}$ si $\underline{m}$ est un cardinal infini régulier, $\underline{m}^+ = \underline{m}^+$ si $\underline{m}$ est un cardinal infini singulier
$\underset{\sim}{\mathrm{card}}(X)$	cardinal de X, unique cardinal équipotent à X

ANNEXE III :

INDEX DES NOTATIONS INFORMATIQUES

ANNEXE III :

INDEX DES NOTATIONS INFORMATIQUES

REFERENCES ET NOTATIONS TYPOGRAPHIQUES

AUTRES NOTATIONS

ANNEXE III :

INDEX DES NOTATIONS INFORMATIQUES

Les notations informatiques sont introduites par chapitre puis classées dans chaque chapitre comme suit : symbole, ordre alphabétique des lettres grecques, ordre alphabétique des lettres latines. Le numéro de paragraphe qui suit chaque notation, représente le paragraphe où elle a été introduite.

REFERENCES ET NOTATIONS TYPOGRAPHIQUES

$n_0 \cdot \ldots \cdot n_{k-1} \cdot n_k$	paragraphe n_k du paragraphe $n_0 \cdot \ldots \cdot n_{k-1}$
$n_0 \cdot \ldots \cdot n_k : m$	définition m du paragraphe $n_0 \cdot \ldots \cdot n_k$
$n_0 \cdot \ldots \cdot n_k \sim m$	Théorème, lemme ou corollaire m du paragraphe $n_0 \cdot \ldots \cdot n_k$
$n_0 \cdot \ldots \cdot n_k - m$	exemple m du paragraphe $n_0 \cdot \ldots \cdot n_k$
□	fin d'une démonstration de théorème ou de lemme, d'un exemple.
x [yy]	référence bibliographique à l'auteur (ou aux auteurs) x et l'année 19yy

AUTRES NOTATIONS

?	affectation aléatoire (V:=?), 2.8.1.1
	reception d'un message sur rendez-vous (Ch?V), 2.8.2.1
!	envoi d'un message sur rendez-vous (Ch!E), 2.8.3.1
|	alternative syntaxique, 2.8
=	identité syntaxique, 2.8.1.1
	relation d'équivalence entre systèmes de transitions 2.5.3

$:=$	affectation à une variable de programme, 2.8.1.1
$:$	suit une étiquette de programme, 2.8.1.1
$;$	composition séquentielle de commandes, 2.8.1.1
$\wedge$	relation de concaténation (de traces), 2.1.1
$\xrightarrow{a}$	relation de concaténation (de traces) via l'action a, 2.1.1
$\rightarrow$	dérivation syntaxique, 2.8
$\rightarrowtail$	relation de préfixe entre traces, 2.6.1
$\twoheadrightarrow$	relation de suffixe entre traces, 2.6.2
$[\![\_ \Vert \_ \Vert \_]\!]$	composition parallèle de processus séquentiels, 2.8.2
ε	caractérise les états initiaux, $(\varepsilon \in (S \to \{tt,ff\}))$, 2.2
$\varepsilon[\![Pr]\!]$	caractérise les états initiaux du programme Pr, 2.8.1.2.3
$\varepsilon\langle S,A,\Sigma\rangle$	caractérise les états initiaux engendrés par la sémantique $\langle S,A,\Sigma\rangle$ $(\varepsilon(\Delta) = [\exists p \in \Sigma.\ p_0 = \Delta])$, 2.3
$\varepsilon\ell[\![Pps]\!]$	caractérise les états initiaux associés au programme asynchrone Pps par la sémantique libérale, 2.8.5.3
$\sigma\alpha(p_i,p_j)$	nombre de fois qu'une action appartenant à α est exécutée entre p_0 et p_j, $(\sigma \in (2^A \to (\Sigma \times \omega \to \omega)))$, 2.8.5.2.6
Σ	ensemble de traces, 2.1.1
$\Sigma[\![Pr]\!]$	ensemble de traces associé au programme Pr, 2.1.2
$\Sigma\langle S,A\rangle$	ensemble des traces sur un ensemble S d'états et un ensemble A d'actions, (abbréviation de $\Sigma^{\leq\omega}\langle S,A\rangle$), 2.1.1
$\Sigma\langle\mathcal{S},\mathcal{A}\rangle$	ensemble des traces sur les ensembles $\mathcal{S}$ d'états et $\mathcal{A}$ d'actions, 2.5.1
$\Sigma^n\langle S,A\rangle$	ensemble des traces de longueur n sur S et A, $\{\langle n,A,\alpha\rangle : n \in (\omega+1) \wedge \Delta \in (n \to S) \wedge \alpha \in (n-1 \to A)\}$, 2.1.1
$\Sigma^{<\ell}\langle S,A\rangle$	ensemble des traces sur S et A de longueur strictement inférieure à ℓ, $(\bigcup_{n \in (\ell \cap \omega)} \Sigma^n\langle S,A\rangle)$, 2.1.1
$\Sigma^{\leq\ell}\langle S,A\rangle$	ensemble des traces sur S et A de longueur inférieure ou égale à ℓ, $(\Sigma^{<\ell}\langle S,A\rangle \cup \Sigma^{\ell}\langle S,A\rangle)$, 2.1.1

