THESE

présentée à

INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE·

pour obtenir le grade de
DOCTEUR DE 3e CYCLE EN INFORMATIQUE

par

Armand PUCCETTI

-:-:-

Preuves de Propriétés de Fatalité Temps-Réel de Programmes ADA, Sémantiques Opérationnelle et Axiomatique Associées.

-:-:-

Thèse soutenue le 1987

devant la commission d'examen:

D 136 037869 7

Président: Mr P. COUSOT

Examinateurs:

INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE
Section Documentaire ENSEM

136037869-1

THESE

CTI 1987 PUCETTI, A.

présentée à

INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE.

pour obtenir le grade de
DOCTEUR DE 3e CYCLE EN INFORMATIQUE

par
Armand PUCCETTI

Service Commun de la Documentation
INPL
Nancy-Brabois

-:-:-

Preuves de Propriétés de Fatalité Temps-Réel de Programmes ADA, Sémantiques Opérationnelle et Axiomatique Associées.

-:-:-

Thèse soutenue le
devant la commission d'examen:

Président: Mr P. COUSOT
Examinateurs:

TABLE DES MATIERES

CHAPITRE I INTRODUCTION

CHAPITRE II LE LANGAGE DE PROGRAMMATION ADA - RAPPELS - HYPOTHESES DE TRAVAIL.

II.A LES STRUCTURES DE CONTROLE SEQUENTIELS. II-2
II.A.i L'instruction D'affectation. II-3
II.A.ii L'instruction Conditionnelle. II-3
II.A.iii L'instruction Répétitive. II-3
II.A.iv L'instruction Composée. II-4
II.A.v L'instruction "null". II-4
II.B LA NOTION DE TACHES ET LEURS DIFFERENTS ETATS. . II-4
II.C LA GESTION DU TEMPS EN ADA. II-5
II.D LES MOYENS DE COMMUNICATION ENTRE TACHES. . . . II-7
II.D.i La Notion De Point D'entrée II-8
II.D.ii L'instruction D'acceptation De Point D'entrée. II-8
II.D.iii Les Interruptions. II-10
II.E LES INSTRUCTIONS "SELECT". II-10
II.E.i L'instruction "select" Coté Appelée. II-10
II.E.ii L'instruction "select" Coté Appelante II-12
II.F LE CYCLE DE VIE D'UNE TACHE. II-13
II.G CONCLUSION. II-15

CHAPITRE III SEMANTIQUE OPERATIONNELLE DES PROGRAMMES ADA.

III.A INTRODUCTION. III-1
III.B DEFINITIONS. III-3
III.B.i Définition: III-3
III.B.ii Remarque: III-4
III.B.iii Définition: III-5
III.B.iv Remarque: III-6
III.B.v Définition: III-6
III.B.vi Notation: III-6
III.C LA RELATION DE TRANSITION t[IP]. III-7
III.C.i Définition: III-9
III.C.ii Définition: La Relation De Transition t[IP]. III-10
III.C.iii Définition: L'ensemble Des Traces D'exécution T[IP] III-11
III.C.iv Définition: III-11
III.C.v Proposition: III-12
III.D LES TRANSITIONS ELEMENTAIRES DE LA RELATION >-IP->. III-12
III.D.i Notation: III-13
III.D.ii Proposition: III-13
III.D.iii Définition: III-14
III.D.iv La Transition Associée A L'instruction D'affectation. III-15
III.D.v La Transition Associée A L'instruction "null". III-15

III.D.vi Transitions Associées A L'instruction Itérative. III-16
III.D.vii Transition Associée A L'instruction Conditionnelle. III-17
III.D.viii Remarques: III-17
III.D.ix Transitions Associées A L'instruction De Délai. III-18
III.D.x Transition Associée A L'état D'erreur. . III-18
III.D.xi Transition Associée A L'état Final. . . . III-19
III.D.xii Transitions Associées A L'instruction "select" Coté Appelé. III-19
III.D.xiii Transitions Associées A L'instruction D'appel De Point D'entrée. III-25
III.D.xiv Transitions Associées A L'instruction D'appel De Point D'entrée Limitée Dans Le Temps. III-26
III.D.xv Définition: III-28
III.D.xvi Proposition: III-29
III.D.xvii Exemple De Graphe De Transition Associé A La Sémantique Observationnelle D'un Programme. III-30
III.E FORMALISATION DES NOTIONS D'EQUITE PAR LES MACHINES DE TURING. III-33
III.E.i Définition: III-34
III.E.ii Définition: III-34
III.E.iii Théorème: III-35
III.E.iv Définition: III-35
III.E.v Notation: III-36
III.E.vi Définition: III-36
III.E.vii Définition: III-36
III.E.viii Machines De Turing A Oracle. III-36
III.E.ix Expression De L'équité Faible. III-40
III.E.x Expression De L'équité Sur Les Points D'entrée. III-43

CHAPITRE IV LE SYSTEME AXIOMATIQUE DE PREUVE A.R.T.

IV.A LANGAGES D'ASSERTIONS ASSOCIES A UN PROGRAMME ADA - DEFINITION. IV-2
IV.A.i Définition: IV-3
IV.A.ii Notation: IV-3
IV.A.iii Définition: IV-4
IV.A.iv Notation: IV-6
IV.A.v Proposition: IV-6
IV.B LE SYSTE'ME DE PREUVE A.R.T. IV-8
IV.B.i Définition: IV-9
IV.B.ii Première Partie: Axiomes Associés Aux Instructions Séquentielles. IV-9
IV.B.iii Deuxième Partie: Axiomes Associés Aux Instructions De Communication. IV-10
IV.B.iv Troisième Partie: Les Règles De Composition. IV-19
IV.B.v Quatrième Partie: Axiomes Et Règles Auxiliaires. IV-20
IV.B.vi Définition: IV-22
IV.C CORRECTION DU SYSTEME AXIOMATIQUE A.R.T. IV-23
IV.C.i Théorème: IV-23
IV.D CONCLUSION. IV-32

CHAPITRE V PROPRIETES DE FATALITE TEMPS-REEL - EXEMPLES DE PREUVES.

V.A PROPRIETES TEMPORELLES EXPRESSIBLES DANS LE SYSTEME A.R.T. V-1
V.A.i Définition: V-5
V.B SYNTHESE D'UN PROGRAMME SATISFAISANT A UNE PROPRIETE DE FATALITE TEMPS-REEL. V-7
V.C UNE METHODE DE PREUVE DE PROPRIETES TEMPS-REEL DANS A.R.T. V-13
V.D PREMIER EXEMPLE: SIMULATION D'UN FILTRE DE COURANT ALTERNATIF. V-15
V.D.i Enoncé Du Problème. V-15
V.D.ii Enoncé De La Propriété A Démontrer. . . . V-18
V.D.iii Preuve De La Propriété De Fatalité Temps-réel. V-19
V.E SECOND EXEMPLE: UN ADDITIONNEUR. V-26
V.E.i Enoncé Du Problème. V-26
V.E.ii Enoncé De La Propriété A Démontrer. . . . V-31
V.E.iii Preuve De La Propriété De Fatalité Temps-réel. V-31

CHAPITRE VI CONCLUSION.

APPENDICE A GRAMMAIRE DU SOUS-ENSEMBLE DE ADA ETUDIE

APPENDICE B REFERENCES BIBLIOGRAPHIQUES

INTRODUCTION

CHAPITRE I

INTRODUCTION

Quoique l'étude de la programmation parallèle ait fait l'objet de nombreuses investigations, celà depuis plus de vingt ans, peu de travaux théoriques ont approfondi les problèmes spécifiques à la programmation temps-réel. Cet état de fait est assez normal dans la mesure où quelques ordres de magnitude au moins séparent la difficulté des problèmes temps-réel de celle de leurs homologues parallèles, tandis que la complexité dans l'étude des problèmes séquentiels n'est supérieure à cette dernière que d'un seul ordre de difficulté. Les langages de programmation temps-réel couramment employés dans l'industrie ne présentent que peu d'évolution par rapport au langage assembleur. Il s'en suit qu'aucune sérieuse analyse de complexité, ni aucune méthodologie, ni aucun standart pour l'implémentation ni pour la portabilité n'existent en matière de logiciels temps-réel. Cette situation est d'autant plus surprenante que la confrontation à des problèmes de ce type se fait de plus en plus jour dans des domaines critiques du monde moderne (conduite automatique de véhicules spatiaux, de centrales nucléaires, de chaines de montage, d'installations chimiques, de robots industriels, par exemple). De nouveaux langages évolués, orientés temps-réel tel que Ada, Occam, Chill ont été développés pour répondre à ces besoins. Leurs standards officiels nécessitent toutefois une formalisation

plus acceptable de leurs concepts temps-réel. Il est donc indispensable d'étudier de nouveaux modèles formels pour mieux raisonner avec de tels concepts, plus réalistes dans le sens où ils doivent être basés sur une architecture de machine multiprocesseurs. Il ne doit plus être question de laisser la gestion du temps au bon vouloir d'un système d'exploitation, en supposant que le calculateur sous-jacent soit suffisamment rapide pour respecter approximativement les temps de réponse (voir [56] par exemple), mais d'imposer au logiciel support (le système exécutif d'un langage par exemple) un respect absolu des délais de réponse.

De manière générale, on peut définir un système temps-réel comme étant un système capable de répondre à des évènements externes dans des délais de réponse prédéfinis.

Un tel sytème doit être sensible à deux types de manifestations de son environnement:

i) des évènements, éventuellement porteurs d'information associée, sur des sources d'évènements. Le système prend en compte les évènements "instantanément" et capte la durée entre deux évènements successifs sur une même source;

ii) des valeurs évoluant continuement (par exemple le température, la pression...). Le système est continuement sensible aux variations de ces valeurs. (Il s'agira par exemple de surveiller qu'une température ne franchisse pas un certain seuil).

De même, le système temps-réel agit sur l'environnement par des manifestations de même type (des sources d'évènement et des variables).

Le présent travail tente de donner une méthode de spécification et de preuve de propriétés temps-réel pour des programmes ADA (cf. [19]). Pour celà,

nous nous sommes basés sur la logique temporelle arborescente (cf. [39]-[43]), grâce à laquelle un système de preuve axiomatique, basé sur un nouvel opérateur "until" a pu être élaboré. Nous avons choisi ADA comme langage support de nos algorithmes temps-réel pour son intérêt stratégique parmi les langages de programmation existants. Nôtre papier de base, [51], proposait une sémantique opérationnelle pour un sous-ensemble de ADA (incluant la notion de tâches concurrentes) et suggérait une possibilité d'extension de celle-ci, permettant de modéliser également des programmes temps-réel.

Au cours de ces quelques dernières années, de nombreuses méthodes suggérant un cadre plus formel pour l'étude des problèmes temps-réel sont apparues. Il est possible de les classer en six grandes catégories, représentant chacune un modèle précis. Nous avons extrait de chacune d'entre-elles un représentant, dont nous allons rappeler les caractéristiques essentielles, dans le but d'établir une comparaison entre les approches synchrones et asynchrones du temps-réel:

1) Dans le papier de Koyman et co. (cf. [33]), les auteurs nous présentent une sémantique dénotationnelle du langage Occam temps-réel, basé sur un modèle de machine multiprocesseur. Cette sémantique associe à une instruction et à un état un ensemble d'historiques des communications, en tenant compte des "temps morts" où aucune communication n'a lieu. Un historique des communications est une suite d'objets de la forme $\langle i,j,v \rangle$ ou $\langle i,j \rangle$, indiquant respectivement que le processus numéro i transmet la valeur v au processus j, ou bien que le processus i ne communique pas avec le processus j. Ces derniers sont indicés par un entier, représentant l'instant courant. Le temps ainsi référencé, de même

que celui auquel il est fait appel par des instructions de type "delay", est considéré comme étant discret. Afin de mesurer celui-ci on ne fait appel à une horloge globale que dans la sémantique même, sans autoriser les instructions du langage à référencer le temps absolu. Dans cette approche, on introduit l'hypothèse de parallélisme maximal: celà signifie que l'on maximise le nombre d'instructions exécutables simultanément (sans pour autant violer les règles de synchronisation) pour un programme et un instant donnés; il n'y a ainsi jamais d'attentes inutiles; on autorise la règle du premier venu-premier servi pour les communications inter-processus. Koyman et Co. étendent ce modèle aux systèmes distribués par la prise en compte des délais de transmission variables dus au réseau.

2) La seconde catégorie est représentée par les papiers [48] à [50]. Moszkowski nous présente une approche du temps-réel spécialisée dans la spécification et la vérification de circuits logiques employés en électronique et l'étend à la spécification de problèmes algorithmiques classiques. L'idée centrale est de décomposer un intervalle fermé borné de IR en sous-intervalles disjoints entre eux, dans une approche descendante, jusqu'à obtenir une suite d'intervalles élémentaires dans chacun desquels il est possible de décrire le comportement des signaux logiques continus à l'aide d'opérateurs simples. Ces derniers sont définis dans une logique temporelle propositionnelle à intervalles du premier ordre. Le temps est considéré comme étant linéaire, on y trouve des opérateurs analogues à ceux employés en logique temporelle classique (par exemple: o, [a] (défini de la manière suivante

$M_{s0,\ldots,sn}$ ([a]w) = vrai si et seulement si

$\forall\, i,j \in \{0,\ldots,n\},\ i<j,\ M_{si,\ldots,sj}$ (w) = vrai),

⟨a⟩, [i], ⟨i⟩, skip, ~~->,...).

3) Berry et Co. étudient un langage synchrone, déterministe, à temps multiforme, dans lequel la seconde n'est qu'un évènement parmi d'autres, appelé Esterel. Le seul moyen de communication entre les processus d'un programme donné est le flash. Ce type d'évènements, porteur d'informations ou non, est géré par des instructions classiques mais puissantes: emit s(X), await s(X), awaitnext s(X), do i upto s(X), do i uptoeach s(X) sont les plus importantes d'entre-elles. L'originalité de cette approche réside dans l'hypothèse de disposer d'une machine infiniment rapide, ce qui entraîne que seules les instructions d'un programme faisant explicitement référence au temps durent un temps non nul à l'exécution. Berry et Co. proposent également une sémantique calculatoire de leur langage, basée sur une sémantique opérationnelle classique, capable de produire directement un compilateur efficace pour ce langage (cette dernière provient du fait qu'un programme Esterel est compilé en un automate d'états finis, séquentiel et déterministe, dont l'éxécution n'exige aucune gestion de processus).

4) Pnueli et De-Roever (cf. [51]) considèrent une approche plus opérationnelle du langage ADA, ayant pour objectif de prouver des propriétés de vivacité de tels programmes, celà dans un système axiomatique de preuve fondé sur la logique temporelle. Ils utilisent pour celà un modèle d'éxécution à entrelacement

non-déterministe des actions atomiques, sous hypothèse d'équité sur les points d'entrée. Un état d'un programme ADA, noté par s=⟨T1-loc,...,Tn-loc,m⟩, contient un multi-point de contrôle, auquel les auteurs suggèrent d'adjoindre n+1 composantes afin de modéliser également des programmes temps-réel: il s'agit de n+1 entiers représentant n horloges locales aux processus et une horloge globale au système (fictive quant à elle). Cette modification de la notion d'état entraîne le rajout de conditions et de transitions supplémentaires dans la sémantique opérationnelle initiale: il est tout d'abord nécessaire de considérer un type de transition supplémentaire (appelé "time-step transition" dans [51]) ayant pour but de faire progresser les tâches en attente devant une instruction de delai d'une unité de temps simultanément; puis, afin de mieux rendre compatible entre elles les mesures des différentes horloges, il faut créer des compteurs d'actions atomiques associés aux tâches du programme. Ceux-ci seront incrémentés d'une unité à chaque pas de calcul et réinitialisés simultanément par une transition du type mentionné ci-dessus. Mais ceci serait encore insuffisant, si on n'imposait pas des contraintes sur les vitesses relatives d'éxécution des différentes tâches, ce qui influencerait sur le choix de la prochaine transition à effectuer.

Notre approche du temps-réel, prenant ces idées comme point de départ, se situe dans la même catégorie que celle-ci. Mais vu les complications introduites par les idées rappelées ci-dessus, nous ne conserverons qu'une horloge globale référençable par toutes les tâches d'un programme donné, ainsi qu'il est indiqué dans le manuel de référence [19]. Cette horloge globale sera prise à valeurs dans IR+.

5) Le papier de Bernstein et Harter [8], nous décrit une approche opérationnelle d'un langage temps-réel à base de signaux, différant de la précédente dans la mesure où on s'appuie sur un modèle de machine monoprocessorisée. L'éxécution des tâches d'un programme est donc réalisée par un entrelacement non-déterministe des actions élémentaires de celle-ci, faiblement équitable (cf. [6], [34] et [45] pour les différents types d'équité). Un état d'un programme est défini comme étant un quadruplet de la forme s=⟨R,m,D,t⟩, où m est l'état mémoire, t la date absolue, R l'ensemble des actions atomiques activables et D={⟨l,T⟩ / l'action atomique d'étiquette l est suspendue pendant T secondes à partir de la date courante}.
Dans cette sémantique, les délais ne sont pas traités avec exactitude. On remarque par exemple que la date de réactivation d'un processus éxécutant une instruction "b:pause x;" est fausse: en effet, lors de la mise en attente de ce processus, on ajoute l'élément ⟨b,ti+x⟩ à la troisième composante de l'état courant "si", puis on réactive celui-ci en un état de date antérieure à ti+x "le plus proche possible" de celui-ci.
Dans ce même papier, on propose une extension de la logique temporelle classique, en adjoignant des opérateurs définis de la manière suivante:

$e \models I1 \overset{<\ n}{\sim\!\sim\!\!\rightarrow} I2$ (I1 et I2 sont des assertions portant sur des états, e est une séquence de calcul)

$e \models I1 \overset{>\ n}{\sim\!\sim\!\!\rightarrow} I2$

$(P \,[]\, Q) = (P \Rightarrow o(P \,[]\, Q) \wedge Q)$ (ce dernier opérateur permet d'exprimer le fait que tant

que P est vrai, Q l'est également)

path(P,Q,R) = (P => (~Q [] ~R))

(de cette manière on voudrait dire que pour joindre P et R l'on est obligé de "passer par" Q)

ainsi que des règles de preuve dont les suivantes:

$$\frac{A \overset{<n}{\sim\sim\!\!>} B,\ C \overset{>n}{\sim\sim\!\!>} D}{A \wedge C \Rightarrow (\sim B \ \Box\ \sim(D \vee oD))}$$

(cette règle dite du progrès relatif montre que depuis un état où A^C est vrai, B sera fatalement vérifié avant l'assertion D)

$$\frac{A \overset{<n}{\sim\sim\!\!>} oA,\ A \overset{>n}{\sim\sim\!\!>} B}{A \Rightarrow \Box\sim B}$$

(ceci est la règle de "l'écoulement correct du temps")

6) Dans [27] et [28] Halbwachs nous conseille une autre méthode de spécification de problèmes temps-réel. On caractérise un évènement par la suite croissante d'instants auxquels il se produit. Sur de telles entités, on peut réaliser de nombreuses opérations: on peut

les filtrer par un autre évènement, les composer afin d'obtenir de nouveaux évènements, on peut projeter un évènement E selon une condition C (x= when C(E)), on peut consulter la valeur courante d'un évènement (Y= current(E)) afin de replacer des évènements sur une même base de temps, on peut échantillonner un évènement (every C(E)) afin de changer de cycle de base... On peut ainsi traiter des problèmes asynchrones aussi bien que des problèmes synchrones, et l'on sous-entend toujours qu'il existe une certaine horloge de base pour tous les évènements, dont le temps de cycle est indéfini à priori ou est à définir par le contexte du problème.