$\Sigma^{\omega}\langle S,A\rangle$	ensemble des traces infinies sur S et A, 2.1.1
$\Sigma^{<\omega}\langle S,A\rangle$	ensemble des traces finies sur S et A, 2.1.1
$\Sigma^{\leq\omega}\langle S,A\rangle$	ensemble des traces sur S et A, 2.1.1
$\Sigma\langle S,A,t,\varepsilon\rangle$	ensemble des traces complètes engendrées par le système de transition $\langle S,A,t,\varepsilon\rangle$, (abbréviation de $\bigcup_{n \in ((\omega+1) \cap \omega)} \Sigma^n\langle S,A,t,\varepsilon\rangle$), 2.4
$\Sigma^n\langle S,A,t,\varepsilon\rangle$	ensemble des traces complètes finies de longueur $n \in (\omega \cap \omega)$ engendrées par le système de transition $\langle S,A,t,\varepsilon\rangle$, $(\{p \in \Sigma^n\langle S,A\rangle : \varepsilon(p_0) \wedge \forall i \in \lvert p \rvert . t_{p_i}(p_i,p_{i+1}) \wedge \forall a \in A, \Delta \in S . \neg t_a(p_{n-1},\Delta)\})$, 2.4
$\Sigma^{\omega}\langle S,A,t,\varepsilon\rangle$	ensemble des traces infinies engendrées par le système de transition $\langle S,A,t,\varepsilon\rangle$, $(\{p \in \Sigma^{\omega}\langle S,A\rangle : \varepsilon(p_0) \wedge \forall i \in \omega . t_{p_i}(p_i,p_{i+1})\})$, 2.4
$\Sigma^{<\ell}\langle S,A,t,\varepsilon\rangle$	ensemble des traces de longueur strictement inférieure à ℓ engendrées par le système de transition $\langle S,A,t,\varepsilon\rangle$, $(\bigcup_{n \in (\ell \cap \omega)} \Sigma^n\langle S,A,t,\varepsilon\rangle)$, 2.4
$\Sigma^{<\omega}\langle S,A,t,\varepsilon\rangle$	ensemble des traces finies engendrées par le système de transition $\langle S,A,t,\varepsilon\rangle$, 2.4
$\Sigma^{\leq\ell}\langle S,A,t,\varepsilon\rangle$	ensemble des traces de longueur inférieure ou égale à ℓ engendrées par le système de transition $\langle S,A,t,\varepsilon\rangle$, $(\Sigma^{<\ell}\langle S,A,t,\varepsilon\rangle \cup \Sigma^{\ell}\langle S,A,t,\varepsilon\rangle)$, 2.4
$\Sigma^{\leq\omega}\langle S,A,t,\varepsilon\rangle$	ensemble des traces complètes engendrées par le système de transition $\langle S,A,t,\varepsilon\rangle$, (noté aussi $\Sigma\langle S,A,t,\varepsilon\rangle$), 2.4
$\Sigma\ell[\![Pps]\!]$	ensemble de traces associé au programme synchrone Pps par la sémantique libérale, 2.8.5.2
a	action (commande, processus, ...), 2.1.1
$\underline{a}$	action unique, 2.5.3.3
A	ensemble d'actions, 2.1.1
$\mathcal{A}$	ensemble des actions, 2.1.2
$\underline{\text{if}}\ \mathcal{B}\ \underline{\text{then}}\ \mathcal{F}$	commande alternative, $(\mathcal{B}; Ch.^{\ell}\mathcal{G}\ \underline{\text{then}}\ \mathcal{F}_{\mathcal{G}} \mid \mathcal{B}; Ch.^{\ell}\mathcal{J}\ \underline{\text{then}}\ \mathcal{F}_{\mathcal{J}})$, 2.8.3.1