Halbwachs se situe dans une approche contraire à la nôtre (qu'est de prouver des propriétés de programmes temps-réel), consistant à donner un cadre formel dans lequel, pour un problème donné, on spécifie les liens (sous la forme d'inégalités bien souvent) entre les occurrences d'évènements, puis à prouver l'adéquation d'une implémentation donnée de ce problème à ces spécifications. Cette méthodologie est retrouvée dans le langage de programmation Lustre (dont les variables sont des évènements), visant des applications temps-réel distribuées.

Nous allons tenter maintenant de comparer ces différentes approches du temps-réel, après avoir rappelé quelques concepts relatifs au langage ADA.

Le langage ADA permet la programmation de problèmes temps-réel et contient pour celà des instructions de manipulation du temps. On y trouve deux notions de temps, indépendantes mais complémentaires: le calendrier (package CALENDAR, cf. [19]) pour le temps absolu, et l'expression d'intervalles de temps que l'on trouve dans les demandes de suspension, les

chiens de garde, et la notion d'immédiat. Toute requête au calendrier rend un résultat précis: l'heure. Par contre les notions d'intervalles de temps sont imprécises, ainsi:

a) un rendez-vous avec un délai de x secondes n'implique pas que le rendez-vous éventuel soit effectivement pris en compte avant x secondes,

b) un rendez-vous immédiat n'implique pas que le rendez-vous soit pris en compte à la même heure que sa demande.

En introduisant dans ADA les notions d'interruption, d'horloge voire de priorité, les auteurs ont voulu en faire un langage de programmation temps-réel. Il convient d'être prudent, car la signification de ces notions dépend fortement des implémentations (la programmation temps-réel n'est pas uniquement du ressort d'un langage, mais aussi de celui d'une méthodologie). ADA ne permet pas de tout faire. Il s'agit surtout d'un langage de programmation système et non de calcul parallèle. Aucun mécanisme ne permet, en particulier, d'exprimer le parallélisme synchrone fin, et tel qu'il est défini, le langage n'est manifestement pas utilisable pour des applications de contrôle de procédés où existent des contraintes de réponse trop strictes.

Malgré celà, ce langage est essentiellement dédié aux applications multiprocesseurs (tels que les systèmes embarqués, où l'on peut se permettre d'employer le nombre de processeurs qu'il faut). Comme les programmes ADA ne sont pas nécessairement implantés sur des machines cibles à processeurs élémentaires identiques, le temps relatif n'est pas forcément le même dans toutes les tâches d'un programme donné (i.e. un évènement tel que la seconde n'est pas perçu exactement au même instant (relativement à l'horloge universelle abstraite) par toutes les tâches s'éxécutant de façon

concurrente).

Ce langage est qualifié d'asynchrone car la fin d'une instruction de délai ne se produit pas nécessairement à l'instant où la durée indiquée est écoulée. En d'autres termes, l'évènement "fin d'éxécution du délai" et celui indiquant que la durée est écoulée (ce dernier doit être imaginé comme étant le tic d'une horloge abstraite) ne sont pas forcément synchrones.

Dans le présent travail, nous considérerons les délais comme étant des valeurs exactes (voir chapitre II), qu'il s'agit de respecter au cours de l'éxécution de façon précise. Cette hypothèse parait plus réaliste lorsqu'on dispose, pour l'éxécution d'un programme donné, d'autant de processeurs que nécessaires.

Afin de comparer les approches synchrones et asynchrones du temps-réel, nous avons retenu six critères. (Nous les référencerons dorénavant par leur numéro dans la classification établie plus haut).

1) PREMIER CRITERE: Le modèle de temps employé.

La quasi-totalité des approches considérées (1,3,4,5,6) considère que l'espace du temps est isomorphe à IN, par analogie avec le découpage d'un programme en cycles-machine élémentaires, et que les durées peuvent être prises comme étant des valeurs entières. Il est pourtant évident que la vitesse d'éxécution des calculateurs double en moyenne tous les deux ans (le record de vitesse est détenu par le Cray 2; son temps de cycle est d'environ 4.1 ns), et nous pensons (ainsi que dans le modèle numéro 2), qu'il est possible (à des fins de raisonnement abstrait et également dans l'intérêt des programmeurs) de nous placer dans le cas limite, où on supposerait la machine capable d'éxécuter un délai de durée

quelconque, à valeur dans IR+.

Dans le modèle numéro 3, le temps est considéré comme étant multiforme, i.e. la mesure d'une suite d'évènements comme le temps est donné par un agent ou un processus extérieur au système. On peut constater que dans la plupart des approches, le temps est un cadencement arbitraire du programme, induit par son environnement (le système d'exploitation par exemple) et il n'existe pas toujours de mesure précise du temps, c'est à dire de lien avec le temps universel tel que nous l'entendons. Dans les modèles 1, 4 et 6, on admet l'existence d'une notion du temps, dont la mesure importe peu et dont la connaissance n'est nécessaire qu'à une implémentation réelle du langage (et non pas à une machine abstraite). Quant au modèle numéro 2, le problème de la connaissance d'une mesure du temps ne se pose guère, car on ne fait que simuler des phénomènes possédant par essence une mesure du temps (donnée par un quartz, un circuit RLC,...). Pour certains problèmes temps-réel il est également nécessaire de connaître la date absolue: celà peut se faire aisément dans le modèle numéro 4, car l'état du programme possède explicitement une composante qu'est le temps universel; mais bien souvent il est indispensable de rajouter un mécanisme particulier (laissé à la charge du programmeur) permettant au programme de la connaître à tout instant (on propose souvent d'introduire un processus horloge ou un compteur supplémentaire).

2) SECOND CRITERE: L'aptitude à modéliser les systèmes temps-réel avec précision.

Les sémantiques temps-réel que nous avons étudiées ne mettent en évidence que deux modèles de machine sous-jacente.

Tout d'abord, historiquement, furent les modèles à l'éxécution entrelacée, simulant le fonctionnement des ordinateurs monoprocesseurs. Ce genre de modèle (ainsi que le modèle de système "conduit par la priorité") se rencontre encore très fréquement dans la pratique (cf. [38], [47] et [57] par exemple), mais sera à l'avenir délaissé en faveur du modèle multiprocesseur, afin de ne servir qu'à des fins de simulation ou de programmation en tant que machine source. Son plus grand défaut est de provoquer des phénomènes de dérive dans le temps:

i) dues à la quantification du temps, on observe des inexactitudes dans l'éxécution d'instructions de délai dans le modèle numéro 5: la réactivation d'un processus ayant éxécuté une telle instruction se fait avant l'heure prévue.

ii) il est souvent impossible de mesurer (et d'estimer) le temps consommé par le système d'exploitation, valeur augmentant avec la charge de ce dernier, donc particulièrement critique pour une application temps-réel.

iii) des erreurs plus graves peuvent se produire dans les systèmes guidés par la priorité des processus (cf. [57]): une tâche n'ayant pas achevée son travail pendant le quota de temps imparti est susceptible d'obtenir une prolongation et devra restituer ce temps immédiatement après (ceci peut être réalisé en privant de calcul cette tâche pendant le cycle système suivant).

Dans le modèle de machine monoprocessorisée, il est en règle générale impossible de respecter les délais de façon précise:

ainsi, le programme suivant ne se terminera qu'à la date t=1+θ, car il n'est possible d'exécuter qu'une transition à la fois.

```
exemple:

        T0 ::                   T1 ::
                10:                     10:
                   pause 1; //             pause 1;
                11:                     11:
```

- Figure 1 -

Le modèle de machine multiprocesseur est surtout employé pour des langages asynchrones (comme CSP-R dans l'approche numéro 1, ou ADA dans notre cas), car on éxécute les transitions de plus courte durée simultanément, en un état donné du programme. On s'efforce d'obtenir un parallélisme maximal, que ce soit dans une sémantique opérationnelle ou dénotationnelle. (Remarquons tout de même que le programme de l'exemple ci-dessus se termine à la date t=1 dans une telle optique, car on peut éxécuter simultanément les deux instructions de même durée dans une seule transition).

Dans la plupart des modèles temps-réel, possédant une mesure du temps, le quantum de temps est une valeur constante, θ, arbitrairement petite par rapport à l'unité qu'est la seconde. Dans les classes numéro 1 et 2, θ est considéré comme étant non-nul, contrairement à la classe numéro 3 où on suppose la machine abstraite infiniment rapide (il s'agit de l'hypothèse dite synchrone forte). Cette dernière hypothèse peut sembler irréaliste, mais a des conséquences non négligeables sur le langage

de programmation: celà évite de recourir au non-déterminisme, pourtant classique dans les communications inter-processus, et permet de produire un programme objet très rapide (ce dernier étant un automate d'état fini). Il existe cependant des cas (tels que les calculs en mode réel, l'appel de fonctions récursives, la création dynamique de processus...) où il sera impossible de conserver une telle hypothèse de travail.

Dans la présente approche, nous proposons de conserver l'idée d'un quantum de temps non-nul, et chaque instruction ou expression du langage nécessitera un temps exact à l'éxécution (fonction de θ), calculable par analyse syntaxique du texte source.

3) TROISIEME CRITERE: Le non-déterminisme à l'éxécution.

La notion de programme déterministe est liée à celle de langage synchrone et permet (comme dans le troisième cas) de compiler le programme en un programme objet plus efficace dont l'ordre des évènements est clair une fois connu l'historique des évènements en entrée, et où les délais peuvent être traités avec exactitude.

L'approche asynchrone du temps-réel laisse surgir des défauts graves, par exemple:

i) - l'explosion du nombre d'états d'un programme,

ii) - la gestion implicite des files d'attente (associées aux moyens de communication inter-processus) nécessaires pour garantir les hypothèses d'équité dans une implantation donnée, d'où "overhead" système.

mais possède également quelques avantages non-négligeables:

i) - une plus grande facilité à modéliser des systèmes distribués, où le reseau de communication est sujet à des variations assimilables à des phénomènes non-déterministes (non-prévisibles pour un utilisateur ponctuel de ce réseau),

ii) - possibilité de synchronisme et d'asynchronisme dans les communications,

iii) - facilité de synchronisation des tâches sans modification majeure dans la sémantique.

4) QUATRIEME CRITERE: La notion d'évènements.

L'évènement est une notion universellement employée dans tous les modèles du temps-réel, car elle assure un cadencement des programmes liés à des stimulis externes (capteurs par exemple) ou à la réalisation de certains états internes, ainsi qu'une synchronisation entre les différentes activités d'un système.

Les évènements peuvent posséder de nombreuses caractéristiques, dont les suivantes: ils peuvent être typés, porteurs d'informations, composables, restrictibles à un bloc d'instructions, être opérandes d'expressions logiques; ils peuvent également être associés à un phénomène extérieur (les instructions "with...use" autorisent ce genre de manipulations), désigner une condition d'erreur ("failures" dans le langage Esterel, "exceptions" en ADA), être un moyen de communication symétrique ou non, être un moyen de synchronisation facile à mettre en oeuvre, être facile à sauvegarder (par le biais de files d'attente par exemple); finalement, ils peuvent être émis et reçus de diverses manières:

i) leur émission peut être réalisée de manière synchrone avec un délai de réception, synchrone sans délai de réception, synchrone à réception instantanée, asynchrone...

ii) leur réception peut, quant à elle, être faite de façon synchrone avec attente limitée dans le temps, elle peut déclencher une interruption (voir la construction "do i upto s" du langage Esterel) on déclencher l'exécution d'une instruction spécifique...

En somme, les évènements constituent une classe d'objets riche en propriétés, dont l'abondance permet de donner une idée de la puissance d'expression d'un langage temps-réel, ainsi que de sa finesse de découpage du temps. (Permettons nous de relever une construction impossible à réaliser en ADA, et issue du modèle numéro 3: l'instruction interruptible par un évènement, encore appelée "upto":

do i upto s(X).

Elle permet d'éxécuter l'instruction i jusqu'à ce qu'apparaisse le flash s. Dans ce cas, son éxécution est interrompue et se poursuit après l'instruction "upto").

La seconde est un type particulier d'évènements, souvent manipulée par des instructions spéciales et plus riches que celles traitant les évènements normaux (il est par exemple possible de les dénombrer, d'en obtenir la valeur courante (cf. 6ème approhe), de les échantillonner,...). Elle est presque toujours émise par une horloge extérieure au système (le modèle numéro 3 fait exception dans le sens où la seconde est un évènement comme les autres qu'il est nécessaire de lier à un processus émetteur, afin d'obtenir une

mesure pour celle-ci). Pour un langage de programmation donné, le nombre de propriétés des évènements dépend essentiellement de la classe de problèmes visée par celui-ci (le langage ADA, dans sa version standard, vise essentiellement des applications systèmes où un découpage fin du temps n'est ni souhaitable ni nécessaire; il est pourtant possible d'orienter ce langage vers des applications temps-réel en permettant l'interfaçage de celui-ci avec des routines écrites en langage assembleur, en forçant les délais à être respectés avec exactitude, en réduisant le temps consacré à la gestion des processus...).

5) CINQUIEME CRITERE: Simplicité et naturel d'expression d'un formalisme.

Il est évident que chaque langage de programmation évolué possède une orientation propre, mais certaines méthodologies sont plus ou moins agréables d'utilisation que d'autres: la manipulation de variables évènements, considérées comme des suites d'instants, l'utilisation d'une machine infiniment rapide, le mélange de différents types d'évènements (tel que dans l'instruction Esterel "during 100 steps and at most 50 seconds do"),... nécessitent une certaine habitude.

A l'opposé de ces langages se trouvent des langages plus évolués (ADA, Occam...) plus faciles d'apprentissage, mais dont la version étendue comporte parfois une quantité impressionnante de détails, d'exceptions, de règles,... A notre avis, il est cependant clair qu'une certaine orthogonalité dans les concepts temps-réel est souhaitable (Esterel semble indiquer la bonne direction), ainsi qu'une sémantique claire et non-ambigue de ceux-ci (la version 1981 du langage ADA a montré à quel point un concept primordial tel que

celui de la terminaison des tâches peut être flou dans sa définition).

6) SIXIEME CRITERE: L'aptitude d'un langage à une complète axiomatisation et à la preuve.

Les approches du temps-réel considérées peuvent être aisément classifiées en trois groupes:

i) - nous avons tout d'abord celles proposant une sémantique (souvent opérationnelle) à un langage temps-réel donné. Leur aptitude à la preuve de propriétés de programmes ainsi qu'à une axiomatisation n'est plus à démontrer. (Il s'agit des catégories 1, 4 et 5).

ii) - nous distinguons encore les approches spécificatrices des contraintes temps-réel (le modèle numéro 6 en est le meilleur exemple), permettant de vérifier l'adéquation d'une implantation donnée à ces contraintes. Elles ne nous paraissent guère satisfaisantes pour effectuer des preuves de programmes, mais leur point fort est de faire un grand pas depuis la spécification informelle d'un problème vers une spécification mathématique claire et concise.

iii) - il reste les approches numéro 3 et 6 pour lesquelles un cadre formel plus puissant reste à trouver, afin de prouver des propriétés de programmes temps-réel, éventuellement de façon semi-automatique. La logique temporelle semble être profitable pour celà, comme le montre le papier [49] de Moszkowski.

Nous allons maintenant donner quelques détails concernant le plan de notre exposé.

Le chapitre II contient quelques rappels relatifs au langage de programmation ADA. Nous nous y concentrons sur les instructions manipulant le temps ainsi que sur les moyens de communication entre les différentes tâches d'un programme. Puis nous définissons le sous-langage de ADA choisi pour la suite de nos travaux, ainsi que les hypothèses supplémentaires relatives à la notion du temps. Ensuite nous nous intéressons aux problèmes spécifiques au temps-réel.

Dans le chapitre III, nous proposerons une sémantique opérationnelle de ce sous-ensemble de ADA, permettant de construire une automate non-déterministe simulant ce langage. Cet automate montrera comment satisfaire l'hypothèse de parallélisme maximal dans un contexte de machine multiprocesseur.

Le chapitre suivant est une étude d'un système de preuve de propriétés temps-réel du type "jusqu'à", noté A.R.T.. A cette fin nous définirons au préalable un ensemble d'opérateurs de fatalité permettant d'exprimer des propriétés temps-réel. Celui-ci contiendra les opérateurs classiques de la logique temporelle, adaptés à notre modèle, ainsi que de nouveaux opérateurs de fatalité tenant compte de la dimension supplémentaire des assertions de notre langage, à savoir le temps.

Le chapitre V étudiera une classification des propriétés de fatalité, exprimables à l'aide des opérateurs définis préalablement. On y explicitera également une méthode de raisonnement, caractéristique du domaine du temps-réel, afin de prouver ces propriétés dans le système axiomatique A.R.T..

Afin d'illustrer les propos du chapitre précédent, nous détaillerons dans le chapitre VI quelques preuves de propriétés de

programmes ADA.

Notre exposé se termine par une conclusion, suivie de la bibliographie. Quelques directives futures seront données pour la suite de nos recherches.

LE LANGAGE DE PROGRAMMATION ADA - RAPPELS
- HYPOTHESES DE TRAVAIL

CHAPITRE II

LE LANGAGE DE PROGRAMMATION ADA - RAPPELS - HYPOTHESES DE TRAVAIL.

Dans ce chapitre, nous allons rappeler les principales caractéristiques du langage ADA qui seront utiles dans l'étude ultérieure d'une sémantique opérationnelle. Nous nous servirons pour celà du manuel de référence [19] édité en décembre 1980, tout en tenant compte des modifications réalisées afin de parvenir à sa version de 1983. L'ensemble des instructions auxquelles nous allons nous intéresser, modulo certaines hypothèses de travail, constituera de part sa syntaxe le sous-ensemble de ADA qui nous sera utile par la suite: il s'agit du sous-ensemble minimal, nécessaire à l'expression de problèmes parallèles temps-réel. (On trouvera sa grammaire complète en appendice).

Dans un premier temps, nous indiquerons quelles sont les instructions séquentielles de notre langage de base, puis nous montrerons grace au concept de tâches, comment une application peut être décomposée en entités indépendantes. Nous étudierons ensuite la notion de temps, ainsi que les moyens de communication et de synchronisation existant entre ces tâches.

Rappelons les principaux traits du langage ADA, qui en font un outil de programmation scientifique universel:

1) Le typage des objets, employé dans toute sa puissance, établit des liens très forts entre ceux-ci et permet de détecter plus d'erreurs à la compilation.

2) La programmation, dans un environnement unifié et adapté au langage, facilitée grâce à des mécanismes d'encapsulation et de compilation séparée lui confèrent une qualité d'outil logiciel unique en son genre.

3) La manipulation des erreurs de programmation donne à l'utilisateur la possibilité de les contrôler et de les rattraper de manière efficace.

4) ADA a été conçu pour une programmation de problèmes parallèles et possède des primitives de synchronisation et de communication unifiées et faciles d'emploi.

5) Il existe également un mécanisme d'abstraction pour les données: la spécification de types de données et la spécification de leur représentation séparent les propriétés abstraites des données de leur réalisation concrète.

6) La généricité est employée afin de pouvoir paramétriser des unités de programmation (packages ou procédures) sur des espaces de données appropriés et afin d'instancier des librairies (ou bibliothèques) prédéfinies.

II.A LES STRUCTURES DE CONTROLE SEQUENTIEL.

Traitons la partie la plus classique dans les langages de programmation,

"commune" à tous les langages de la même famille que Pascal.

II.A.i L'instruction D'affectation.