$A[\![Pr]\!]$	ensemble d'actions associé au programme Pr, 2.1.2
$\langle A,\Sigma\rangle$	sémantique concordante à $\langle S,A,\Sigma\rangle$ à l'annulation des états près, 2.5.3.2
$Al[\![Pps]\!]$	ensemble d'actions associé au programme synchrone par la sémantique libérale, 2.8.5.3
$Ar[\![Pps]\!]$	ensemble réduit d'actions associé au programme synchrone Pps, 2.8.5.2.5
$Acc\langle S,A,t,\epsilon\rangle$	caractérise les états accessibles du système de transition $\langle S,A,t,\epsilon\rangle$, 2.5.2
B	expression booléenne d'un programme, 2.8.1.2.4
$\mathcal{B}$	ensemble des expressions booléennes, 2.8.1.1
$\mathbb{B}$	sémantique des expressions booléennes, $\mathbb{B}\in(\Sigma\to((V\to\mathcal{D})\to\{tt,ff\}))$, 2.8.1.2.4
$\mathbb{B}[\![B]\!](M)$	valeur de l'expression booléenne B dans l'état mémoire M, 2.8.1.2.4
$Blo\langle S,A,t,\epsilon\rangle$	caractérise les états de blocage du système de transition $\langle S,A,t,\epsilon\rangle$, 2.5.1
$C[\![Pps]\!]$	ensemble des états de contrôle associé au programme synchrone Pps, 2.8.5.2.1
Ch	un canal de communication, 2.8.3.2.2
$\underline{Ch}$	action correspondant à une communication sur le canal ch, 2.8.3.2.2
$\mathcal{C}$	ensemble des commandes séquentielles, $(\mathcal{C}\to skip \mid V:=\mathcal{E} \mid V:=? \mid if\ \mathcal{B}\ then\ P_2\ else\ P_2\ fi \mid while\ \mathcal{B}\ do\ P_2\ od)$, 2.8.1.1
$\mathcal{C}_a$	ensemble des commandes des processus parallèles asynchrones, $(\mathcal{C}_a\to skip \mid V:=\mathcal{E} \mid V:=? \mid if\ \mathcal{B}\ then\ P_{ca}\ else\ P_{ca}\ fi \mid while\ \mathcal{B}\ do\ P_{ca}\ od \mid \langle P_{ca}\rangle)$, 2.8.1.1
$\mathcal{C}_c$	ensemble des commandes des processus parallèles communicants $(\mathcal{C}_c\to skip \mid V:=\mathcal{E} \mid V:=? \mid if\ \mathcal{B}\ then\ P_{cc}\ else\ P_{cc}\ fi \mid while\ \mathcal{B}\ do\ P_{cc}\ od \mid \langle P_{ca}\rangle \mid Ch!\mathcal{E} \mid Ch?V \mid select\ alt_1\ or \dots or\ alt_n\ es)$, 2.8.3.1
$\mathcal{C}_h$	ensemble des canaux de communication, 2.8.3.1
$\mathcal{C}_s$	ensemble des commandes des processus parallèles séquentiels $(\mathcal{C}_s\to skip \mid V:=\mathcal{E} \mid V:=? \mid if\ \mathcal{B}\ then\ P_{cs}\ else\ P_{cs}\ fi \mid while\ \mathcal{B}\ do\ P_{cs}\ od \mid P(\mathcal{E}) \mid \dots)$, 2.8.5.1

$cond$	$cond[\![P_s]\!](L,L')(M)$ est la condition sur l'état mémoire M pour que le contrôle passe de L à L' dans l'exécution d'un pas de P_s, 2.8.1.2.4
$\mathcal{D}$	domaine des valeurs des variables des programmes, 2.8.1.2.1
E	expression d'un programme, 2.8.1.2.4
$\mathbb{E}$	sémantique des expressions, $(\mathbb{E}\in(\mathcal{E}\to((V\to\mathcal{D})\to\mathcal{D})))$, 2.8.1.2.4
$\mathbb{E}[\![E]\!](M)$	valeur de l'expression E dans l'état mémoire M, 2.8.1.2.4
$\mathcal{E}$	ensemble des expressions, 2.8.1.1
$Enabled(a,i,p,\Sigma)$	l'action a est activable au point $i\in\lvert p\rvert$ d'une trace p de Σ, $([i\in\lvert p\rvert \wedge \exists q\in\Sigma.(i\in\lvert q\rvert \wedge q^{<i}=p^{<i} \wedge q_i=a)])$, 2.6.4
$\underline{Efus}$	extension par fusions, 2.6.5
$Efus(\langle S,A,\Sigma\rangle)$	extension par fusions de la sémantique $\langle S,A,\Sigma\rangle$, $(\langle S,A,\{p^{\wedge}q : p\in\Sigma^{<\omega}\langle S,A\rangle \wedge q\in\Sigma^{\leq\omega}\langle S,A\rangle \wedge \exists p',q'\in\Sigma.(p\twoheadrightarrow p' \wedge q\twoheadrightarrow q')\}\rangle)$, 2.6.5
$\simeq\langle f_s\rangle$	concordance à une fonction f_s des états près, 2.5.3.1
$\simeq\langle f_s\rangle(\langle S,A,\Sigma\rangle)$	sémantique concordante à $\langle S,A,\Sigma\rangle$ à la fonction f_s des états près, $(\langle f_s[S],A,\{\langle m,f_s(A),a\rangle : \langle m,A,a\rangle\in\Sigma\}\rangle)$, 2.5.3.1
$Ffus$	fermeture par fusions, 2.6.5
$Ffus(\langle S,A,\Sigma\rangle)$	fermeture par fusions de la sémantique $\langle S,A,\Sigma\rangle$, $(\bigcup_{m\geq 0} Efus^m)$, 2.6.5
$Flim$	fermeture par limites, 2.6.2
$Flim(\langle S,A,\Sigma\rangle)$	fermeture par limites de la sémantique $\langle S,A,\Sigma\rangle$, $(\langle S,A,\Sigma\cup\{p\in\Sigma^{\omega}\langle\cdot,A\rangle : \forall n\in\omega.\exists q\in\Sigma.\ p^{\leq n}\twoheadrightarrow q\}\rangle)$, 2.6.4
$f[x\leftarrow y]$	substitution syntaxique, (f où y est substitué à x), 4.3.2.1.4.2
$if\dots then\dots else\dots fi$	composition alternative de commandes, 2.8.1
$if\dots then\dots fi$	"
$Isem(Se)$	valeur initiale du sémaphore Se, $(Isem\in(\mathcal{V}_e\to\mathcal{D}))$, 2.8.5.1