Elle permet de ranger la valeur d'une expression de type entier ou réel dans une variable de même type. On n'autorisera aucune conversion de types et on ne s'intéressera jamais aux problèmes de précision des objets réels. Les opérateurs employés dans ces expressions seront, soit prédéfinis (pris dans l'ensemble: {+, -, *, /, **, rem, mod}), soit extraits de bibliothèques prédéfinies (on emploiera par exemple les fonctions mathématiques classiques: LN, COS, SIN, EXP).
La définition des premiers, voir chapitre IV de [19], est étendue aux ensembles IN et IR. Les seconds seront pris dans leur sens mathématique habituel.

II.A.ii L'instruction Conditionnelle.

La syntaxe de celle-ci est:

"if E then I end if"

où I est une instruction séquentielle et E une expression booléenne. Dans cette dernière (ainsi que dans toutes les conditions booléennes employées dans ce langage) on utilisera bien sur des opérateurs de relation (de l'ensemble {<, <=, >, >=, =, /=}), ainsi que les connecteurs logiques classiques (pris dans {or, not, and, xor}).

II.A.iii L'instruction Répétitive.

La syntaxe de cette instruction est:

"while E loop I end loop"

où on suppose que I est une instruction séquentielle et E une expression booléenne définies comme ci-dessus.

II.A.iv L'instruction Composée.

Il est possible de composer séquentiellement deux instructions séquentielles en les juxtaposant.

II.A.v L'instruction "null".

Lorsque un processus exécute cette instruction, il ne se produit rien.

II.B LA NOTION DE TACHES ET LEURS DIFFERENTS ETATS.

Une tâche est une entité d'un programme capable de s'exécuter en parallèle avec d'autres tâches. Celles-ci peuvent posséder des points d'entrée appelables par d'autres tâches. Deux tâches peuvent être synchronisées: lorsque l'une d'entre elles "appelle" un point d'entrée de la seconde et lorsque cette dernière accepte cet appel on dit qu'il y a rendez-vous. Ces points d'entrée constituent également le principal moyen de communication entre les tâches.
D'un point de vue programmation, une tâche est constituée de deux parties distinctes: sa spécification et son corps (cf. [19] chapitre 9). Nous ne conserverons, pour notre part, que la seconde et supposerons que la syntaxe d'une tâche est:

task <nom de tâche> is
<zone déclarative>

begin
I ;
end <nom de tâche> ;

où I est une instruction ou une suite d'instructions séquentielles et où la zone déclarative ne contient que des déclarations de variables du type entier ou réel, ainsi qu'une éventuelle initialisation.
Ainsi un programme est constitué (pour notre part) d'une suite finie de tâches juxtaposées.
En pratique, la notion de tâche est beaucoup plus riche en propriétés mais les quelques unes mentionnées ci-dessus suffisent pour nos applications. Citons tout de même quelques caractéritiques marquantes:

1) Il est possible de déclarer un type de tâche, afin d'obtenir par exemple une famille de tâches de même forme, de taille quelconque.

2) Il est possible de créer une tâche par allocation dynamique.

3) Un programme peut contenir plusieurs niveaux de parallélisme, respectant certaines règles précises au niveau de l'élaboration, de l'activation, de l'exécution et de la terminaison (voir [19] pour la définition claire de ces concepts).

II.C LA GESTION DU TEMPS EN ADA.

ADA propose un certain nombre de types et de primitives permettant la prise en compte du temps-réel (mesuré par une horloge) ou du temps simulé. Le type prédéfini "DURATION" permet de manipuler des intervalles de temps en fractions de secondes. Une tâche est susceptible de suspendre son exécution de x secondes au moins, par l'instruction:

"delay x"

Cette instruction ne tient pas compte des dérives qui peuvent se produire sur des requêtes à l'horloge, en programmation temps-réel.
Il est possible de programmer des actions périodiques, des chiens de garde, malgré celà, en faisant en sorte que la durée moyenne de suspension soit respectée. Dans l'exemple suivant, on exécute l'instruction I avec une période de 30 secondes en moyenne (ceci à condition que la durée d'exécution de I soit inférieure à 30 secondes).

```
exemple:
        task P is
                intervalle : duration := 30.0 ;
                prochain : time :=clock() + intervalle ;
        begin
                while true loop
                        delay prochain - clock() ;
                                I ;
                        prochain := prochain + intervalle ;
                end loop ;
        end P ;
```

- Figure 2 -

Dans notre étude, nous supposerons des délais comme étant exacts et l'argument des instructions de délai comme étant des réels (notés "float" pour des raisons de cohérence avec ce qui précède).
Le paquetage "CALENDAR" défini le type privé "time", la fonction "clock" qui

donne l'heure absolue, ainsi qu'un ensemble de fonctions (YEAR, MONTH, DAY, SECOND,...) et d'opérateurs (+, -) afin de manipuler les notions de temps.

II.D LES MOYENS DE COMMUNICATION ENTRE TACHES.

Malgré l'existence officielle d'une méthode de communication par variables partagées, notre étude ne s'intéressera qu'à la notion de "tranfert de messages" par rendez-vous. La première, quant à elle, est souvent qualifiée de marginale et est peu recommandée.

II.D.0.a Définition:

On dit qu'il y a rendez-vous entre une tâche T1 dite appelante et une tâche T2 dite appelée lorsque T1 appelle une entrée e de T2 et que T2 accepte cet appel. Le rendez-vous est terminé lorsque les actions associées à une instruction accept "sur" le point d'entrée e ont été exécutées. Chaque tâche reprend alors son exécution.
Fin de la définition.

Ce mécanisme

1) - assure la communication synchrone d'informations typées entre tâches,

2) - permet de résoudre des problèmes de synchronisation.

II.D.i La Notion De Point D'entrée.

Une point d'entrée est déclaré dans la partie spécification d'une tâche et est restreint à cette tâche. Une telle déclaration ressemble à une spécification de procédure. L'appel d'un point d'entrée se fait par une

instruction semblable à un appel de procédure.

II.D.ii L'instruction D'acceptation De Point D'entrée.

Cette instruction mentionne le point d'entrée, ses paramètres formels, ainsi qu'une suite d'instructions à exécuter lors de la prise en compte d'un rendez-vous. Une telle instruction ne peut apparaître que dans le corps de la tâche où elle est déclarée. Donc une tâche ne peut accepter de rendez-vous que sur ses points d'entrée. Lors de l'appel d'un point d'entrée, le rendez-vous a lieu immédiatement lorsque la tâche appelante est en attente sur l'entrée, sinon l'appelante est mise en attente et la demande est mémorisée dans une file FIFO associée au point d'entrée. De même, lorsqu'une tâche atteint une instruction "accept", la plus ancienne des demandes est prise en compte, si elle existe, sinon elle se met dans un état d'attente. (Remarquons encore qu'une tâche peut posséder plusieurs instructions accept désignant la même entrée). Un rendez-vous se termine lorque l'instruction accept atteint son "end" final: à ce moment, les paramètres de retour sont transférés dans la tâche appelante.

Par rapport à la syntaxe officielle de cette instruction, nous imposerons une contrainte supplémentaire: les paramètres d'entrée (de type "in") précèderont les paramètres de sortie (de type "out") sur un point d'entrée donné.

On déclarera un point d'entrée par une instruction de la forme:

entry e(u -> : in ⟨type entrée⟩ ; v -> : out ⟨type sortie⟩)

Les spécifications des paramètres formels seront identiques dans les instructions "accept" associées à e. En résumé, on peut schématiser un

couple de tâches susceptible de communiquer sur le point d'entrée e par:

```
task T1 is
        .
        .
        .
begin
        .
        .
        T2.e(a1,a2,...,an,b1,...bp);
        .
        .
        .
end T1;
task T2 is
        .
        .
        .
begin
        .
        .
        accept e(u1:in t1,...,un:in tn;
                 v1:out t1',...,vp:out tp') do
                 I ;
        end e;
        .
        .
        .
end T2;
```

- Figure 3 -

II.D.iii Les Interruptions.

Le concept de rendez-vous permet la liaison de tâches ADA à des dispositifs physiques ou à des programmes de bas niveau réalisant l'interface nécessaire. Une interruption peut être assimilée à l'appel d'une entrée d'une tâche utilisateur. Pour celà, le langage offre une construction permettant d'affecter des entrée aux classes d'interruption définie sur un matériel donné. (Voir la construction "with...use" dans le chapitre 13 du manuel de référence).

II.E LES INSTRUCTIONS "SELECT".

Sous ce nom sont regroupées plusieurs constructions permettant des rendez-vous conditionnels dans la tâche appelée ou dans la tâche appelante.

II.E.i L'instruction "select" Coté Appelée.

Il s'agit d'une instruction structurée dont les composantes sont des "gardes" suivies de "branches".
Une "garde" est une expression booléenne. Il existe trois types de branches:

1) des branches de rendez-vous contenant une instruction "accept",

2) des branches temporisées commençant par une instruction de délai,

3) des branches de terminaison réduites au mot clé "terminate".

Nous ne considérerons, dans notre sous-langage, que les deux permiers types.
Voyons comment fonctionne une telle instruction:

1) Tout d'abord on évalue les gardes afin de déterminer quelles branches sont ouvertes,

2) puis on évalue les expressions de délai figurant dans les branches temporisées.

3) Maintenant, le fonctionnement diffère selon la nature des branches ouvertes. Il faut distinguer trois cas:

i) + L'attente simple: ceci correspond au cas où seules des branches de rendez-vous sont ouvertes. Si au moins une des branches est franchissable (i.e. une branche au moins est ouverte et au moins un appel est en attente sur le point d'entrée associé), alors l'une d'entre elles est choisie de façon arbitraire, le rendez-vous correspondant est pris en compte, l'instruction "accept" et les instructions suivantes de la branche sont exécutées. Sinon, la tâche attend jusqu'au premier appel de l'un des points d'entrée situé dans une branche ouverte.

ii) + L'attente limitée dans le temps: correspond au cas où au moins une branche temporisée est ouverte. Dans ce cas, l'exécution se déroule de la même manière que ci-dessus, à la différence près qu'un délai d'attente maximal (qu'est le minimum des délais apparaissant dans les branches temporisées ouvertes) est donné. Une fois ce délai écoulé, si aucun rendez-vous n'a lieu, on exécute les instructions de la branche temporisée indiquant ce délai maximal.

iii) + Si aucune branche n'est ouverte on considère que l'instruction "select" est terminée et on poursuit en séquence (ceci n'est pas le cas dans la version officielle du langage, où on détecte alors une erreur dans la tâche contenant l'instruction).

Remarque: Dans [19], chapitre 9, on considère l'instruction "select" côté appelée dans toute sa généralité, et on notera la présence de branches spéciales appelées "terminate", ou de parties "else" dans celle-ci. Mais nous pensons avoir conservé toute la puissance

d'expression, malgré les restrictions, et celà dans le cadre de programmes à un degré de parallélisme.

II.E.ii L'instruction "select" Coté Appelante.

Celle-ci recouvre deux utilisations: la demande de rendez-vous immédiat et l'appel limité dans le temps. Nous ne conserverons que la seconde d'entre elles. Une instruction d'appel limité dans le temps a la forme suivante:

```
select
                -> ->
        T.e(u;v);
        I1;
or
        delay x;
        I2;
end select;
```

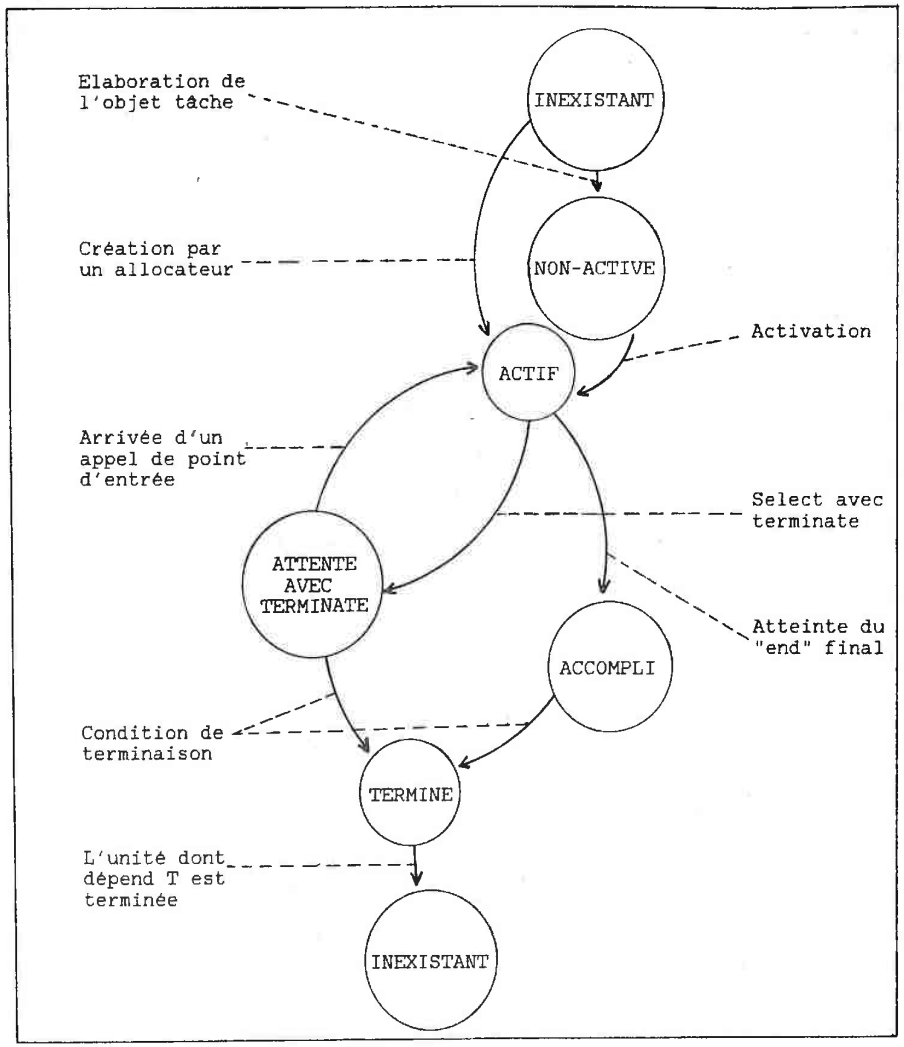
Elle introduit une demande de rendez-vous sur l'entrée e de la tâche T. Si au bout de x secondes la tâche T n'a pas acceptée cet appel, celui-ci est annulé et l'instruction I2 est exécutée. Sinon, l'exécution se poursuit avec l'instruction I1, une fois le rendez-vous terminé.

II.F LE CYCLE DE VIE D'UNE TACHE.

A des fins de synthèse, nous allons résumer (figure 4) le cycle de vie d'une tâche ADA, prise dans le sens de [19].
Toute déclaration d'objet tâche T crée une nouvelle tâche. Les entrées sont installées et peuvent alors être appelées: on dit que T est dans l'état non-activé. L'activation ultérieure de la tâche (qui consiste en l'élaboration de la partie déclarative du corps du type tâche et qui est réalisée autant de fois qu'il y a d'objets tâche de ce type déclarés) la fait passer dans l'état actif. L'exécution de son corps est lancée.

Lorsque cette dernière atteint son "end" final, elle passe dans l'état accompli (l'attribut T'COMPLETED devient vrai), puis dans l'état terminé. Lorsqu'une tâche exécute une instruction d'attente sélective comprenant une branche de terminaison ouverte et aucune branche de rendez-vous franchissable, son état devient "en attente avec terminate". Elle retourne à l'état actif dès qu'un appel intervient sur une branche ouverte et passe brièvement par l'état terminé lorsque toutes les conditions de terminaison se trouvent être remplies. Finalement, une tâche passe de l'état terminé à l'état inexistant dès que l'unité dont elle dépend est terminée.

Tous ces états possèdent, comme on peut le constater, un nom indiquant l'opération physique en machine à réaliser pour parvenir à eux. (Par exemple, le passage à l'état actif d'une tâche T donnée peut signifier son enfilement dans une file d'attente de processus activables, cette dernière étant une donnée d'un ordonnanceur de tâches; de même le passage à l'état inexistant peut être considéré comme la suppression d'un processus de la mémoire virtuelle du système d'exploitation).



- Figure 4: CYCLE DE VIE D'UNE TACHE T -

II.G CONCLUSION.

Cette brève étude des concepts temps-réel et de ceux liés au parallélisme nous donne à ce stade une première vue du fonctionnement des programmes ADA, indispensable pour le chapitre suivant. Nous avons décrit de manière informelle quels peuvent être les différents états d'une instruction (en particulier d'un "select") et d'une tâche. Ces traits du langage seront, dans le contexte du sous-ensemble choisi, respectés le mieux possible dans les sémantiques formelles.

Les problèmes relatifs aux algorithmes parallèles tels l'absence de blocage, l'absence de famine,... ont fait l'objet d'études approfondies dans le cadre de langages tels que CSP, MODULA 2, PASCAL-CONCURRENT. Nous ne nous préoccupons guère par la suite de ce genre de problèmes, typiquement liés à la programmation concurrente.

Une petite note sur la notion d'arbitraire s'impose. Ce terme, souvent en relation avec la notion de programmes erronnés, indique qu'une grande liberté est accordée aux écrivains de compilateurs dans les stratégies réalisant des choix qualifiés comme tels. Mais, on met en garde le programmeur qu'un algorithme reposant sur ces stratégies doit être considéré comme erronné (cette définition est extraite de [19]). Cependant, en ce qui concerne la sélection d'une branche franchissable d'une instruction "select côté appelé", il aurait été préférable de la réaliser de manière aléatoire. Ceci garantirait l'absence de famine d'une branche d'un "select".

SEMANTIQUE OPERATIONNELLE DES PROGRAMMES ADA

CHAPITRE III

SEMANTIQUE OPERATIONNELLE DES PROGRAMMES ADA.

III.A INTRODUCTION.

Dans ce chapitre, nous allons étudier une sémantique opérationnelle des programmes ADA temps-réel parallèles, non-déterministes, modélisant l'exécution de ces derniers sur une machine multiprocesseurs et assurant un parallélisme maximal.

Nous supposerons ainsi, que chaque tâche d'un programme possède un processeur personnel ainsi qu'une mémoire propre. Ces processeurs peuvent communiquer entre eux par un certain nombre de canaux (dont la réalisation matérielle nous intéresse peu, mais pouvant être par exemple une suite de bus ou de processeurs supplémentaires), simulant l'activité des points d'entrée. Nous admettrons également que ces calculateurs élémentaires ont un même temps de cycle noté θ, pendant lequel ils sont capables d'exécuter certaines actions atomiques élémentaires (par exemple une affectation, une opération arithmétique à arguments entiers ou réels, l'évaluation d'une expression booléenne simple...).

Après avoir défini la notion d'état d'un programme, nous formaliserons la notion de sémantique abstraite. Celle-ci nécessite la définition d'une relation de transition entre états, que nous détaillerons ensuite. Dans

cette denière nous utiliserons une méthode d'étiquetage des instructions à la LAMPORT, qui sera définie en même temps que les différents types de transitions.

Remarquons tout d'abord, que pour un type de machine donné, nous pouvons construire une fonction récursive, associant à toute expression bien formée, arithmétique ou booléenne, son temps d'exécution. Supposons que nous nous restreignons à des objets entiers ou réels (constantes ou variables) on peut définir cette fonction pour notre machine abstraite de la manière suivante:

f : EXPRESSIONS ---> IR+

$\forall$ E $\in$ IR $\cup$ IN $\cup$ $\mathcal{V}$,

f(E) = 0

$\forall$ E1,E2 $\in$ EXPRESSIONS, $\forall$ op $\in$ {+, -, <, $\leq$, >, $\geq$, =, /=, and, or, xor, rem, mod},

f(E1 op E2) = f(E1) + f(E2) + θ

$\forall$ E $\in$ EXPRESSIONS, $\forall$ op $\in$ {+, -, not},

f(op E) = f(E) + θ

$\forall$ E1,E2 $\in$ EXPRESSIONS, $\forall$ op $\in$ {*, /},

f(E1 op E2) = f(E1) + f(E2) + 2θ

$\forall$ E1,E2 $\in$ EXPRESSIONS,

f(E1 ** E2) = f(E1) + f(E2) + 10θ.

($\mathcal{V}$ représentant l'ensemble des variables d'un programme donné).

Cette définition suppose bien entendu que les expressions soient correctement typées, tel qu'il est exigé dans [19], et suppose également qu'il n'est jamais nécessaire de faire des conversions entre types numériques.

Elle n'est pas limitative, dans la mesure où il est possible de rajouter des

opérateurs nouveaux nécessaires pour certains calculs et impossibles à réaliser à l'aide de ceux indiqués dans la définition de f. Nous venons de considérer pour f, une fonction très particulière, mais satisfaisante si l'on regarde les temps indiqués par les constructeurs de microprocesseurs ou d'ordinateurs. Tout autre modèle de fonction pour f conviendrait également, à condition qu'il respecte un certain réalisme relativement à la complexité des algorithmes de calculs numériques employés en pratique, dans l'écriture de bibliothèques de fonctions scientifiques (voir [32] par exemple). Dans la suite de notre étude, nous considérons que θ = 1E-03 secondes lorsque sa connaissance est nécessaire.

III.B DEFINITIONS.

Nous considèrerons qu'un programme ADA est une suite finie de n tâches et nous noterons IP = (T1//...//Tn).

III.B.i Définition:

Soit IP = (T1//...//Tn) un programme ADA.

Un état du programme IP est un quadruplet s=<t,c,m,D> où

1) t désigne la date courante relativement à l'horloge maîtresse du système,

2) c = <c1,...,cn> est un multi-point de contrôle où $\forall$ i $\in$ {1,...,n}, ci désigne la valeur du compteur ordinal de la tâche Ti,

3) m = ⟨m1,...,mn⟩ est l'état mémoire du système. On considère que ∀ i ∈ {1,...,n}, mi est une application de $\mathcal{V}$i dans IR⊕IN, où $\mathcal{V}$i désigne l'ensemble des noms de variables employées dans Ti et IR⊕IN est le treillis résultant de la somme coalescente des treillis réels et entiers. (Voir STOY [58]). On supposera que les $\mathcal{V}$i sont disjoints deux à deux.

4) D est un ensemble de tâches en train d'exécuter une action atomique n'ayant fini ce travail à l'instant t. Formellement, D = {(T,u)/ T n'a pas fini d'exécuter la dernière instruction atomique, T ∈ {T1,...,Tn} et T nécessite encore u unités de temps pour achever celle-ci, u ∈ IR+}.

Fin de la définition.

Nous noterons par S[IP] l'ensemble des états du programme IP.

III.B.ii Remarques:

- Pour un état s quelconque, nous noterons par s.t, s.c, s.m, s.D sa première, seconde, troisième, quatrième composante respective.
- La définition précédente mettait en jeu la notion d'action atomique. Toutefois, celle-ci ne peut être définie clairement que dans la sémantique opérationnelle abstraite. Mais nous pouvons tout de même indiquer, quels sont les types d'actions atomiques utilisés:

a) - l'affectation d'une valeur numérique à une variable,

b) - l'évaluation d'une condition booléenne,

c) - l'exécution d'un délai,

d) - l'évaluation d'une instruction "select",

e) - la mise en attente d'un appel de point d'entrée,

f) - l'acceptation d'un rendez-vous,

g) - la fin d'un rendez-vous,

h) - l'exécution d'une attente active.

III.B.iii Définition:

Soit IP un programme ADA. On appelle sémantique abstraite du programme IP le quadruplet ⟨S[IP],I[IP],t[IP],T[IP]⟩ où

1) S[IP] est l'ensemble des états du programme,

2) I[IP] est l'ensemble des états initiaux de IP,

3) t[IP] est une relation de transition sur S[IP], modélisant l'exécution de IP,

4) T[IP] est l'ensemble des traces de IP générées par les trois composantes ci-dessus. D'un point de vue mathématique, T[IP] est un ensemble de fonctions partielles de IR+ dans S[IP].

Fin de la définition.

III.B.iv Remarque:

Pour un programme IP donné, on peut définir l'ensemble des états initiaux de la manière suivante:

$I[IP] = \{\langle 0, c^I, m^I, \emptyset\rangle\}$ où $c = \langle c1,\ldots,cn\rangle$, ci désignant le premier point de contrôle de Ti, $\forall\, i \in \{1,\ldots,n\}$ et $m = \langle m1,\ldots,mn\rangle$, mi étant l'état mémoire initial de Ti (où toutes les variables peuvent être indéfinies:

$\forall\, v \in \mathcal{V}i\ ,\ mi^I(v) = \perp\ ),\ \forall\, i \in \{1,\ldots,n\}.$

Fin de la remarque.

III.B.v Définition:

Soient IP un programme ADA, et s un état de IP.

On dit que s est un état observable de IP quand

$\{T \mid \exists\, t > 0,\ (T,t) \in s.D\} \neq \{T1,\ldots,Tn\}$

Fin de la définition.

Intuitivement, un état est observable quand au moins une action atomique en cours d'exécution dans les différentes tâches vient de se terminer, les états initiaux étant également considérés comme observables.

III.B.vi Notation:

On désignera par S0[IP] l'ensemble des états observables de IP.

III.C LA RELATION DE TRANSITION t[IP].

Nous allons définir la relation de transition t[IP] entre états observables, puis nous l'étendrons à S[IP] tout entier.
Pour résoudre le premier de ces problèmes, on procèdera de manière analogue à l'élaboration d'une relation de transition dans une sémantique opérationnelle sur des structures discrètes.
Soit s un état observable de IP, $s=\langle t,c,m,D\rangle$.

1) Déterminons l'ensemble des tâches $\mathcal{T}[s]$ capables de s'exécuter à partir de l'état s, pendant une durée finie telle que l'état syntaxique (point de contrôle et état mémoire) des tâches non-élément de $\mathcal{T}[s]$ restent constant pendant cette durée. Cette dernière devra être la plus grande possible.
En l'état s, nous associons à chaque tâche Ti ($i \in \{1,\ldots,n\}$) les trois choses suivantes:

a) deux points de contrôle, notés ci et ci', dont le second est le successeur du premier (on peut supposer, grace à la numérotation de LAMPORT, qu'il existe une relation de succession entre les points de contrôle d'une tâche donnée, calculable par analyse syntaxique),

b) un réel di défini par:

di = si $\nexists y \in \mathbb{R} / (Ti,y) \in s.D$
alors la durée d'exécution de l'action atomique placée entre les points de contrôle ci et ci'
sinon l'unique réel y tel que $(Ti,y) \in s.D$
fsi

Intéressons nous au minimum de ces réels juste définis.

Soit

$$M = \min_{j=1}^{n} dj,$$

et posons alors

$$\mathcal{T}[s] = \{Tj \;/\; dj = M = \min_{i=1}^{n} di \}.$$

Nous venons ainsi de déterminer l'ensemble des tâches se déplaçant au cours de la prochaine transition commençant en s. Les points de contrôle des tâches n'y figurant pas, resteront constants par définition même de $\mathcal{T}[s]$.
Remarquons encore que dans $\mathcal{T}[s]$, on peut trouver des couples de tâches susceptibles d'entrer en rendez-vous (à cause des choix non-déterministes à réaliser dans certains cas) pendant la prochaine transition.

2) Maintenant, cherchons comment construire la quatrième composante de l'état successeur de s. Soit s' cet état.

a) Supprimons de s.D l'ensemble des couples (T,M) tels que $T \in \mathcal{T}[s]$. De cette manière nous indiquons que les tâches de $\mathcal{T}[s]$, figurant dans s.D, ne sont plus en "retard" sur l'instant courant s'.t.

b) Rajoutons ensuite l'ensemble des tâches, dont la présente transition a provoqué par son exécution un "retard" sur la date courante, c'est à dire qu'il n'a pu être possible de prendre en compte.
Il s'agit précisément de l'ensemble
{(Tj,dj) / (∀ y ∈ IR, (Tj,y) ∉ s.D) et Tj ∈ $\mathcal{T}[s]$}.
En effectuant cette démarche, nous simplifions la vision que

nous possédons sur les tâches "en cours d'exécution" (i.e. celles qui ne sont pas élément de $\mathcal{T}[s]$), en supposant que leur point de contrôle ainsi que leur état mémoire restent inchangés. Ceci n'est qu'une modélisation approximative d'un ensemble d'activités très éparses et dont la finesse nécessaire à leur description n'est guère intéressante. (Tous ces processus peuvent avoir des activités fort diverses comme: la recherche d'une adresse en mémoire, le calcul d'une sous-expression arithmétique...).

c) Finalement, il nous faut tenir compte de la nouvelle date en s' (égale à s'.t = s.t+M), en soustrayant à la seconde composante des éléments de l'ensemble obtenu ci-dessus, la valeur M.
(On démontre aisément que la seconde composante de ces éléments est toujours positive, quelle que soit la transition effectuée).

En résumé, on peut définir une relation sur S0[IP] notée)-IP->.

III.C.i <u>Définition:</u>

∀ s,s' ∈ S0[IP], s)-IP-> s' si et seulement si

$$[(s \neq s') \wedge succ(s.c,s'.c) \wedge \mathcal{T}[s](s.m,s'.m) \wedge$$

$$s'.D=((s.D \cup \bigcup_{j=1}^{n} \{(Tj,dj)\})\setminus\{(T,u)/\ u=s'.t-s.t,\ T \in \{T1,\ldots,Tn\}\})$$

$$\alpha\ (s'.t-s.t) \wedge$$

$$(\forall\ s'' \in S[IP],\ (s'' \neq s' \wedge s.t \leq s''.t < s'.t)$$

$$\Rightarrow$$

$$(s''.c = s.c \wedge s''.m = s.m \wedge$$

$$s''.D = \{(T,u)/\ \exists\ u',\ (T,u') \in s.D \wedge$$

u=u'-(s".t-s.t)}))]

Fin de la définition.

où nous avons employé les notations suivantes:

1) "succ" désigne une relation de succession entre points de contrôle (nous la définirons au paragraphe suivant),

2) ℭ[s] est une relation de transformation sur les états mémoire, modélisant l'effet de l'exécution d'une action atomique de chacune des tâches de 𝒯[s] sur les mémoires locales de ces dernières,

3) ∝ est un opérateur soustrayant à la seconde composante des éléments de son premier argument, son second opérande.

III.C.ii Définition: La Relation De Transition t[IP].

∀ s,s' ∈ S0[IP], t[IP](s,s') si et seulement si
s >-IP-> s'.
∀ s" ∈ S[IP] \ S0[IP],
(∃ s,s' ∈ S0[IP], s >-IP-> s' ^ s.t < s".t < s'.t)
=>
(t[IP](s,s") ^ t[IP](s",s')).
Fin de la définition.

Notation:
Nous désignerons par t*[IP] la fermeture transitive de la relation t[IP].

III.C.iii Définition: L'ensemble Des Traces D'exécution T[IP].

Soient IP un programme ADA, S[IP] l'ensemble des états de IP,

S0[IP] l'ensemble des états observables de IP, t[IP] la relation de transition associée à IP et définie dans la définition précédente.
T[IP] est l'ensemble des fonctions
w : IR+ ---> S[IP]
vérifiant

i) ∀ x ∈ IR+, w(x).t = x+K pour un certain K ∈ IR+,

ii) ∀ x ∈ IR+, [(w(x) ∈ S0[IP]) ^
(∃ y >x, w(y) ∈ S0[IP] ^ ∀ z ∈]x,y[,
w(z) ∉ S0[IP])]
=>
[t[IP](w(x),w(y))].

On appelle K le décalage horaire associé à w.
Fin de la définition.

III.C.iv Définition:

Une trace d'exécution w d'un programme IP est dite initiale si w(0) ∈ I[IP].
Une trace d'exécution w appartient à T[IP] commence en l'état s ∈ S[IP] si w(0)=s. L'ensemble des traces de T[IP] commençant en s est noté T[IP](s).
Fin de la définition.

III.C.v Proposition:

Soient IP un programme ADA, ⟨S[IP],I[IP],t[IP],T[IP]⟩ la

sémantique abstraite associée à IP.

On a ∀ w ∈ T[IP], ∀ x ∈ IR+, t*[IP](w(0),w(x)).

Fin de la proposition.

La démonstration de cette proposition est triviale, il suffit d'utiliser pour celà les définitions précédentes.

Afin de parfaire la définition de la relation de transition t[IP] associée à IP, il nous reste à préciser les transitions élémentaires réalisables sur S0[IP].

III.D LES TRANSITIONS ELEMENTAIRES DE LA RELATION)-IP-(.

Considérons un programme ADA, IP=(T1//.../Tn).

Nous avons vu précédemment qu'une transition de la relation)-IP-(permettait la progression de plusieurs tâches simultanément. Dans la description détaillée d'une transition de la forme s)-IP)-(s' qui va suivre, nous n'expliciterons que les composantes relatives à un élément de $\mathcal{C}$[s] (ou de deux éléments de $\mathcal{C}$[s] lorsque nécessaire). Celà suffit pour définir complètement la relation)-IP-(, car afin de calculer la valeur de s', il suffit d'appliquer la méthode ci-dessous à toutes les tâches de $\mathcal{C}$[s] et de laisser inchangées les composantes de s relatives aux autres processus.

Nous ne préciserons plus, dans le présent paragraphe, la valeur de s'.D pour des raisons de simplicité d'écriture et car sa connaissance (déjà décrite en III.C) est triviale. Il en sera de même pour la première composante de s'.

III.D.i Notation:

Un état s=(t,c,m,D) sera souvent noté s=(t,@1,...,@n,D) avec

∀ i ∈ {1,...,n}, @i=(ci,mi).

Fin de la notation.

III.D.ii Proposition:

Soient IP un programme ADA, s ∈ S0[IP], s=(t,@1,...,@n,D), @j=(cj,mj) ∀ j ∈ {1,...,n},

soit w l'unique trace partielle commençant en l'état initial de IP et passant par s (sous réserve que I[IP] soit réduit à un élément), i ∈ {1,...,n}.

Alors ∃! s'=(t',@1',...,@n',D') où

∀ j ∈ {1,...,n}, @j'=(cj',mj') tel que

1) t' < t,

2) ci' = ci,

3) on a la propriété logique

Const(i)(s,s')=[∀s" ∈ S0[IP], s"=(t",@1",...,@n",D"), @j"=(cj",mj") ∀ j ∈ {1,...,n}, si s')-IP-)*s")-IP-)*s, on a ci"=ci]

(cette propriété indique qu'en tous les états situés entre s' et s, le point de contrôle de la tâche Ti est constant et égal à ci),

4) s' est le "premier" état avant s possédant la propriété ci-dessus i.e. ∀ h ∈ S0[IP] si (h ∈ w ^ h)-IP-)*s'), alors Const(i)(h,s') est faux.

Fin de la proposition.

Cette proposition affirme qu'il existe un unique état antérieur à s,

noté s', appartenant à w tel que:

a) le point de contrôle de la tâche Ti reste constant sur le chemin menant de s' à s,

b) l'état prédécesseur de s' noté pre(s') vérifie, sous hypothèse d'existence:
s".ci ≠ s'.ci

III.D.iii Définition:

Soient IP un programme ADA, s un état de IP accessible à partir de I[IP].
On définit pour tout i appartenant à {1,...,n} la fonction
Ei : S0[IP] ---> IR+
s ---> t-t'
où t' est la première composante de l'état s' associé à s dans la proposition ci-dessus.
Fin de la définition.

Cette fonction nous permet de connaître, pour un processus Ti donné, la date antérieure à la date courante s.t à laquelle Ti a progressé pour la dernière fois.
Considérons un programme ADA IP, s ∈ S0[IP],
s=⟨t,@1,...,@n,D⟩ avec ∀ j ∈ {1,...,n}, @j=(cj,mj) et soit Ti un élément de 𝒯[s].

III.D.iv La Transition Associée A L'instruction D'affectation.

Supposons que la tâche Ti ait atteint le point de contrôle ci, précédant une affectation, soit schématiquement


```
                .
                .
                .
        ci:
                x:=e ;
        ci':
                .
                .
                .
```

alors on a:

s ⟩-IP-⟩ ⟨t',@1',...,@(i-1)',(ci',mi[e/x]),...,@n',D'⟩

Remarquons que l'on a: M = min{dj/ j=1,...,n} ≤ θ+f(e).

III.D.v La Transition Associée A L'instruction "null".

Si le point de contrôle ci précède l'instruction "null" i.e.

```
                .
                .
                .
        ci:
                null ;
        ci':
                .
                .
                .
```

on a:

s ⟩-IP-⟩ ⟨t',@1',...,@(i-1)',(ci',mi),@(i+1)',...,@n',D'⟩

avec M ≤ θ.

III.D.vi Transitions Associées A L'instruction Itérative.

Supposons que Ti se trouve dans la situation suivante:

```
                .
                .
                .
        ci:
                while B loop
        ci':
                        S ;

                end loop ;
        ci":
                .
                .
                .
```

on obtient alors

```
s >-IP-> si B(mi) = ⊥ alors erreur
         sinon
                si B(mi) alors
                ⟨t',@1',...,@(i-1)',(ci',mi),...,@n',D'⟩
                sinon
                ⟨t',@1',...,@(i-1)',(ci",mi),...,@n',D'⟩
                fsi
         fsi
```

avec $M \leq f(B) + \theta$

Si par contre, Ti se trouve dans le cas représenté schématiquement par

```
                .
                .
                .
                while B loop
        ci':
                        S ;
        ci:
                end loop;
        ci":
                .
                .
                .
```

on peut écrire:

```
s >-IP-> si B(mi) = ⊥ alors erreur
         sinon
           si B(mi) alors
           ⟨t',@1',...,@(i-1)',(ci',mi),@(i+1)',...,@n',D'⟩
           sinon
           ⟨t',@1',...,@(i-1)',(ci",mi),@(i+1)',...,@n',D'⟩
           fsi
         fsi
```


où B(mi) est la valeur de l'expression B, obtenue en substituant aux variables de Ti leur valeur en l'état s. De même que précédement, on a $M \leq f(B)+\theta$.

III.D.vii Transition Associée A L'instruction Conditionnelle.

Supposons que Ti ait atteinte l'instruction représentée schématiquement de la manière suivante:

```
                .
                .
                .
        ci:
                if B then
        ci':
                        S ;
                end if;
        ci":
                .
                .
                .
```

alors

```
s>-IP-> si B(mi) = ⊥ alors erreur
        sinon
          si B(mi) alors
          ⟨t',@1',...,@(i-1)',(ci',mi),@(i+1)',...,@n',D'⟩
          sinon
          ⟨t',@1',...,@(i-1)',(ci",mi),@(i+1)',...,@n',D'⟩
          fsi
        fsi
```

avec $M \leq \theta$

III.D.viii Remarques:

1) Dans les transitions que nous venons de citer, l'évaluation d'expressions peut donner un résultat indéfini ou mathématiquement erronné (par exemple: 2/0 ou $\perp \leq x$ ou encore LN(-1)), dans quels cas on aboutit à un état particulier d'erreur.