L	étiquette d'un programme, 2.8.1.1
$\mathcal{L}$	ensemble des étiquettes, 2.8.1.1
$\mathcal{LP}$	langage de programmation, 2.1
M	un état mémoire, $(M \in \mathcal{M})$, 2.8.1.2.1
$\mathcal{M}$	ensemble des états mémoires, $(\mathcal{M} = (\mathcal{V} \to \mathcal{D}))$, 2.8.1.2.1
$\langle m, A, a \rangle$	trace de longueur n $(n \in \omega+1)$ où A est la séquence d'états sur S $(A \in (n \to S))$ et a la séquence d'actions sur A $(a \in (n-1 \to A))$, 2.1.1
p	trace, $(p = p_0$ si $\lvert p \rvert = 1$, $p = \langle p_0 \xrightarrow{a_i} p_{i+1} : i \in \lvert\lvert p \rvert\rvert \rangle$ si $\lvert p \rvert > 1)$, 2.1.1
$\lvert p \rvert$	longueur de la trace p en nombre d'états, 2.1.1
$\lvert\lvert p \rvert\rvert$	longueur de la trace p en nombre d'actions, 2.1.1
p_i	ième état de la trace p, 2.1.1
$\ddagger_i$	ième action de la trace p, 2.1.1
$p^{<m}$	préfixe $\langle p_0 \dots p_{m-1} \rangle$ d'une trace p, 2.1.1
$p^{\leq m}$	préfixe $\langle p_0 \dots p_m \rangle$ d'une trace p, 2.1.1
$p^{>m}$	suffixe $\langle p_{m+1} \dots \rangle$ d'une trace p, 2.1.1
$p^{\geq m}$	suffixe $\langle p_m \dots \rangle$ d'une trace p, 2.1.1
$p^{<m,n>}$	tranche d'une trace p, $(\ (p^{<m})^{>n}\ )$, 2.1.1
$p^{\leq m,n\geq}$	tranche d'une trace p, $(\ (p^{\leq m})^{\geq n}\ )$, 2.1.1
$p^{<m,n\geq}$	tranche d'une trace p, $(\ (p^{<m})^{\geq n}\ )$, 2.1.1
$p^{<m,n>}$	tranche d'une trace p, $(\ (p^{\leq m})^{>n}\ )$, 2.1.1
$p \leftrightarrow q$	la trace p est préfixe de la trace q, $(\exists i \in \lvert q \rvert . p = q^{<i})$, 2.6.1
$p \twoheadrightarrow q$	la trace p est suffixe de la trace q, $(\exists i \in \lvert q \rvert . p = q^{>i})$, 2.6.2
$p \xrightarrow{a} q$	concaténation des traces p et q par l'action a, 2.1.1
$\underline{\ddagger}(S_\ell)$	prendre sur un sémaphore S_ℓ 2.8.5.1
$\underline{\ddagger}(S_\ell, i)$	action qui correspond à l'exécution de la commande $\underline{\ddagger}(S_\ell)$ par le processus P_{rs_i} et au passage de ce sémaphore par i