2) Les traces d'exécution de nos programmes sont infinies car une fois que l'on a atteint le point de sortie du programme ou un état d'erreur, on se maintient dans le même état indéfiniment.

III.D.ix Transitions Associées A L'instruction De Délai.

```
        .
        .
        .
ci:
        delay E ;
ci':
        .
        .
        .
```

E étant une expression arithmétique à résultat réel, la transition associée est triviale:

s >-IP-> <t',@1',...,@(i-1)',(ci',mi),@(i+1)',...,@n',D'>

sa durée étant inférieure à E(s).

III.D.x Transition Associée A L'état D'erreur.

erreur >-IP-> erreur

Cette transition n'a aucune durée propre, car l'état d'erreur provoque l'arrêt d'un système réel, ou du moins le traitement de l'erreur si l'on se situe dans un cadre plus général (c'est le cas d'un langage où on autoriserait la manipulation des exceptions, par exemple).

III.D.xi Transition Associée A L'état Final.

Lorsque les tâches T1,...,Tn ont atteint leur point de sortie respectif, noté $c1^f, c2^f, \ldots, cn^f$, il se produit la transition que voici:

$$s \text{ >-IP-> } \underline{si} \bigwedge_{j=1}^{n} (cj = cj^f) \underline{alors}\ s\ \underline{fsi}$$

III.D.xii Transitions Associées A L'instruction "select" Coté Appelé.

Considérons la portion de programme que voici, issue de Ti:

```
                         .
                         .
              0          .
            ci:
                        select
    1                      .
   ci:                     .
                           .
                          or
                                       ->        ->
                        when bk => accept ak(uk:in ; vk : out) do
              2,k
            ci:                          1
                                         Sk ;
              3,k
            ci:
                                   end ak;
              4,k
            ci:                    2
                                   Sk ;
                        or
                          .
                          .
                          .
                        or
                        when wj => delay tj ;
              5,j
            ci:                    3
                                   Sj ;
                        or
                          .
                          .
                          .
                        end select ;
              6
            ci:         .
                        .
                        .
```

dont l'instruction "select" contient p branches de rendez-vous et q branches temporisées, et soient les assertions d'état

$$B(s) = \bigvee_{k=1}^{p} bk(s), \qquad C(s) = \bigvee_{j=1}^{q} cj(s)$$

posons

$$\Psi(s) = \begin{cases} \min_{j=1}^{q} \{tj / wj(s)\} & \text{si } \bigvee_{j=1}^{q} wj(s) \text{ est vrai,} \\ +\text{infini sinon.} \end{cases}$$

1) Premier cas: ci=c(i,0)

```
s >-IP-> s'= si ~B(s) ^ ~C(s) alors
                                 6
             <t',@1',...,@(i-1)',(ci,mi),@(i+1)',...,@n',D'>
             sinon
                                 1
             <t',@1',...,@(i-1)',(ci,mi),@(i+1)',...,@n',D'>
             fsi
```

Cette transition, conforme à la sémantique informelle de l'instruction "select" donnée au chapitre précédent, consiste à évaluer l'instruction en question. Elle a une durée maximale égale à θ, i.e. M≤θ. En l'état s', on connaîtra la valeur de di associée à la tâche Ti, grâce à celles de B(s') et C(s'). Elle est définie par le tableau suivant:

B(s')	C(s')	di(s')
Vrai	Vrai	θ
Vrai	Faux	θ
Faux	Vrai	Ψ(s')

On constate ainsi que lorsque B(s') est vrai, la tâche Ti se trouve dans une phase d'attente active (conformément à [19]). Dans le cas contraire, l'exécution se poursuivra en c(i,6). Tant que le point de contrôle de la tâche Ti se trouvera en c(i,1), il est possible de connaître, en un état s donné, les deux valeurs suivantes, capables de décider de la prochaine transition consernant Ti: la durée pendant laquelle ce point de contrôle se trouvait déjà en c(i,1), Ei(s), et la durée maximale pendant laquelle celui-ci se maintiendra en c(i,1).

III.D.xii.a <u>Notation:</u>

Le point de contrôle c(i,0) associé au "select" S, sera noté <u>wait</u>(s).

<u>Fin de la notation.</u>

2) <u>Second cas:</u> $c_i=c(i,1)$

i) <u>Premier sous cas:</u> $\sim B(s) \wedge C(s)$ est vrai.

$$s \;\text{>-IP->}\; \langle t',@1',\ldots,@(i-1)',(c_i^{6},m_i),@(i+1)',\ldots,@n',D'\rangle$$

Il est évident que la durée maximale de cette transition est $\gamma(s)$.

ii) <u>Second sous cas:</u> $(B(s) \wedge \sim C(s))$ est vrai et

$\exists\, k \in\{1,\ldots,p\},\ \exists\, j \in\{1,\ldots,n\},\ j\neq i$ tels que c_j=<u>at</u> <u>wait</u>$(T_i.a_k(\vec{e};\vec{f}))$ (où $\vec{e}$ et $\vec{f}$ sont des paramètres effectifs compatibles avec les paramètres formels $\vec{u}_k$ et $\vec{v}_k$ respectivement).

Supposons donc que la tâche Tj se trouve dans un état d'attente de rendez-vous "devant" une instruction d'appel de point d'entrée, et que le point de contrôle immédiatement suivant cj, noté RV:, indique un état d'attente de fin de rendez-vous, soit schématiquement:

.
.
.
cj:
RV: $T_i.a_k(\vec{e};\vec{f})$;
.
.
.

On a la transition,

$$s \;\text{>-IP->}\; s'= \langle t',@1',\ldots,@(j-1)',(RV(a_k),m_j),@(j+1)',\ldots$$
$$\ldots,@(i-1)',(c_i^{2,k},m_i[\vec{u}_k/\vec{e}]),\ldots,@n',D'\rangle$$

dont la durée maximale est de θ. Tj entre dans un état particulier, et nous convenons qu'en celui-ci on a dj=+infini pour des raisons évidentes.

iii) <u>Troisième sous cas:</u> $B(s) \wedge \sim C(s) \wedge$

$[\forall\, k \in \{1,\ldots,p\},\ \forall\, j \in \{1,\ldots,n\},\ j\neq i,$ $c_j\neq$<u>at</u> <u>wait</u>$(T_i.a_k(\vec{e},\vec{f}))]$

Lorsqu'aucune entrée de Ti, se situant dans une branche ouverte du "select" n'est appelée, cette tâche se maintient en attente active indéfiniment.

$$s \;\text{>-IP->}\; \langle t',@1',\ldots,@n',D'\rangle$$

$M\leq\theta$.

iv) <u>Quatrième sous cas:</u> $B(s) \wedge C(s) \wedge$

$[\forall\, k \in \{1,\ldots,p\},\ \forall\, j \in \{1,\ldots,n\},\ j\neq i,$ $c_j\neq$<u>at</u> <u>wait</u>$(T_i.a_k(\vec{e},\vec{f}))]$

Dans ce cas, il est nécessaire de vérifier si le temps $\gamma(s)$ s'est écoulé. Dans l'affirmative l'exécution se poursuit par S(j,3), sinon Ti reste en attente.

s >-IP-> <u>si</u> $E_i(s) < \gamma(s)-\theta_1$ <u>alors</u>

$\langle t',@1',\ldots,@(i-1)',(c_i,m_i),@(i+1)',\ldots,@n',D'\rangle$

<u>sinon</u>

$\langle t',@1',\ldots,@(i-1)',(c_i^{5,j}, m_i),@(i+1)',\ldots,@n',D'\rangle$

<u>fsi</u>

Ici encore $M\leq\theta$.

v) <u>Cinquième sous cas:</u> $B(s) \wedge C(s) \wedge$

$\exists\, k \in\{1,\ldots,p\},\ \exists\, j \in\{1,\ldots,n\},\ j\neq i$ tels que c_j=<u>at</u> <u>wait</u>$(T_i.a_k(\vec{e};\vec{f}))$

Dans le cas d'une attente active dans le temps où paraît un rendez-vous

avec Tj, on obtient:

$$s \;\text{>-IP->}\; s'=\langle t',@1',\ldots,@(j-1)',(RV(ak),mj),\ldots$$

$$\ldots,@(i-1)',(ci,\ mi[uk/e]),@(i+1)',\ldots,@n',D'\rangle$$

où RV(ak) est le point de contrôle de Tj immédiatement suivant cj, vérifiant succ(cj,RV(ak)), et où M≤θ. Nous conviendrons qu'en s' on a dj=+infini.

3) Troisième cas: ci=c(i,3,k).

Lorsque la tâche appelée atteint la fin de l'instruction "accept", elle termine son rendez-vous en retournant les paramètres de sortie à la tâche appelante. Pour savoir quelle est la tâche appelante il suffit de parcourir les @j, j ∈ {1,..n}, soit Tj cette tâche, et soit cj' le point de contrôle de celle-ci tel que succ(cj,cj').

$$s \;\text{>-IP->}\; \langle t',@1',\ldots,@(j-1)',(cj',mj[s\ /vk]),$$

$$@(j+1)',\ldots$$

$$\ldots,@(i-1)',(ci^{4,k},mi),@(i+1)',\ldots,@n',D'\rangle$$

III.D.xii.b Remarque:

Pour une instruction "select" de la forme ci-dessus, notée S, nous désignerons souvent le label fictif c(i,1) par wait(S).

Fin de la remarque.

III.D.xiii Transitions Associées A L'instruction D'appel De Point D'entrée.

Supposons que Ti contienne l'instruction suivante:

.
.
.

1
ci:

2 3
ci: ci:

Tj.a(e ;s);

4
ci:

.
.
.

Où

1) c(i,1) est l'étiquette précédant l'instruction, indiquant la fin de la précédente,

2) c(i,2) est une étiquette fictive marquant une attente de rendez-vous, souvent notée wait(Tj.a(e ;s)),

3) c(i,3) est une étiquette fictive matérialisant un état d'attente de fin de rendez-vous, souvent désigné par RV(a),

4) c(i,4) est le label suivant immédiatement l'instruction en question.

Si ci=c(i,1), on dit que l'instruction entre dans un état d'attente:

$$s \;\text{>-IP->}\; s'=\langle t',@1',\ldots,@(i-1)',(ci^{2},mi),$$

$$@(i+1)',\ldots,@n',D'\rangle$$

avec M≤θ. De même qu'avec l'instruction "select" précédente, on supposera qu'en s' on a di=+infini, car Ti est obligé d'attendre un temps indéfini (sa transition suivante se produisant à une date indépendante de sa volonté). Les cas où ci=c(i,2) ou bien ci=c(i,3) ont été traités dans le paragraphe précédent, de manière indirecte. L'activation de la tâche Ti en ces points de contrôle ne dépend pas

d'elle-même, ce qui justifiait le choix de dj=+infini en ces états.

III.D.xiv Transitions Associées A L'instruction D'appel De Point D'entrée Limitée Dans Le Temps.

Soit l'instruction suivante contenue dans la tâche Ti, où les étiquettes c(i,1) à c(i,4) servent la même cause que ci-dessus:

```
                    .
                    .
                    .
           1
           ci:
                    select
 2   3                                -> ->
ci: ci:                   Tj.a(e ,s ) ;
           4
           ci:
                          S1 ;
                    or
                          delay E ;
           5
           ci:
                          S2 ;
                    end select ;
           6
           ci:
                    .
                    .
                    .
```

1) Premier cas: ci=c(i,1).

```
                                          2
s >-IP-> <t',@1',...,@(i-1)',(ci,mi),@(i+1)',...,@n',D'>
M≤θ.
```

2) Second cas: ci=c(i,2) et (∃ S instruction "select" coté appelée contenant a comme point d'entrée d'une branche ouverte en s telle que cj=at wait(S)).

Dans le cas où Tj est susceptible de recevoir l'appel, les deux tâches transitent simultanément:

```
                                   -> ->
s >-IP-> <t',@1',...,@(j-1)',(cj',mj[u /e ]),@(j+1)',...
                        3
         ...,@(i-1)',(ci,mi),...,@n',D'>
```


où cj' est le successeur du point de contrôle cj, et $\vec{u}$ est le vecteur des paramètres d'entrée formels associés au point d'entrée a de S. On supposera bien entendu qu'il y a compatibilité entre $\vec{u}$ et $\vec{e}$. Dans la présente transition on prendra M≤θ.

3) Troisième cas. ci=c(i,2) et (∃S instruction "select" coté appelée contenant a comme point d'entrée d'une branche ouverte en s / cj=at wait(S)).

Si en l'état s la tâche Tj n'est prête à recevoir l'appel de point d'entrée, Ti attend en c(i,2) que cette condition soit verifiée:

```
s >-IP-> si Ei(s) < E(s)-θ alors
                                     2
           <t',@1',...,@(i-1)',(ci,mi),@(i+1)'...@n',D'>
           sinon
                                     5
           <t',@1',...,@(i-1)',(ci,mi),@(i+1)'...@n',D'>
           fsi
```

avec M≤θ. Les autres transitions auxquelles participe Ti ont été étudiées indirectement dans les paragraphes qui précèdent. Lorsque ci=c(i,3), la tâche attend comme dans le cas d'un appel simple, la fin du rendez-vous et poursuivra son exécution en c(i,4).

III.D.xiv.a Remarques:

a) Nous n'avons pas étudié l'instruction d'acceptation de point d'entrée simple, car on a équivalence entre les deux instructions suivantes:

```
         ->       ->
accept a(u : in ;v : out) do
       S ;
end a;
```

<=>

```
select
  when true => accept a(u : in ;v : out)
                  do
                     S ;
                  end a ;
end select ;
```

b) Les actions atomiques dont nous venons de décrire les transitions associées durent un temps précis que nous avons indiqué pour chacune d'elles. De par le modèle multiprocesseur, une transition s>-IP->s' ne dure pas nécessairement le temps associé aux actions élémentaires activées pendant celle-ci. Nous avons, tout au long de ce paragraphe, donné un majorant pour M=s'.t-s.t, qu'est précisement la durée d'exécution de ces actions atomiques.

III.D.xv Définition:

Soit IP un programme ADA.

On appelle sémantique observationnelle associée à IP, le triplet Obs(IP) = < S0[IP], I[IP], >-IP-> >.

Fin de la définition.

III.D.xvi Proposition:

Soient IP un programme ADA, Obs(IP) sa sémantique observationnelle. Si Card(I[IP])=1, alors le graphe de transition construit par Obs(IP) est acyclique. Il sera noté $\mathcal{G}$(IP).

Fin de la proposition.

Démonstration:

Procédons par l'absurde, et supposons que $\mathcal{G}$(IP) contienne un cycle, i.e.

∃ n ∈ IN, h > 1, ∃ (sj)j=1,...,h une suite d'états de S0[IP] tels que

s1 >-IP-> s2 >-IP-> ... s(n-1) >-IP-> sh = s1.

a) Supposons que ∃ j ∈ {1,...,n} tel que sj.c ≠ s1.c:

celà signifie qu'il existe une tâche de {T1,...,Tn} ayant exécutée une boucle dans son programme au moins. Il existe donc, dans le cycle ci-dessus, une transition d'évaluation de la condition associée à cette boucle. Or, nous venons de voir qu'une telle transition dure un temps θ≠0 (voir paragraphe II.D).

Donc sh.t > s1.t, car la suite (sj.t)j=1,...,h est monotone. Ce qui entraîne que s1≠sh, chose impossible par hypothèse. Nous pouvons ainsi affirmer que: ∀ j ∈{2,...,h}, sj.c=s1.c.

b) L'état mémoire d'une tâche ne pouvant changer que lorsque son compteur ordinal progresse, on obtient donc que

∀ j ∈ {2,...,h}, sj.m = s1.m.

c) Grâce à l'hypothèse d'écoulement monotone du temps, et vu l'hypothèse de départ, on peut également affirmer que:

s1.t = s2.t = ... = sh.t.

d) D'après la définition de Obs(IP), la seule possibilité restante est de transiter d'un état à un autre, en faisant progresser les tâches de s1.D en un temps nul à chaque fois. Celà signifie qu'on n'utilise alors que les éléments de la forme (T,0) de s1.D. En supprimant des éléments de cette forme, on n'inflige aucun retard à aucune tâche. Donc on peut écrire:

sh.D c≠ s(h-1).D c≠ ... s2.D c≠ s1.D,

et ainsi sn.D c≠ s1.D.

Comme Card(s1.D) est fini, on obtient que s1≠sn. Ce cas est également impossible par hypothèse.

Finalement, on obtient s1 = s2 = ... = sh, ce qui est contraire à l'hypothèse de départ.

C.Q.F.D.

III.D.xvii Exemple De Graphe De Transition Associé A La Sémantique Observationnelle D'un Programme.

Considérons le programme IP1=(T1//T2) de la figure 5.

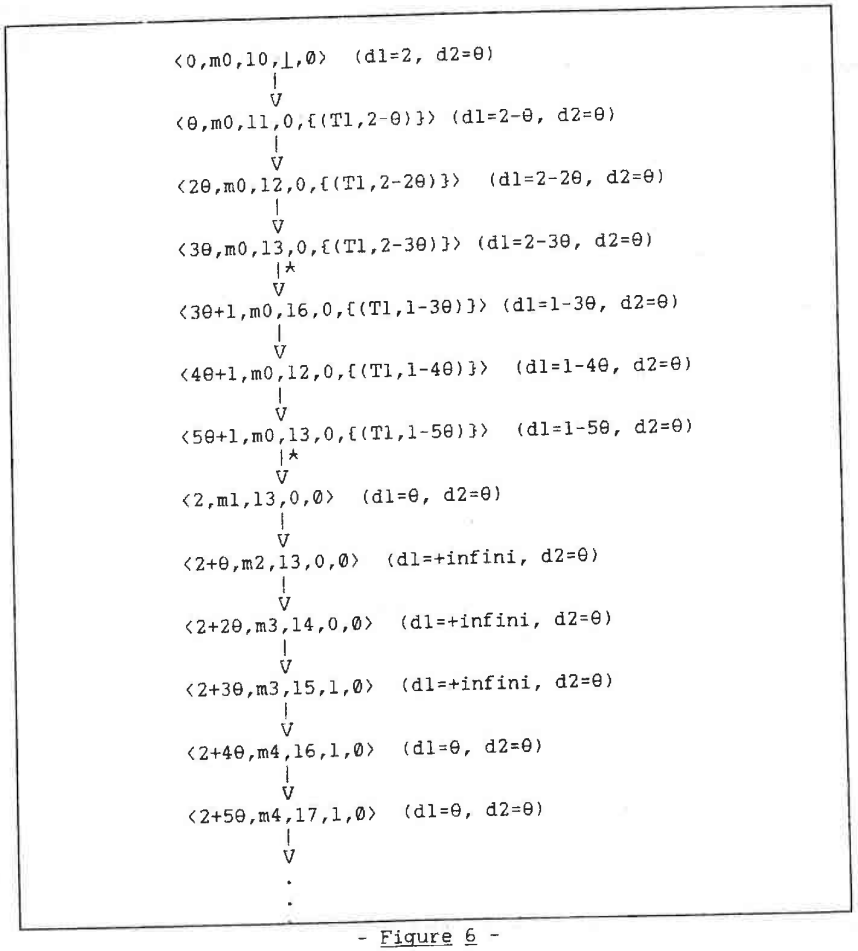

```
          task T1 is

          begin
          <<m0>>
                    delay 2.0 ;
          <<m1>>
<<m2>> <<m3>>          T2.a ;
          <<m4>>
          end T1 ;

          task T2 is
                    b : integer ;
          begin
          <<10>>
                    b:=0 ;
          <<11>>
                    while b = 0 loop
          <<12>>
                              select
                              when true => accept a() do
          <<14>>
                                        b :=1 ;
          <<15>>
                                        end a ;
                              or
                              when true => delay 1.0 ;
                              end select ;
          <<16>>
                    end loop ;
          <<17>>
          end T2 ;
```

- Figure 5 -

alors $\mathcal{U}$(IP1) est le graphe de la figure 6.