$\ddagger$	action correspondant à un pas d'exécution du prélude d'un programme parallèle, 2.8.2.2.2
$\ddagger'$	action correspondant à un pas d'exécution du postlude d'un programme parallèle, 2.8.2.2.2
P	propriété d'un programme, $(P \in ((\underline{\mathrm{Spec}} \times \underline{\mathrm{Sem}} \langle \mathcal{S}, \mathcal{A} \rangle) \to \{tt; ff\}))$, 2.5
$\mathcal{P}_{\mathrm{pa}}$	programme parallèle asynchrone, 2.8.2.1
$\mathcal{P}_{\mathrm{pc}}$	programme parallèle communicant, 2.8.3.1
$\mathcal{P}_{\mathrm{ps}}$	programme parallèle synchrone, 2.8.5.1
$\mathcal{P}_{\mathrm{p\omega}}$	programme parallèle faiblement équitable, 2.8.4.1
$\mathcal{P}_{\mathrm{r}}$	programme, 2.1.2
$\mathcal{P}_{\mathrm{ra}}$	processus asynchrone, 2.8.2.1
$\mathcal{P}_{\mathrm{rc}}$	processus parallèle communicant par envois de messages sur rendez-vous, 2.8.3.1
$\mathcal{P}_{\mathrm{rs}}$	processus synchrone, 2.8.5.1
$\mathcal{P}_{\mathrm{s}}$	programme séquentiel, 2.8.1.1
$\{\mathcal{P}_{\mathrm{s}}\}$	une liste de commandes séquentielles exécutées de manière indivisible, 2.8.2.1
$\mathbb{P}_{\mathrm{pa}}$	ensemble des programmes parallèles asynchrones, $(\mathcal{P}_{\mathrm{pa}} \to \mathcal{P}_{\mathrm{s}} [\![\mathcal{P}_{\mathrm{ra}_0} \| \dots \| \mathcal{P}_{\mathrm{ra}_{n-1}}]\!] ; \mathcal{P}_{\mathrm{s}}'$ $(n>1)\ )$, 2.8.2.1
$\mathbb{P}_{\mathrm{pc}}$	ensemble des programmes parallèles communicants, $(\mathcal{P}_{\mathrm{pc}} \to \mathcal{P}_{\mathrm{s}} [\![\mathcal{P}_{\mathrm{rc}_0} \| \dots \| \mathcal{P}_{\mathrm{rc}_{n-1}}]\!] ; \mathcal{P}_{\mathrm{s}}'$ $(n>1)\ )$, 2.8.3.1
$\mathbb{P}_{\mathrm{ps}}$	ensemble des programmes parallèles synchrones, $(\mathcal{P}_{\mathrm{ps}} \to \mathcal{P}_{\mathrm{s}} [\![\mathcal{P}_{\mathrm{rs}_0} \| \dots \| \mathcal{P}_{\mathrm{rs}_{n-1}}]\!] ; \mathcal{P}_{\mathrm{s}}'$ $(n>1)\ )$, 2.8.5.1
$\mathbb{P}_{\mathrm{p\omega}}$	ensemble des programmes parallèles faiblement équitables, $(\mathcal{P}_{\mathrm{p\omega}} \to \mathcal{P}_{\mathrm{pa}})$
$\mathbb{P}_{\mathrm{r}}$	ensemble des programmes $(\mathcal{P}_{\mathrm{r}} \to \mathcal{P}_{\mathrm{s}} \mid \mathcal{P}_{\mathrm{pa}} \mid \mathcal{P}_{\mathrm{pc}} \mid \mathcal{P}_{\mathrm{p\omega}} \mid \mathcal{P}_{\mathrm{ps}})$, 2.8
$\mathbb{P}_{\mathrm{ra}}$	ensemble des processus asynchrones, $(\mathcal{P}_{\mathrm{ra}} \to \mathcal{L}_0 : \mathcal{C}_{a_0} ; \dots ; \mathcal{L}_{m-1} : \mathcal{C}_{a_{m-1}} ; \mathcal{L}_m :$ $(m>1)\ )$, 2.8.2.1
$\mathbb{P}_{\mathrm{rc}}$	ensemble de processus communicant par envois de messages sur rendez-vous $(\mathcal{P}_{\mathrm{rc}} \to \mathcal{L}_0 : \mathcal{C}_{c_0} ; \dots ; \mathcal{L}_{m-1} : \mathcal{C}_{c_{m-1}} ; \mathcal{L}_m :$ $(m>1)\ )$, 2.8.3.1