- Figure 6 -

Dans l'automate associé, $\mathcal{G}$(IP1), figure 6, nous avons indiqué à coté de chaque état observable s, la valeur du couple (d1(s),d2(s)).

III.E FORMALISATION DES NOTIONS D'EQUITE PAR LES MACHINES DE TURING.

Contrairement aux modèles exécutifs à entrelacement non-déterministes des instructions élémentaires, il subsiste dans notre cas un seul

problème d'équité: celui posé dans la réception des appels de points d'entrée. Pour celà, Pnueli et De-Roever, [51] introduisent des règles de preuve supplémentaires, qui sous hypothèse d'équité (faible ou forte), permettent d'exprimer dans la preuve d'une propriété de vivacité d'un programme qu'une instruction progressera fatalement. Mery [45] résoud ce problème dans le cas des sémaphores, en se restreignant aux traces d'exécution équitables.

III.E.i Définition:

Soit IP un programme ADA. Une trace w $\in$ T[IP] possède la propriété d'équité (encore appelée équité sur les points d'entrée) s'il n'existe pas de tâche Ti $\in$ {T1,...,Tn} attendant indéfiniment sur un appel de point d'entrée e, tandis qu'une infinité d'appels sur e sont acceptés en w.

Fin de la définition.

Rappelons quelques résultats relatifs aux fonctions récursives, extraits de Rogers [53], puis exprimons cette propriété dans le cadre des fonctions A-récursives partielles.

III.E.ii Définition:

La classe des fonctions récursives primitives est la plus petite classe de fonctions $\mathcal{C}$ telle que:

1) les fonctions constantes

$\forall k>0, \forall m\geq 0, \lambda x1...xk[m]$ appartiennent à $\mathcal{C}$,

2) la fonction $\lambda x[x+1]$ appartient à $\mathcal{C}$,

3) les fonctions identité $\lambda x1...xk[xi]$, $1 \leq i \leq k$, $k \geq 1$ appartiennent à $\mathcal{C}$,

4) si F est une fonction k-aire de $\mathcal{C}$ et G1,...,Gk sont des fonctions m-aires ($m \in IN^*$) de $\mathcal{C}$, alors la fonction $\lambda x1...xm[F(G1(x1,...,xm),...,Gk(x1,...,xm))]$ appartient à $\mathcal{C}$,

5) si H est une fonction (k+1)-aire de $\mathcal{C}$, ($k \in IN^*$), et G une fonction (k-1)-aire de $\mathcal{C}$, alors l'unique fonction F satisfaisant aux conditions suivantes appartient à $\mathcal{C}$:

$F(0,x2,...,xk) = G(x2,...,xk)$

$F(y+1,x2,...,xk) = H(y,F(y,x2,...,xk),x2,...xk)$.

Fin de la définition.

III.E.iii Théorème:

Il existe deux fonctions primitives récursives G et H (respectivement à une et à trois variables) telles que l'ensemble $\{ F = \lambda x[G(\mu y[H(z,x,y)=1])], z \in IN\}$ caractérise l'ensemble des fonctions partielles récursives.

Fin du théorème.

Voir [53] pour la démonstration.

III.E.iv Définition:

On définit par Px l'ensemble des instructions associées à l'entier x dans la liste de tous les ensembles d'instructions.

x s'appelle le nombre de Gödel de Px.

Fin de la définition.

III.E.v Notation:

On note par Φx la fonction partielle récursive déterminée par Px. $Wx = Domaine(\Phi x) = Dom(\Phi x)$.

Fin de la notation.

III.E.vi Définition:

Un ensemble A est récursif s'il possède une fonction caractéristique récursive, i.e.

$\exists$ F fonction récursive /$\forall$ x, ($x \in A \Rightarrow F(x) = 1$) et ($x \notin A \Rightarrow F(x)=0$).

Fin de la définition.

III.E.vii Définition:

Un ensemble A est récursivement énumérable si $A=\emptyset$ ou il existe une fonction récursive F telle que $A=Im(F)$.

Fin de la définition.

III.E.viii Machines De Turing A Oracle.

III.E.viii.a Définition:Machine De Turing A Oracle. -

Une machine de Turing à Oracle est un dispositif constitué d'un ruban infini subdivisé en cellules, d'un automate de transition déterministe possédant une tête de lecture-écriture ainsi que d'un ensemble X c IN. Le ruban

possède deux directions de déplacement, et la tête de lecture-écriture pointe toujours sur une cellule de celui-ci. L'automate est défini par un ensemble fini de règles de la forme suivante:

⟨q1,C,Op,q2⟩ ou ⟨q1,C,q2,q3⟩,

où q1, q2, q3 sont des états de l'automate, Op est une opération à effectuer, Op ∈ {1,B,D,G}, C est le contenu de la cellule courante, C ∈ {1,B}.

Un quadruplet de la première forme ci-dessus indique que si l'automate se trouve en l'état initial q1, si la tête de lecture-écriture pointe sur une cellule contenant le caractère C, alors il faut effectuer l'opération Op et se placer dans l'état q2. Une règle de la seconde forme, quant à elle, exprime le fait que si l'on se trouve en l'état q1, si la cellule courante contient le caractère C et si N est le nombre de caractères "1" portés par le ruban, on se placera dans l'état q2 si N ∈ X, et dans l'état q3 sinon.

Fin de la définition.

Une machine de Turing à oracle, dont l'ensemble associé est X détermine par la thèse de Church relativisée, un algorithme relatif à X.

III.E.viii.b Notations:

- De même qu'avec les machines de Turing classiques, on numérote les machines de Turing à oracle. On désignera par Pz' la machine à oracle d'index z dans la numérotation de Gödel.
- On désigne par

Ψ_z^X

la fonction partielle calculée par Pz' relativement à l'ensemble X. Si l'on se donne une fonction partielle Ψ_z^X, relative à X, une donnée x, (inscrite sur la bande de la machine associée) un diagramme décrivant l'ensemble des calculs réalisés par $\Psi_z^X(x)$ lorsque X varie.

Ce diagramme est un arbre de décision dont les branches se subdivisent en deux, à chaque fois que l'algorithme pose une question de la forme "y appartient à X?" à l'oracle.

A chaque branche de celui-ci, il est possible d'associer deux ensembles:

D' = { y ∈ M /une réponse affirmative à la question "y ∈ X ?" est utilisée sur la branche},

D" = { y ∈ M / une réponse négative à la question "y ∈ X ?" est utilisée sur la branche},

Il est évident que l'intersection de D' et de D" est vide.

Dans le processus ci-dessus, on peut générer l'ensemble récursivement énumérable suivant:

{⟨y,u,v⟩ / il existe une branche finie pour laquelle D'=Du et D"=Dv et telle que la valeur en sortie de Pz' est y}.

Grace au s-m-n théorème, (cf. [53]), on obtient la proposition:

III.E.viii.c Proposition:

Si l'on note Wz = Dom(Φz) le domaine de la fonction récursive partielle Φz dans la numérotation de Gödel, on a:

∃ h fonction récursive telle que

$Wh(z)=\{\langle x,y,u,v\rangle$ / il existe une branche finie dans le diagramme associé à Pz' ayant pour valeur d'entrée x, pour valeur de sortie y et telle que $D'=Du$ et $D''=Dv\}$

Fin de la proposition.

Pour la démonstration voir [53].

III.E.viii.d Définition:

$\langle x,y,u,v\rangle$ est consistant si l'intersection de Du et de Dv est vide.

$\langle x1,y1,u1,v1\rangle$ et $\langle x2,y2,u2,v2\rangle$ sont compatibles si l'intersection de Du1 et de Dv2 est vide et celle de Du2 avec Dv1 est vide.

Fin de la définition.

III.E.viii.e Définition:

Wz est régulier si

i) $\forall \langle x,y,u,v\rangle \in Wz$, $\langle x,y,u,v\rangle$ est consistant.

ii) $\forall \langle x1,y1,u1,v1\rangle, \langle x2,y2,u2,v2\rangle \in Wz$,

$(\langle x1,y1,u1,v1\rangle \neq \langle x2,y2,u2,v2\rangle) \Rightarrow \langle x1,y1,u1,v1\rangle$ et $\langle x2,y2,u2,v2\rangle$ ne sont pas compatibles.

Fin de la définition.

III.E.viii.f Théorème:

Il existe une fonction récursive ρ telle que

$\forall z \in IN$,

$W_{\rho(z)}$ est régulier

et

si Wz est régulier, alors $W_{\rho(z)} = Wz$.

Fin du théorème.

Pour la démonstration, voir [53].

III.E.viii.g Définition:

Soit ρ la fonction récursive du théorème précédent.

$\overset{X}{\Phi}z = \{\langle x,y\rangle / \exists u,v \in IN, \langle x,y,u,v\rangle \in W_{\rho(z)}$ et

$Du \subset X$ et $Dv \subset \overline{X}\}$

Fin de la définition.

III.E.viii.h Définition:

Une fonction F est partielle A-récursive si $\exists z \in IN$,

$F = \overset{A}{\Phi}z$.

Une fonction F est A-récursive si $\exists z \in IN$,

$F = \overset{A}{\Phi}z$ et $\overset{A}{\Phi}z$ est totale.

Fin de la définition.

III.E.viii.i Théorème:

Si A est récursif alors

$\overset{A}{\Phi}z$ est une fonction récursive partielle.

Fin du théorème.

III.E.ix Expression De L'équité Faible.

Etudions tout d'abord le problème posé par la simulation de l'exécution d'un programme parallèle par une machine de Turing simple. Ce problème a été longuement étudié dans le passé (cf. [6] par exemple) et des solutions satisfaisant à celui-ci du point de vue théorique ont été implantées par exemple dans les systèmes d'exploitation d'ordinateurs.

L'idée essentielle consiste à découper les différents processus en actions atomiques, dont l'exécution est entrelacée et placée sous le contrôle d'un ordonnanceur de travaux. Dans [6] par exemple, Apt et Co. décrivent de façon précise de tels ordonnanceurs, dont nous nous sommes inspirés dans [52].

Considérons l'approche de Pnueli et De-Roever (cf.[51]), où on simule des programmes parallèles écrits en ADA, à un niveau de parallélisme. Un état d'un programme IP=(T1//.../Tn) est de la forme s=⟨c1,...,cn,m1,...,mn⟩ où ∀ j ∈ {1,...,n} cj désigne la valeur du compteur ordinal de Tj et mj son état mémoire. Dans cette approche on élabore également une relation de transition entre états, classique et proche de notre)-IP-(, ainsi qu'une notion de trace d'exécution.

En un état s donné, le choix d'un processus pour l'exécution de la prochaine action atomique doit vérifier l'hypothèse d'équité faible.

III.E.ix.a Définition:

Une trace d'execution w est faiblement équitable si et seulement si elle est finie ou infinie et dans ce dernier cas il est impossible de trouver un état s de w et une tâche T de IP tels que T est continuellement activable à partir de s en w et activée un nombre fini de fois en w.

Fin de la définition.

Le choix de la prochaine action atomique par l'ordonnanceur peut être laissé à un oracle X. Ainsi un programme ADA peut être transformé en une suite d'actions atomiques dont le scheduler se réduit à la suite d'instructions suivantes:

```
⟨⟨c1,...,cn,m1,...,mn,1⟩⟩ ∈ X ?
si oui alors exécuter l'action numéro c1 de T1
sinon
⟨⟨c1,...,cn,m1,...,mn,2⟩⟩ ∈ X ?
```



```
si oui alors exécuter l'action numéro c2 de T2
sinon
        .
        .
        .
⟨⟨c1,...,cn,m1,...,mn,n⟩⟩ ∈ X ?
si oui alors exécuter l'action numéro cn de Tn
sinon
        {l'exécution de IP est terminée}
fsi
.
.
.
fsi
```

où ⟨⟨ ⟩⟩ est la fonction associant à un (2n+1)-uplet ⟨c1,...,cn,m1,...,mn,i⟩ son code entier. Dans [53] on définit une fonction de codage de la manière suivante:

Soit la fonction

$\pi : IN \times IN \longrightarrow IN$

$(x,y) \longrightarrow (x*x + 2xy + y*y + 3x + y)$

récursive et bijective;

soit la suite infinie de fonctions $(\pi_i)_{i \in IN^*}$ définie récursivement par:

$$\pi^1 = \lambda x[x]$$

$$\forall k > 0,\quad \pi^{k+1} = \lambda x1 \ldots x(k+1)[\pi^k(\pi^k(x1,\ldots,xk),x(k+1))]$$

on a ainsi

$$\langle\langle c1,\ldots,cn,m1,\ldots,mn,i\rangle\rangle = \pi^{2n+1}(c1,\ldots,cn,m1,\ldots,mn,i)$$

pour s=⟨c1,...,cn,m1,...,mn⟩ un état de IP et i∈{1,...,n}.

On peut donc affirmer que IP correspond (par la thèse de Church relativisée) à une machine de Turing à oracle, ou bien que sa fonction calculée Φ est partielle X-récursive:

$$\exists z \in IN,\ \Phi = \Phi^X_z$$

III.E.ix.b Remarque:

Nous venons de supposer abusivement, ci-dessus, que l'état mémoire des différentes tâches en s est un entier. Il est évident que si le nombre de variables de celles-ci est constant dans IP, une telle simplification est tout à fait légale, vu l'existence d'une fonction de codage. Pour être tout à fait correct, il aurait fallu écrire

$$\langle\langle mi(vi), \ldots, mi(vi_{Card(\mathcal{D}\, i)}) \rangle\rangle$$

au lieu de mi ci-dessus, celà pour tout i de l'ensemble {1,...,n}, avec

$$\{vi, \ldots, vi_{Card(\mathcal{D}\, i)}\} = Dom(mi).$$

III.E.x Expression De L'équité Sur Les Points D'entrée.

Replaçons nous dans le cadre de programmes ADA temps-réel, celui choisi pour notre étude.

Nos programmes étant de longueur finie, nous pouvons supposer qu'il existe un nombre fini de points d'entrée pour un programme IP donné. Mais le non-déterminisme associé au moyen de communication employé ici, fait correspondre à ceux-ci des machines de Turing à oracle non nécessairement récursives. Soit IP un programme ADA, dont les variables sont du type entier. Simulons donc l'exécution de IP par une machine de Turing à oracle et à scheduler équitable, réalisant un entrelacement non-déterministe de ses actions atomiques.

Voyons quelle pourrait être la forme d'un ordonnanceur de tâches, contraint de respecter l'hypothèse d'équité sur les points d'entrée, définie plus haut.

1) Associons à chaque point d'entrée du programme IP, une file d'attente contenant les appels de ce point d'entrée rangés dans un ordre quelconque: il s'agit d'une liste non-bornée d'objets de la forme

$$\langle T, \vec{e}, \vec{f} \rangle \text{ où}$$

- $T \in \{T1, \ldots, Tn\}$
- $\vec{e}$ est le vecteur des paramètres d'entrée d'un appel sur le point d'entrée
- $\vec{f}$ est le vecteur de variables désignant les paramètres de sortie de ce même appel.

Lors de l'appel de cette entrée par une instruction de la forme

```
Ti.a( e, f )    ou de la forme
select
    Ti.a( e, f );
    S1;
or
    S2;
end select;
```

on enfile le triplet $\langle Ti, \vec{e}, \vec{f} \rangle$ dans la file associée à l'entrée a.

A chacune de ces entrées a, on associe bien entendu un compteur noté a'COUNT, et indiquant le nombre d'éléments de la file d'attente. Ces données appartiennent à l'ordonnanceur et lui permettent de décider lors d'un rendez-vous éventuel quels sont les partenaires à mettre en correspondance.

2) Soit s ∈ S0[IP], s=⟨t,c,m,D⟩, supposons qu'il existe une tâche Ti telle que ci=wait(S), S étant une instruction "select" côté appelé. Supposons de plus que le scheduler ait décidé d'exécuter le rendez-vous correspondant à une branche ouverte de point d'entrée a, dans S. Il pourra questionner un oracle Xa afin de trouver le correspondant de Ti, par les instructions que voici:

```
⟨⟨cl,...,cn,m1,...,mn,1⟩⟩ ∈ Xa?
si oui alors
extraire le triplet numéro 1 de la file
associée à a
sinon
⟨⟨cl,...,cn,m1,...,mn,2⟩⟩ ∈ Xa?
        .
        .
        .
⟨⟨cl,...,cn,m1,...,mn,a 'COUNT⟩⟩ ∈ Xa?
si oui alors
extraire le triplet numéro a 'COUNT de la file
associée à a
fsi
.
.
.
fsi
                -⟩ -⟩
{Soit ⟨Tj,e ,f ⟩ le triplet extrait }

Prendre Tj comme tâche appelante pour le
rendez-vous sur a.
```

Supposons finalement que a1,...,al soient les points d'entrée utilisés dans IP, tous distincts entre eux. Soient X1,...,Xn les oracles associés à ces points d'entrée.

Soit Φ la fonction calculée par IP. On peut affirmer que Φ est une fonction partielle (X1 join X2 join ... join X1)-récursives, ou en d'autres termes que

$$\exists z \in IN, \Phi^{X1 \text{ join } X2 \ldots \text{ join } Xn} = \Phi z$$

avec la définiton suivante de la fonction join:

Soient A et B deux sous-ensembles de W,

A join B = {y ∈ IN / ∃ x ∈ A, y = 2x} U

{y ∈ IN / ∃ x ∈ B, y = 2x+1}

et avec

X1 join X2 join ... join X1 = (...(X1 join X2) join X3... join X1).

LE SYSTEME AXIOMATIQUE DE PREUVE A.R.T.

CHAPITRE IV

LE SYSTEME AXIOMATIQUE DE PREUVE A.R.T.

A partir de la logique temporelle arborescente, définie par MANNA et PNUELI (cf. [39] à [43]), nous construisons un système axiomatique pour notre sous-ensemble de ADA. Ce système nous permettra de prouver des propriétés temps-réel sous hypothèse d'équité sur les points d'entrée. A.R.T. est composé de quatre parties:

1) d'un ensemble d'axiomes spécifiant le comportement des instructions séquentielles,

2) d'un ensemble d'axiomes évoquant le fonctionnement des rendez-vous,

3) de règles de composition séquentielles et parallèles,

4) de règles secondaires, certaines particulières à notre modèle exécutif, d'autres nécessaires à la preuve de certaines prémisses des règles précédentes.

Décrivons au préalable le langage d'assertions choisi, ainsi que son interprétation.

IV.A LANGAGES D'ASSERTIONS ASSOCIES A UN PROGRAMME ADA - DEFINITION.

Les assertions de contrôle classiques sont nécessaires à la spécification de l'état du contrôle, à savoir les assertions "at S", "in S" et "after S" où S est une instruction séquentielle ADA.
Soit S une instruction séquentielle ADA.

1) On dit que "at S" est vrai en s si et seulement si l'étiquette précédant la première action atomique de S est une composante du vecteur s.c.

2) On dit que "after S" est vrai en s si et seulement si le label suivant immédiatement l'instruction S est une composante de s.c.

3) On dit également que "in S" est vrai en s si et seulement si une des étiquettes contenues (au sens du chapitre III) dans S fait partie du vecteur s.c. Dans ce cas, le concept d'étiquette désigne aussi bien les étiquettes fictives que les labels classiques, leur ordre relatif dans IP étant défini par la relation de succession "succ". On notera respectivement ces assertions par

$$s \models \underline{\text{at}}\ S, \quad s \models \underline{\text{after}}\ S, \quad s \models \underline{\text{in}}\ S$$

lorqu'elles sont vérifiées en s.
Nous emploierons également les abréviations suivantes:

$$\underline{\text{at}}\ IP = \bigwedge_{i=1}^{n} \underline{\text{at}}\ Ti, \qquad \underline{\text{after}}\ IP = \bigwedge_{i=1}^{n} \underline{\text{after}}\ Ti$$

4) Nous définissons un opérateur supplémentaire, nécessaire à l'expression de la date absolue dans un programme IP en l'état s: on dit que "is u" est vrai en s, si et seulement si la première composante de s est égale au réel u. Formellement on écrira

$$\forall u \in \mathbb{R}+, (s \models \underline{\text{is}}\ u \text{ si et seulement si } s.t=u).$$

IV.A.i Définition:

Un langage d'assertion A pour le programme ADA IP est un ensemble d'assertions vérifiant les propriétés suivantes:

1) Pour toute instruction séquentielle S contenue dans l'une des tâches de IP, les assertions de contrôle "at S", "after S" et "in S" sont dans A.

2) Tout prédicat dépendant de l'état mémoire de IP est dans A.

3) Toute conjonction finie d'assertions de A est dans A.

4) Toute disjonction d'assertions de A est dans A.

5) Pour tout réel positif u, l'assertion de datation "is u" est dans A.

Fin de la définition.

IV.A.ii Notation:

Pour un état s donné d'un programme IP, le fait qu'une assertion P du langage d'assertion A associé à IP est vérifiée en s sera notée par "s|=P" ou encore par "P(s)".

Fin de la notation.

Définissons maintenant une interprétation des assertions d'un langage d'assertions A associé à IP.

IV.A.iii Définition:

Soient IP un programme ADA, A un langage d'assertions de IP. L'interprétation de P ∈ A, dépendant de la forme des

assertions de A est définie de la manière suivante:

Soit s un état de IP.

1) Si P est une assertion de contrôle, s|= P a été défini plus haut.

2) Si P est une assertion précisant un état mémoire de IP, s|= P si et seulement si la valeur des variables de IP en l'état s vérifie P.

3) Si P=(/\Pi),i∈I, avec Pi∈A et Card(I)<+infini, s|= P si et seulement si (∀ i ∈ I, s|= Pi).

4) Si P=(\/Pi),i∈I, avec (∀ i ∈ I, Pi ∈ A), s|= P si et seulement si (∃ i ∈ I, s|= Pi).

5) Si P=is u, pour un certain u ∈ IR+, |= P a été défini précédement.

Fin de la définition.

Maintenant nous pouvons définir, dans une interprétation donnée, les opérateurs temporels qui nous seront nécessaires.

Soient IP un programme ADA, ⟨S[IP], I[IP], t[IP], T[IP]⟩ une sémantique abstraite de IP telle que Card(I[IP])=1. Soit Te[IP] le plus grand sous-ensemble de T[IP] tel que toute trace de Te[IP] vérifie la propriété d'équité sur les points d'entrée de IP. Soient s ∈ S[IP] et P, Q, R des assertions de A, un langage d'assertions de IP.

1) L'implication:

s|= P => Q ssi (s|= P => s|= Q)

2) L'opérateur "next":

s|= o P ssi ∀ s' ∈ S[IP], t[IP](s,s') => s'|= P

3) L'opérateur "always":

s|= [] P ssi ∀ s' ∈ S[IP], t[IP]*(s,s') => s'|= P

4) L'opérateur de fatalité:

s|= <> P ssi ∀ w ∈ Te[IP](s), ∃ i ∈ Dom(w), w(i) |= P

5) L'opérateur "Until":

s|= P Until Q ssi ∀ w ∈ Te[IP](s), ∃ i ∈ Dom(w),
(∀ j ∈ [0,i[, w(j)|= P) ^ (w(i)|= Q).

Cet opérateur généralise l'opérateur "until" classique, et indique qu'à partir de s, P reste continuellement vraie jusqu'à ce que Q soit vérifiée, ce qui se produira fatalement.

6) L'opérateur de fatalité dans le passé:

s|= ♦ P ssi ∃ s0 ∈ I[IP], ∃ s' ∈ S[IP], t[IP]*(s0,s') ^ t[IP]*(s',s) ^ (s'|= P)

Celui-ci indique que si l'on inverse le sens d'écoulement du temps, P est fatalement vérifiée, ce qui équivaut à dire qu'il existe un état antérieur à s où P était vérifiée.

7) L'opérateur "until-temporisé":

∀ u ∈ IR+, s|= P--u-->Q,R ssi ∃ t0 ∈ IR+, (s|= P ^ is t0) =>
(s|= Q Until (R ^ is t0+u))

De cette manière, nous exprimons le fait que si P est vérifiée en s, Q sera satisfaite pendant u secondes à partir de s, puis au bout de u secondes l'assertion R sera verifiée.

8) L'opérateur de fatalité exacte:

$\forall u \in IR+$, $s \models \langle u \rangle P$ ssi $s \models true \text{ --}u\text{--}\rangle true, P$

Ceci équivaut à dire que P sera verifiée après u secondes à partir de s.

9) L'opérateur de fatalité moindre:

$\forall u \in IR+$, $s \models \langle\langle u \rangle P$ ssi $\exists t' \in [0,u[$, $s \models \langle t' \rangle P$

10) L'opérateur de fatalité absolue:

$\forall u \in IR+$, $s \models \langle\langle u \rangle\rangle P$ ssi $s \models \langle\rangle (P \wedge \underline{is}\ u)$

Contrairement à l'opérateur de fatalité exacte, nous indiquons que P sera fatalement vérifiée à la date absolue u.

11) L'opérateur de possibilité:

$s \models \langle\rangle\_ P$ ssi $\exists w \in Te[IP](s)$, $\exists i \in Dom(w)$, $w(i) \models P$

Cet opérateur nous permet d'exprimer le fait que P sera satisfaite sur au moins une trace d'exécution partant de s.

IV.A.iv Notation:

$s \models \langle\langle u \rangle P$ ssi $(s \models \langle\langle u \rangle P)$ v $(s \models \langle u \rangle P)$ pour u réel positif.

Fin de la notation.

IV.A.v Proposition:

Soient IP un programme ADA, IP=(T1//.../Tn), A un langage d'assertion associé à IP, P, Q des éléments de A, $\langle S[IP], I[IP], t[IP], Te[IP] \rangle$ sa sémantique abstraite dans un modèle équitable, $y \in IR+$.

Alors on a les équivalences suivantes:

1) $s \models P \Rightarrow [] Q \Leftrightarrow \forall x \in IR+, s \models P \text{ --}x\text{--}\rangle Q, Q$

2) $s \models P \Rightarrow \langle\rangle Q \Leftrightarrow \exists x \in IR+, s \models P \text{ --}x\text{--}\rangle true, Q$

3) $s \models P \Rightarrow \langle y \rangle Q \Leftrightarrow s \models P \text{ --}y\text{--}\rangle true, Q$

4) $s \models P \Rightarrow \langle\langle y \rangle Q \Leftrightarrow \exists x \in [0,y], s \models P \text{ --}x\text{--}\rangle true, Q$

5) $s \models P \Rightarrow \langle\langle y \rangle\rangle Q \Leftrightarrow \exists x \in IR+, s \models P \text{ --}x\text{--}\rangle true, Q \wedge \underline{is}\ y$

6) $s \models P \Rightarrow \langle\rangle\_ Q \Leftrightarrow s \models P \Rightarrow (\sim([] \sim Q))$

Fin de la proposition.

Démonstration:

La preuve de ces équivalences est facile à l'aide des définitions ci-dessus. Donnons la première à titre d'exemple:

Condition nécessaire: Supposons que

$P(s) \Rightarrow (\forall s' \in S[IP], t[IP]^*(s,s') \Rightarrow Q(s'))$

et montrons que

$\forall x \in IR+, \exists t' \in IR+, [P(s) \wedge s.t=t'] \Rightarrow$

$[\forall w \in Te[IP](s), \exists i \in IR+, (\forall j \in [0,i], Q(w(j)))$

$\wedge (w(i).t=t'+x)]$

Soit $x \geq 0$, prenons $t'=s.t$, et supposons que P(s) est vrai. Par hypothèse nous avons:

$\forall s' \in S[IP], t[IP]^*(s,s') \Rightarrow Q(s')$ (1)

Soit $w \in Te[IP]$, prenons $i=x$ et montrons que la conclusion est alors satisfaite:

a) Soit j ∈ [0,i]: par la proposition III.C.v du chapitre précédent,
t[IP]*(w(0),w(j)) est vrai, donc comme s=w(0), on a Q(w(j)) grace à (1).

b) Par définition d'une trace d'exécution,
∃ K ∈ IR+, ∀ z ∈ IR+, w(z).t=z+K, donc,
w(i).t = K+i = K+x = w(0).t+x = s.t+x = t'+x.

Condition suffisante: Supposons P(s) est vrai, soit s' ∈ S[IP], tel que t[IP]*(s,s'). Utilisons l'hypothèse en posant x = s'.t-s.t. Alors,
∃ t' ≥ 0, [P(s) ^ s.t=t'] =>
[∀ w ∈ Te[IP](s), ∃ i ≥ 0,
(∀ j ∈ [0,i], Q(w(j)) ^ w(i).t=t'+x)].
Soit w une trace partant de s et passant par s': celle-ci existe de part le modèle et car t[IP]*(s,s'). Alors on peut dire que ∃ i ≥ 0, ∀ j ∈ [0,i], Q(w(j)).
D'où Q(w(i)) = Q(w(x)) = Q(s') est vrai.
C.Q.F.D.

IV.B LE SYSTEME DE PREUVE A.R.T.

Le comportement d'un programme ADA donné, IP sera explicité en découpant l'intervalle temporel [0,+infini[en sous-intervalles fermés dans chacun desquels il est possible de décrire le comportement de IP de manière précise, grace aux axiomes du système A.R.T.
Pour celà nous considérons l'opérateur "until-temporisé" comme opérateur de base pour A.R.T.

IV.B.i Définition:

Soit IP=(T1//...//Tn) un programme ADA, P une assertion d'un langage d'assertion de IP, t0, t ∈ IR+, p ∈ IN*, p<n. On dit que les tâches T1,...,Tp n'interfèrent pas sur l'assertion P pendant l'intervalle de temps [t0,t0+t[si l'assertion suivante est vraie:
is t0 ^ P --t--> P, true.
Fin de la définition.

Enonçons le sytème axiomatique de preuve.
Soient IP un programme ADA, A un langage d'assertions pour IP.

IV.B.ii Première Partie: Axiomes Associés Aux Instructions Séquentielles.

Soient Ti une tâche de IP, P une assertion de A ne portant que sur les variables locales de Ti.
Supposons également qu'il y a absence d'interférence des autres tâches, différentes de Ti sur P pendant le durée indiquée dans chaque axiome. Nous le supposerons de même pour l'assertion P[v/e] dans le premier axiome.

1) A-AFF:
P[v/e] ^ at v:=e --θ+f(e)--> at v:=e ^ P[v/e], after v:=e ^ P

2) A-NULL:
P ^ at null --θ--> P ^ at null, P ^ after null

3) A-DELAY:

P ^ at delay E --E--> P ^ at delay E, P ^ after delay E

4) Soit S l'instruction séquentielle

S = if B then S1 ; end if ;

A-IFF1:

P ^ at S ^ B --θ--> P ^ at S ^ B, P ^ at S1 ^ B

A-IFF2:

P ^ at S ^ ~B --θ--> P ^ at S ^ ~B, P ^ after S ^ ~B

5) Soit S l'instruction suivante

S = while B loop S1; end loop;

A-WHILE1:

P ^ at S ^ B --θ--> P ^ at S ^ B, P ^ at S1 ^ B

A-WHILE2:

P ^ at S ^ ~B --θ--> P ^ at S ^ ~B, P ^ after S ^ ~B

A-WHILE3:

P ^ after S1 ^ B --θ--> P ^ after S1 ^ B, P ^ at S1 ^ B

A-WHILE4:

P ^ after S1 ^ ~B --θ--> P ^ after S1 ^ ~B, P ^ after S ^ ~B

L'axiome A-DELAY exprime le fait qu'une tâche dont le point de contrôle est situé devant une instruction "delay x" y attendra pendant x secondes au bout desquelles, elle se déplacera vers la fin de cette instruction.

IV.B.iii Deuxième Partie: Axiomes Associés Aux Instructions De Communication.

Soient Ti une tâche de IP, P une assertion de A ne contenant aucune référence explicite au temps et ne portant que sur les variables de

l'ensemble $\mathcal{V}_i$.

1) Soit S l'instruction suivante, contenue dans Ti:

S = Tj.a(e ,f);

où Tj est une tâche de IP et a un point d'entrée de Tj, et soit S' une instruction "select côté appelée" de Tj, susceptible d'entrer en rendez-vous avec Ti.

i) A-AP1:

P ^ at S --θ--> P ^ at S, P ^ at wait(S)

ii) A-AP2:

P ^ Tj at wait(S') ^ Ti at wait(S) --θ-->P ^ Ti at wait(S) ^ Tj at wait(S'), P ^ Ti at RV(a)

iii) A-AP3:

P ^ Ti at wait(S) ^ ~Tj at wait(S') --θ--> P ^ Ti at wait(S), P ^ Ti at wait(S)

iv) A-AP4:

P ^ Ti at RV(a) ^ Tj at (end a) --θ--> P ^ Ti at RV(a) ^ Tj at (end a), P[..] ^ Ti after (end a)

v) A-AP5:

P ^ Ti at RV(a) ^ ~Tj at (end a) --θ--> P ^ Ti at RV(a), P ^ Ti at RV(a)

Le premier de ces axiomes décrit la mise en attente de rendez-vous de Ti, les deux axiomes suivants expriment le comportement de Ti selon que Tj est susceptible ou non d'entrer immédiatement en rendez-vous. De même, les axiomes A-AP4 et A-AP5 montrent que Ti attend de façon active la fin du rendez-vous lorque celui-ci a lieu,

et transite vers le début de l'instruction suivante une fois que l'instruction "accept" correspondante est achevée.

P[..] est l'assertion obtenue par substitution des paramètres de retour aux paramètres formels f, dans P. Les points de contrôle wait(S), wait(S') et RV(a) correspondent aux étiquettes fictives associées aux instructions de communication, étudiées au chapitre précédent.

2) Soit l'instruction suivante

```
S = select
                -> ->
        Tj.a(e ,f );
        S1;
    or
        delay E ;
        S2;

    end select; S3;
```

où Tj ∈ {T1,...Tn}\{Ti}, a est un point d'entrée de Tj, E est une expression arithmétique à résultat dans IR+.

i) A-SA1:

P ^ Ti at S --θ--> P ^ Ti at S, P ^ Ti at wait(S)

ii) A-SA2

P ^ Ti at wait(S) --E--> [P ^ [(Ti at wait(S) ^ ~(Tj in(accept a...end a))] v [(Ti at RV(a)) ^ (Tj in (accept a...end a))]] v [P[..] ^ (Ti in (S1;S3)[..] v Ti at S1) ^ Tj after(accept a...end a)],

[P ^ ((Ti at RV(a) ^ Tj in (accept a...end a)) v Ti at S2)] v [P[..] ^ (Ti in (S1;S3)[..] v Ti at S1[..])]

iii) A-SA3:

P ^ Ti at RV(a) ^ Tj at(end a) --θ--> P ^ Ti at RV(a) ^ Tj at(end a), P[..] ^ Ti at S1[..]

iv) A-SA4:

P ^ Ti at RV(a) ^ ~Tj at(end a) --θ--> P ^ Ti at RV(a), P ^ Ti at RV(a).

Après un axiome classique, A-SA1, nous expliquons le comportement de Ti pendant une durée E à partir de sa date de mise en attente de rendez-vous: pendant l'intervalle de temps [0,E[relativement à cette date, la tâche Ti peut:

i) attendre dans le même état un éventuel rendez-vous,

ii) être en état d'attente de fin de rendez-vous, dans quel cas la tâche appelée exécute l'instruction "accept" correspondante,

iii) ou bien elle a terminé un rendez-vous; dans ce dernier cas elle exécute les instructions S1 et S3 en séquence.

De même, E secondes après la mise en attente de rendez-vous, Ti peut:

i) être en rendez-vous avec Tj,

ii) avoir achevé un tel rendez-vous et exécuté (S1;S3),

iii) ou bien peut ne pas avoir réalisé de rendez-vous du tout avec Tj; elle poursuit alors avec (S2,S3).

Lorsque celà nous a été possible, nous avons indiqué conjointement à chacun de ces cas où se situe la tâche Tj, dans le but de guider les

preuves: une fois que nous composerons A-SA2 avec une assertion relative à Tj, cette dernière permettra de simplifier le résultat en supprimant les cas inutiles distingués pour Ti dans A-SA2. Les axiomes A-SA3 et A-SA4 sont identiques à A-AP4 et A-AP5 dans leur sémantique et indiquent comment se comporte Ti pendant le rendez-vous.

3) Considérons l'instruction "select" suivante, possédant p branches de rendez-vous et q branches temporisées. Les points d'entrée seront tous supposés différents et l'on ne mentionnera pas le type des paramètres d'entrée et de sortie.

```
S=select
    .
    .
    .
    or
                                  ->       ->
    when bk => accept ak(uk: in; vk: out) do

                    1
                   Sk;

                end ak;

                    2
                   Sk;
    or
    .
    .
    .
    when wj => delay tj

                    3
                   Sj;
    or
    .
    .
    .
  end select;
```

Soient les assertions mémoire

$$C = \bigvee_{j=1}^{q} wj \qquad B = \bigvee_{j=1}^{p} bj$$

et soit t= min{tj/ 1≤j≤q, wj} si q > 0, +infini si q=0.

Soit Ti la tâche contenant l'instruction S.

i) A-SEL1:

P ^ at S --θ--> P ^ at S, P ^ at wait(S)

Introduisons pour la suite de ce paragraphe une notion de distance.

IV.B.iii.a Définition:

On considère une fonction distance.

$$d: \{T1,\ldots,Tn\} \times \bigcup_{j=1}^{n} LABEL(Tj) \text{ -----> } \overline{IR}+$$

telle que pour Tj ∈ {T1,...,Tn} et l ∈ LABEL(Tj), d(Tj, l) est la distance de la tâche Tj à son point de contrôle l, c'est à dire le temps nécessaire à Tj pour atteindre l depuis l'instant courant. (∀ j ∈ {1,...,n}, LABEL(Tj) désigne l'ensemble des étiquettes de Tj, obtenues selon la méthode de numérotation du chapitre précédent).

Fin de la définition.