$\mathcal{P}rs$	ensemble des processus synchrones, ($\mathcal{P}rs \longrightarrow \mathcal{L}_0 : \mathcal{C}s_0 ; \ldots ; \mathcal{L}_{m-1} : \mathcal{C}s_{m-1} ; \mathcal{L}_m :$ $(m>1)$), 2.8.5.1
$\mathcal{P}s$	ensemble des programmes séquentiels, ($\mathcal{P}s \longrightarrow \mathcal{L}_0 : \mathcal{C}_0 ; \ldots ; \mathcal{L}_{m-1} : \mathcal{C}_{m-1} ; \mathcal{L}_m :$ $(m>0)$), 2.8.1.1
$\{\mathcal{P}s\}$	ensemble des listes de commandes séquentielles exécutées de manière indivisible, 2.8.2.1
Pref	fermeture par préfixes, 2.6.1
$\text{Pref}^{<\omega}$	préfermeture par préfixes finis, 2.6.1
$\text{Pref}(\langle S,A,\Sigma\rangle)$	fermeture par préfixes de la sémantique $\langle S,A,\Sigma\rangle$, ($\langle S,A,\{p\in\Sigma^{<\omega}\langle S,A\rangle.\ \exists q\in\Sigma.\ p\mapsto q\}\rangle$), 2.6.1
$\text{Pref}^{<\omega}(\langle S,A,\Sigma\rangle)$	préfermeture par préfixes finis de la sémantique $\langle S,A,\Sigma\rangle$, ($\langle S,A,\{p\in\Sigma^{<\omega}\langle S,A\rangle.\ \exists q\in\Sigma.\ p\mapsto q\}\rangle$), 2.6.1
Q	file d'attente associée à un sémaphore, ($Q\in(\mathcal{N}e\to n^{<\omega})$ où n est le nombre de processus synchrones), 2.8.5.2.1
ra	relation entre actions, ($ra\in(\mathcal{A}t\times\mathcal{A}t\to\{tt,ff\})$), 2.5.3
rs	relation entre états, ($rs\in(\mathcal{S}t\times\mathcal{S}t\to\{tt,ff\})$), 2.5.3
$\simeq\langle rs,ra\rangle$	concordance aux relations rs entre états et ra entre actions près, 2.5.3
$\simeq\langle rs,ra\rangle(p,q)$	concordance aux relations rs entre états et ra entre actions près entre les traces p et q, ($[\![\,|p|=|q|\ \wedge\ \forall i\in[\![|p|.\ rs(p_i,q_i)\ \wedge\ \forall i\in[\![\,|p|.\ ra(p_i,q_i)]\!]$), 2.5.3
$\simeq\langle rs,ra\rangle(\langle S,A,\Sigma\rangle,\langle S',A',\Sigma'\rangle)$	concordance aux relations rs entre états et ra entre actions près entre sémantiques, ($[\![\,S'=rs[S]\ \wedge\ A'=ra[A]\ \wedge\ \Sigma'=\simeq\langle rs,ra\rangle[\Sigma]\,]\!]$), 2.5.3
$\simeq\langle rs,ra\rangle(\langle S,A,\xrightarrow{t},E\rangle,\langle S',A',\xrightarrow{t'},E'\rangle)$	concordance aux relations rs entre états et ra entre actions près entre systèmes de transition, ($\simeq\langle rs,ra\rangle(\langle S,A,\Sigma\langle S,A,\xrightarrow{t},E\rangle\rangle,\langle S',A',\Sigma\langle S',A',\xrightarrow{t'},E'\rangle\rangle)$), 2.5.3
$\text{Redei}\ \langle A'\rangle(p)$	trace dérivée de $p\in\Sigma\langle S,A\rangle$ par réduction des actions inobservables $A\cup A'$, 2.5.4.2
$\text{Redei}\ \langle S'\rangle\langle S\rangle$	trace dérivée de $s\in\Sigma\langle S,A\rangle$ par réduction des états inobservables SvS', 2.5.4.1

$\text{Redei}\ \langle S'\rangle(\langle S,A,\Sigma\rangle)$	sémantique dérivée de $\langle S,A,\Sigma\rangle$ par réduction des états inobservables SvS', ($\langle S',A^{*\omega},\ \text{Redei}\ \langle S'\rangle[\Sigma]\rangle$), 2.5.4.1
$\text{Redei}\ \langle S'\rangle(\langle S,A,\xrightarrow{t},E\rangle,\langle S',A',\xrightarrow{t'},E'\rangle)$	réduction des états inobservables SvS' entre systèmes de transition, ($[\text{Redei}\ \langle S'\rangle(\langle S,A,\Sigma\langle S,A,\xrightarrow{t},E\rangle\rangle) = \langle S,A',\Sigma\langle S',A',\xrightarrow{t'},E'\rangle\rangle]$), 2.5.4.1
$\text{Redeia}\ (\langle S,A,\Sigma\rangle)$	Réduction aux états et actions accessibles de la sémantique $\langle S,A,\Sigma\rangle$, ($\langle\{a\in S:\exists p\in\Sigma,i\in[\![|p|.\ p_i=a\},\{a\in A:\exists p\in\Sigma,j\in[\![|p|.\ p_j=a\},\Sigma\rangle$), 2.6.3
$\text{Redeia}\ (\langle S,A,\xrightarrow{t},E\rangle,\langle S',A',\xrightarrow{t'},E'\rangle)$	réduction aux états et actions accessibles entre systèmes de transition ($[\text{Redeia}\ (\langle S,A,\Sigma\langle S,A,\xrightarrow{t},E\rangle\rangle) = \langle S',A',\Sigma\langle S',A',\xrightarrow{t'},E'\rangle\rangle]$), 2.6.3
Reps	réduction par élimination des traces préfixes strictes, 2.6.6
$\text{Reps}\ (\langle S,A,\Sigma\rangle)$	réduction par élimination des traces préfixes strictes de la sémantique $\langle S,A,\Sigma\rangle$, ($\langle S,A,\{p\in\Sigma:\forall p\in\Sigma.\ (p\mapsto q)\Rightarrow(p=q)\}\rangle$), 2.6.6
Rtran	rétraction par transitions, 2.6.8
$\text{Rtran}(\langle S,A,\Sigma\rangle)$	rétraction par transitions de la sémantique $\langle S,A,\Sigma\rangle$, ($\langle S,A,\Sigma\langle S,A,t\langle S,A,\Sigma\rangle,E\langle S,A,\Sigma\rangle\rangle\rangle$), 2.6.8
a	état, 2.1.1
$\hat{a}$	état unique, 2.5.3.2
S	ensemble d'états, 2.1.1
$S[\![Pr]\!]$	ensemble non vide d'états associé au programme Pr, 2.1.2
Se	sémaphore, 2.8.5
Sp	spécification d'un programme, 2.5
$Sl[\![Pps]\!]$	ensemble d'états associé au programme synchrone Pps par la sémantique libérale, 2.8.5.3
$\mathcal{S}t$	ensemble des états, 2.1.2
$\mathcal{S}e$	ensemble des sémaphores, ($\mathcal{S}e\subseteq\mathcal{C}$), 2.8.5
$\langle S,\Sigma\rangle$	sémantique concordante à $\langle S,A,\Sigma\rangle$ à l'annulation des actions près, 2.5.3.3
$\langle S,A,\xrightarrow{t}\rangle$	sémantique, 2.1.2
$\langle S,A,\Sigma\langle S,A,\xrightarrow{t},E\rangle\rangle$	sémantique engendrée par le système de transition $\langle S,A,\xrightarrow{t},E\rangle$, 2.4
$\langle S[\![Pr]\!],A[\![Pr]\!],\Sigma[\![Pr]\!]\rangle$	sémantique associée au programme Pr, 2.1.2 - 2.8.1.2

$\langle S, A, t, E\rangle$	système de transition, 2.2
$\langle S, A, t\langle S,A,\Sigma\rangle, E\langle S,A,\Sigma\rangle\rangle$	système de transition engendré par la sémantique $\langle S,A,\Sigma\rangle$, 2.3
$\langle S[\![Pr]\!], A[\![Pr]\!], t[\![Pr]\!], E[\![Pr]\!]\rangle$	système de transition associé au programme Pr, 2.8.1.2
skip	commande nulle, 2.8.1.1
Sem $\langle \mathcal{S}, \mathcal{A}\rangle$	ensemble des sémantiques sur les ensembles $\mathcal{S}$ d'états et $\mathcal{A}$ d'actions $(\{\langle S,A,\Sigma\rangle : S\in\mathcal{S} \wedge A\in\mathcal{A} \wedge \Sigma\in\Sigma\langle S,A\rangle\})$, 2.1.2
$\langle$Sem $\langle \mathcal{S},\mathcal{A}\rangle, \subseteq, \langle 0,0,0\rangle, \langle \mathcal{S},\mathcal{A},\Sigma\langle\mathcal{S},\mathcal{A}\rangle\rangle\rangle$	treillis complet des sémantiques, 2.5.1
Sfair $(\langle S,A,\Sigma\rangle)$	réduction d'une sémantique $\langle S,A,\Sigma\rangle$ aux traces fortement équitables, (Sfair $\langle A\rangle(\langle S,A,\Sigma\rangle)$), 2.6.4
Sfair $\langle\alpha\rangle(\langle S,A,\Sigma\rangle)$	réduction d'une sémantique $\langle S,A,\Sigma\rangle$ aux traces fortement équitables pour un ensemble α d'actions, $(\langle S,A,\{p\in\Sigma : \lvert p\rvert=\omega \Rightarrow \neg(\exists a\in\alpha, i\in\omega.(\forall j>i.\exists k>j.$ Enabled $(a,k,p,\Sigma))\wedge\forall j>i.\, p_j\neq a)\}\rangle)$, 2.6.4
Spec	ensemble des spécifications, 2.5
succ	succ $[\![Ps]\!](L)(M,M')$ est la condition pour que l'état mémoire M' soit successeur de l'état mémoire M après exécution d'un pas de Ps au point de contrôle L, 2.8.1.2.4
Suff	fermeture par suffixes, 2.6.2
Suff $(\langle S,A,\Sigma\rangle)$	fermeture par suffixes de la sémantique $\langle S,A,\Sigma\rangle$, $(\langle S,A,\{p\in\Sigma^{\leq\omega}\langle S,A\rangle : \exists q\in\Sigma.\, p\frown q\in\Sigma\}\rangle)$, 2.6.2
se … or … or … es	commande alternative, 2.8.3.1
t	relation de transition, $t\in(A\to(S\times S\to\{tt,ff\}))$, 2.2
t^*	fermeture transitive réflexive de t, $(t^*(s,s') = \bigcup_{n\geq 0} t^n(s,s')$ avec $t^0(s,s')=[s=s']$, $t^{n+1}(s,s')=[\exists a\in A, s''\in S.(t_a(s,s'')\wedge t^n(s'',s'))]$), 2.6.2
$t\,!\,S_d, S_i, S_f\,!_{\langle\rangle}(s,s')$	relation de transition entre un état s de S_d et un état s' de S_f par aucune action sur des états intermédiaires de S_i, $([\exists n\in(n+2\to S).(s_0=s\in S_d \wedge \forall j\in(n+1\setminus 0).s_j\in S_i \wedge s_{n+1}\in S_f \wedge \forall j\in(n+1).t_{a_j}(s_j,s_{j+1}))])$, 2.5.4
$t\llbracket Pr\rrbracket$	relation de transition associée au programme Pr, 2.8.1.2

$t\,!\,S_d, S_i, S_f\,!_{\langle a_0,\ldots,a_n\rangle}(s,s')$	relation de transition entre un état s de S_d et un état s' de S_f par les actions $a_0 \ldots a_n$ sur des états intermédiaires de S_i, $([\exists n\in(n+2\to S).(s_0=s\in S_d \wedge \forall j\in(n+1\setminus 0).s_j\in S_i \wedge s_{n+1}\in S_f \wedge \forall j\in(n+1).t_{a_j}(s_j,s_{j+1}))])$, 2.5.4
$t\langle S,A,\Sigma\rangle$	relation de transition engendrée par la sémantique $\langle S,A,\Sigma\rangle$, $([\exists p\in\Sigma, i\in\lvert p\rvert.(p_i=s \wedge p_{i+1}=a \wedge p_{i+1}=s')] = t_a(s,s'))$, 2.3
$tl\,[\![Pps]\!]$	relation de transition associée au programme synchrone Pps par la sémantique libérale, 2.8.5.3
Tran $\langle \mathcal{S},\mathcal{A}\rangle$	ensemble des systèmes de transitions sur les ensembles $\mathcal{S}$ d'états et $\mathcal{A}$ d'actions, $(\{\langle S,A,t,E\rangle : S\in\mathcal{S}\wedge A\in\mathcal{A}\wedge t\in(A\to(S\times S\to\{tt,ff\}))\wedge E\in(S\to\{tt,ff\})\})$, 2.2
$\underline{v}$ (se)	rendre le sémaphore se, 2.8.5.1
$\underline{v}$ (se, i)	action qui correspond à l'exécution de la commande $\underline{v}$(se) par le processus Pr_i qui libère le sémaphore alors qu'aucun autre processus n'était en attente sur ce sémaphore, 2.8.5.2.2
$\underline{v}$ (se, i, j)	action qui correspond à l'exécution de la commande $\underline{v}$(se) par le processus Pr_i qui libère le sémaphore et permet au processus Pr_j qui était en attente de le passer, 2.8.5.2.2
VA	ensemble des variables auxiliaires, 4.1
$\mathcal{V}$	ensemble des variables, 2.8.1.1
$\underline{p}$ (se, i)	action qui correspond à l'exécution de la commande $\underline{p}$(se) par le processus Pr_i qui provoque sa mise en attente devant le sémaphore se, 2.8.5.2.2
Wfair $(\langle S,A,\Sigma\rangle)$	réduction d'une sémantique $\langle S,A,\Sigma\rangle$ aux traces faiblement équitables, (Wfair $\langle A\rangle(\langle S,A,\Sigma\rangle)$), 2.6.4
Wfair $\langle\alpha\rangle(\langle S,A,\Sigma\rangle)$	réduction d'une sémantique $\langle S,A,\Sigma\rangle$ aux traces faiblement équitables pour un ensemble α d'actions, $(\langle S,A,\{p\in\Sigma : \lvert p\rvert=\omega \Rightarrow \neg(\exists a\in\alpha, i\in\omega.\forall j>i.(\text{Enabled}(a,k,p,\Sigma) \wedge p_j\neq a))\})$, 2.6.4
while … do … od	composition itérative de commandes, 2.8.1.1