IV.B.iii.b Définition

Soit la fonction dT associant, à une tâche Tj ∈ {T1,...,Tn} et à S=Ti.a, une instruction d'appel du point d'entrée a de Tj un réel positif indiquant le temps nécessaire à Tj pour parvenir à l'instruction de la même forme syntaxique que S, la plus proche de la date courante: soit $\mathcal{S}$ l'ensemble des instructions de la même forme syntaxique que S, contenue dans Tj:

$$dT(Tj,S)=\begin{cases} \min\{ d(Tj,\underline{wait}(I))/\ I \in S \wedge d(Tj,\underline{wait}(I)) \geq 0\}, & \text{si } card(S) < +infini \\ +infini \text{ sinon} \end{cases}$$

Fin de la définition.

ii) A-SEL2:

S'il existe une tâche $Tj \in \{T1,...,Tn\}\backslash\{Ti\}$ et un entier $i \in \{1,...,p\}$ tels que

$$D = d(Tj,Ti.ai'(ej,fj)) = \min\{dT(T',Ti.ak)/\ bk \wedge dT(T',Ti.ak) < +infini \wedge T' \neq Ti \wedge 1 \leq k \leq p\}$$

et $D < t$, alors

$Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge C \wedge P \xrightarrow{D+\theta} Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge C \wedge P,$

$Ti \underline{at}\ {}^{1}Si'[.] \wedge B \wedge C \wedge Tj \underline{at}\ RV(ai') \wedge P[.].$

où [.] désigne la substitution des paramètres effectifs dans les paramètres formels de la tâche appelée correspondante,
i.e. $[.] = [ej, ui']$.

iii) A-SEL3:

S'il existe $Tj \in \{T1,...,Tn\}\backslash\{Ti\}$ et $i' \in \{1,...p\}$ tels que:

$$D = d(Tj,Ti.ai'(ej;fj)) = \min\{dT(T',Ti.ak)/\ bk \wedge dT(T',Ti.ak) < +infini \wedge T' \neq Ti \wedge 1 \leq k \leq p\}$$

et $D \geq t$, alors

$Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge C \wedge P \xrightarrow{t} Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge C \wedge P,$

$Ti \underline{at}\ {}^{3}Sj' \wedge B \wedge C \wedge P.$

où j' est défini par $tj'=t$.

iv) A-SEL4:

S'il existe $Tj \in \{T1,...,Tn\}\backslash\{Ti\}$, et un entier $i' \in \{1,...,p\}$ tels que:

$$D = d(Tj,Ti.ai'(ej,fj)) = \min\{dT(T',Ti.ak)/\ bk \wedge dT(T',Ti.ak) < +infini \wedge T' \neq Ti \wedge 1 \leq k \leq p\}$$

soit fini,

alors

$Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge \neg C \wedge P \xrightarrow{D+\theta} Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge \neg C \wedge P,$

$Ti \underline{at}\ {}^{1}Si'[.] \wedge B \wedge \neg C \wedge Tj \underline{at}\ RV(ai') \wedge P[.].$

sinon,

v) A-SEL5:

$Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge \neg C \wedge P \xrightarrow{+infini} Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge \neg C \wedge P,\ Ti \underline{at}\ \underline{wait}(S) \wedge B \wedge \neg C \wedge P$

vi) A-SEL6:

$Ti \underline{at}\ \underline{wait}(S) \wedge \neg B \wedge C \wedge P \xrightarrow{t} Ti \underline{at}\ \underline{wait}(S) \wedge \neg B \wedge C \wedge P,$

$Ti \underline{at}\ {}^{3}Sj' \wedge \neg B \wedge C \wedge P$

où j' est défini par $tj'=t$

vii) A-SEL7:

$Ti \underline{at}\ \underline{wait}(S) \wedge \neg B \wedge \neg C \wedge P \xrightarrow{\theta} Ti \underline{at}\ \underline{wait}(S) \wedge \neg B \wedge \neg C \wedge P,$

$Ti \underline{after}(S) \wedge \neg B \wedge \neg C \wedge P.$

viii) A-SEL8:

Si $Tj \in \{T1,...,Tn\}\backslash\{Ti\}$ et $i' \in \{1,...,p\}$, alors

$Ti \underline{at}\ (\underline{end}\ ai') \wedge Tj \underline{at}\ RV(ai') \wedge P \xrightarrow{\theta} Ti \underline{at}\ (\underline{end}\ ai') \wedge Tj \underline{at}\ {}^{2}RV(ai') \wedge P,$

$Ti\ Si' \wedge T'\ \underline{after}\ (Ti.ai') \wedge P$

ix) A-SEL9:

Si l désigne l'étiquette précédant l'instruction "end ai'" contenue dans S, i' ∈ {1,...,p}, et si Tj ∈ {T1,...,Tn}\{Ti}, alors

$$\forall x \in [0,d(Ti,l)],$$
$$Ti \text{ in } {}^{l}Si' \wedge Tj \text{ at } RV(ai') \text{ --x--> } Ti \text{ in } {}^{l}Si' \wedge Tj \text{ at } RV(ai'),$$
$$Ti \text{ in } {}^{l}Si' \wedge Tj \text{ at } RV(ai')$$

Les axiomes A-SEL1 à A-SEL9 décrivent comment la tâche Ti exécute une instruction "select" côté appelée. Nous distinguons tous les cas possibles, décrits informellement au chapitre II, selon le nombre et le type des branches ouvertes dans S.

Nous avons raccourci les preuves de propriétés temps-réel dans le système A.R.T. en prévoyant à l'avance dans les axiomes A-SEL2 et A-SEL4 la date du premier rendez-vous, via la notion de distance d'une tâche à une instruction. Au lieu de celà, nous aurions pu introduire des axiomes permettant l'attente active de Ti (en découpant l'intervalle de temps d'attente en sous-intervalles de longueur θ), conformément à la sémantique abstraite de IP, évitant ainsi une prévision de son exécution.

Il aurait alors suffi de rajouter un axiome, permettant les rendez-vous uniquement lorsque les deux partenaires sont dans un état "wait", et de durée θ. Notre choix est plus intéressant car il permet de regrouper des intervalles temporels inintéressants en un même intervalle.

Nous noterons par "|- P" le fait qu'une assertion P ∈ A a été déduite dans le système A.R.T.

IV.B.iv Troisième Partie: Les Règles De Composition.

Soient P, P1, P2, Q, Q1, Q2, Q', R, R1, R2, R' des assertions de A, et t0, t1, £, £0, £1 des réels positifs quelconques.

1) R-SEQ:

A.R.T. |- P --£0--> Q, Q' ^ is t0

A.R.T. |- Q' ^ is t0 --£1--> R, R' ^ is t1

A.R.T. |- P --£0+£1--> Q v R, R' ^ is t1

2) R-COMP:

A.R.T. |- P1 --£--> Q1, R1

A.R.T. |- P2 --£--> Q2, R2

A.R.T. |- P1 ^ P2 --£--> Q1 ^ Q2, R1 ^ R2

Ces deux règles sont essentielles à notre système de preuve car elles permettent de composer séquentiellement des assertions du type "until-temporisées" et de réaliser des compositions "parallèles" (analogues à celles réalisées dans les systèmes à la HOARE comme en [24]) de celles-ci afin de décrire des propriétés globales du programme.

Soient (Pk),k∈{0,...,K} ∈ A et (£k)k∈{0,...,K-1} ∈ IR+,

pour un certain K ∈ IN*.

3) R-INDUCTION:

$\forall k \in \{1,\ldots,K\}$, A.R.T. $\vdash P_k \xrightarrow{\pounds(k-1)} P_k, P_{(k-1)}$

$$\text{A.R.T.} \vdash P_K \xrightarrow{\sum_{i=0}^{K-1} \pounds_i} \bigvee_{i=0}^{K} P_i, P_0$$

Cette règle d'induction constitue une généralisation de R-SEQ, souvent bien utile dans les preuves.

IV.B.v Quatrième Partie: Axiomes Et Règles Auxiliaires.

Soient P, Q, R, R1, R2, U1, U2 des assertions de A, £, £', t0, u des réels positifs.

1) A-START:

is 0 => at IP

2) A-END:

$\forall i \in \{1,\ldots,n\}$, $\forall \pounds \in \mathbb{R}+$,

after Ti --£--> after Ti, after Ti

3) R-HYP:

Si P => Q est vraie pour le programme IP, i.e. $\forall s \in S[IP]$, P(s) => Q(s), alors

A.R.T. |- P => Q

4) R-IMPL:

A.R.T. |- P => Q

A.R.T. |- Q --£--> R1, R2

A.R.T. |- R1 => U1

A.R.T. |- R2 => U2

A.R.T. |- P --£--> U1, U2

5) R-SIMPL:

A.R.T. |- P --£--> Q, R

$\forall \pounds' \in [0,\pounds[$, A.R.T. |- P --£'--> Q, Q

6) R-IS:

A.R.T. |- P --£--> Q, R

A.R.T. |- P ^ is t0 --£--> Q ^ ($\exists$ u $\in [t0,t0+\pounds[$, is u), is (t0+£) ^ R

Quelques explications s'imposent pour ces règles d'inférence et ces axiomes auxiliaires.

L'axiome A-START définit l'origine des temps à l'exécution de IP, à laquelle on se réfèrera dans les assertions de datation "is".

Notre hypothèse des traces d'exécution infinies se reflète dans l'axiome A-END, où l'on formalise une fois de plus le fait qu'une tâche terminée le reste indéfiniment.

Nul n'ignore que toute extension axiomatisée de l'arithmétique est incomplète (voir par exemple [53]): ce résultat est connu sous le nom de "théorème d'incomplétude de Gödel". Nous remédions à cet obstacle par l'utilisation d'un oracle: supposons que nous

puissions décider qu'une implication est vraie ou fausse, alors dans le cas affirmatif nous déduirons qu'elle est vérifiée dans A.R.T.

La règle R-IMPL est des plus classiques et permet de supprimer des données inutiles dans une proposition. De même, la règle de simplification R-SIMPL permet de déduire d'une proposition portant sur un intervalle fermé borné réel, des informations portant sur un sous-intervalle compact ayant même borne gauche.

Il vient naturellement à l'esprit une règle capable d'introduire une assertion de datation dans une proposition "until-temorisée", celà afin d'établir les prémisses de certaines règles d'inférence telles R-SEQ: cet objectif est atteint avec R-IS.

IV.B.vi Définition:

Soient IP un programme ADA, Q une proposition "until-temporisée". Une preuve de la proposition Q dans le système axiomatique A.R.T. est une suite finie d'assertions AS1,...,ASq telle que

∀ i ∈ {1,...,q}, ASi est

- une implication valide pour IP i.e. un élément de l'ensemble {P => Q / |= P => Q},
- ou un axiome de A.R.T.,
- ou bien la conséquence d'une règle d'inférence de A.R.T. dont les prémisses sont dans l'ensemble {AS1,...,AS(i-1)}.

Fin de la définition.

Maintenant que le système de preuve a été énoncé, nous devons vérifier qu'il est correct i.e. que toute proposition prouvée dans celui-ci est

vraie:

∀ AS proposition "until-temporisée", A.R.T.|- AS => |= AS.

IV.C CORRECTION DU SYSTEME AXIOMATIQUE A.R.T.

IV.C.i Théorème:

Soient IP un programme ADA, A un langage d'assertion pour IP, P, Q, R ∈ A, t'≥0.

Si A.R.T.|- P --t'--> Q, R alors |= P --t'--> Q, R.

Fin du théorème.

Démonstration:

Vérifions pour celà la correction des règles d'inférence:

1) La règle R-SEQ: Soient les hypothèses suivantes

H1 = ∀ s ∈ S[IP], ∃ t1' ∈ IR+, [P(s) ^ s.t = t1']
=> [∀ w ∈ Te[IP](s), ∃ il ∈ Dom(w),
(∀ j ∈ [0,il[, Q(w(j)) ^
(Q'(w(il))) ^ w(il).t = t0 = t1'+£0)]

H2 = ∀ s ∈ S[IP], ∃ t2' ≥ 0,
[Q'(s) ^ s.t = t0 = t2'] =>
[∀ w ∈ Te(s), ∃ i2 ∈ Dom(w),
(∀ j ∈ [0,i2[, R(w(j)))
^ (R'(w(i2)) ^ w(i2).t = t1 = t2'+£1)].

Soit s ∈ S[IP], grace à H1, ∃ t1' ≥ 0, [...] => [...]. Prenons t' = t1' et supposons que l'on ait P(s) ^ s.t = t'.

Soit w ∈ Te[IP](s), d'après H1 on a encore ∃ il ∈ Dom(w), (...) ^ (...). Considérons l'état s' = w(il) et appliquons H2 à celui-ci:

$\exists$ t2' $\geq$ 0, [Q'(s') $\wedge$ s'.t = t0 = t2'] $\Rightarrow$ (1)

($\forall$ w' $\in$ Te(s'), $\exists$ i2 $\in$ Dom(w'), ($\forall$ j $\in$ [0,i2[,

R(w'(j))) $\wedge$ (R'(w'(i2)) $\wedge$ w'(i2).t = t1 = t2'+ε1)],

Dans ces conditions, l'assertion Q'(s') $\wedge$ s'.t = t0 = t2' est vérifiée, pour un certain t2' $\in$ IR+. Prenons dans (1) pour w' la trace w partant de w(i1). On obtient donc:

$\exists$ i2 $\in$ Dom(w'), ($\forall$ j $\in$ [0,i2[, R(w'(j)) $\wedge$

(R'(w'(i2)) $\wedge$ w'(i2).t = t1 = t2'+ε1) (2)

Posons i = i1+i2 et montrons que la conclusion est vérifiée:

$\star$ Soit j $\in$ [0,i[: Si j $\in$ [0,i1[, par l'assertion H1, Q(w(j)) est vérifiée, donc (Q v R)(w(j)) également.

Si j $\in$ [i1,i[, par H2, R(w'(j-i1)) = R(w(j)) est vraie, donc on a (Q v R)(w(j)).

$\star$ Grace à (2) on a également

R'(w'(i2)) $\wedge$ (w'(i2).t = t1 = t2'+ε1) = R'(w(i1+i2)) $\wedge$ (w(i1+i2).t = t1 = t2'+ε1 = t0+ε1 = t1'+ε0+ε1 = t'+ε0+ε1)

soit la conclusion désirée, à savoir:

$\forall$ s $\in$ S[IP], $\exists$ t' $\geq$ 0, [P(s) $\wedge$ s.t = t'] $\Rightarrow$

[$\forall$ w $\in$ Te[IP](s), $\exists$ i $\in$ Dom(w),

($\forall$ j $\in$ [0,i[, (Q v R)(w(j))) $\wedge$

(R'w(i)) $\wedge$ w(i).t = t1 = t'+ε0+ε1)].

C.Q.F.D.

2) La règle R-COMP:

Supposons les hypothèses suivantes:

H1 = $\forall$ s $\in$ S[IP], $\exists$ t1' $\in$ IR+, [P1(s) $\wedge$ s.t = t1'] $\Rightarrow$

[$\forall$ w $\in$ Te[IP](s), $\exists$ i1 $\in$ Dom(w),

($\forall$ j $\in$ [0,i1[, Q1(w(j))) $\wedge$

(R1(w(i1)) $\wedge$ w(i1).t = t1'+ε)]

H2 = $\forall$ s $\in$ S[IP], $\exists$ t2' $\in$ IR+, [P2(s) $\wedge$ s.t = t2'] $\Rightarrow$

[$\forall$ w $\in$ Te[IP](s), $\exists$ i2 $\in$ Dom(w), ($\forall$ j $\in$ [0,i2[, Q2(w(j))) $\wedge$ (R2(w(i2,) $\wedge$ w(i2).t = t2'+ε)]

Soit s $\in$ S[IP], appliquons H1 et H2 à cet état, et considérons t'=t1'.

Supposons que (P1 $\wedge$ P2)(s) $\wedge$ (s.t = t') est vérifiée.

Alors grace à H1 on a

$\forall$ w $\in$ Te[IP](s), $\exists$ i1 $\in$ Dom(w), (...) $\wedge$ (...).

De même, grace à H2 on a

$\forall$ w $\in$ Te[IP](s), $\exists$ i2 $\in$ Dom(w), (...) $\wedge$ (...), et on a également, en comparant le premier terme des implications contenues dans H1 et H2 que t1' = t2' = s.t.

Considérons w une trace équitable partant de s. On obtient, grace aux hypothèses:

$\exists$ i1 $\in$ Dom(w), ($\forall$ j $\in$ [0,i1[, Q1(w(j)))

$\wedge$ (R1(w(i1)) $\wedge$ w(i1).t = t1'+ε), (3)

$\exists$ i2 $\in$ Dom(w), ($\forall$ j $\in$ [0,i2[, Q2(w(j))) $\wedge$

(R2(w(i2)) $\wedge$ w(i2).t = t2'+ε) (4)

Comme t1' = t2', on a w(i1).t = w(i2).t. Or pour une trace w on a par définition:

$\exists$ K $\in$ IR+, $\forall$ x $\in$ IR+, w(x).t = x+K.

Donc (w(i1).t = i1+K = i2+K) $\Rightarrow$ (i1 = i2).

Montrons qu'en posant i = i1 = i2 on a la conclusion:

$\star$ Soit j $\in$ [0,i[: par (3) et (4), il est évident que

(Q1 ^ Q2)(w(j))

* Il est non moins trivial que (R1 ^ R2)(w(i)) ^ (w(i).t = t'+£) est vérifiée.

D'où la conclusion: ∀ s ∈ S[IP], ∃ t' ∈ IR+, [(P1 ^ P2)(s) ^ s.t = t'] =>

[∀ w ∈ Te[IP](s), ∃ i ∈ Dom(w),

(∀ j ∈ [0,i[, (Q1 ^ Q2)(w(j))) ^

((R1 ^ R2)(w(i)) ^ w(i).t = t'+£)]

C.Q.F.D.

3) La règle R-INDUCTION:

Supposons vraies par hypothèse les K propositions suivantes notées H1,...,HK:

∀ k ∈ {1,...,K},

Hk = ∀ s ∈ S[IP], ∃ tk' ∈ IR+, [Pk(s) ^ s.t = tk']

=>

[∀ w ∈ Te[IP](s), ∃ ik ∈ Dom(w),

(∀ j ∈ [0,ik[, Pk(w(j))) ^

(P(k-1)(w(ik)) ^ w(ik).t = tk'+£(k-1))]

et cherchons à démontrer la proposition que voici:

CONCL = ∀ s ∈ S[IP], ∃ t' ∈ IR+,

[PK(s) ^ s.t = t'] =>

[∀ wK ∈ T[IP](s), ∃ i ∈ Dom(w(K)),

(∀ j ∈ [0,i[,

$$\bigvee_{\eta=0}^{K} P_\eta(wK(j)))$$

$$\wedge\ (P\eta(wK(i)) \wedge wK(i).t = \sum_{\eta=0}^{K-1} \varepsilon_\eta + t')]$$

Nous allons démontrer cette dernière en construisant une suite de K réels i1,...,iK et une suite de K états associés au découpage d'une trace wK en K morceaux, par les hypothèses H1,...,HK. Le réel i recherché dans la conclusion est la somme de ces réels.

Soit sK ∈ S[IP], grace à Hk, ∃ tk' ∈ IR+, [...] => [...]. Prenons t' = tK' dans CONCL, et supposons que [PK(s) ^ s.t = t'] soit vraie.

Soit wK ∈ Te[IP](sK), alors par HK,

∃ iK ∈ IR+, (...) ^ (...).

Posons s(K-1) = wK(iK) et appliquons l'hypothèse H(K-1) à celui-ci avec t(K-1) = wK(iK).t. On a alors:

[P(K-1)(s(K-1)) ^ s(K-1).t = t(K-1)'] =>

[∀ w ∈ Te[IP](s(K-1)), ∃ i(K-1) ∈ Dom(w),

(P(K-2)(w(i(K-1))) ^ (w(i(K-1)).t = t(K-1)'+£(K-2)))

^ (∀ j ∈ [0,i(K-1)[, P(K-1)(w(j)))]. (5)

Comme P(K-1)(s(K-1)) ^ (s(K-1).t = t(K-1)') = P(K-1)(wK(iK)) ^ wK(iK).t = t(K-1)' est vraie par HK, nous pouvons appliquer le membre droit de l'implication (5) à la sous-trace w(K-1) de wK commençant en l'état s(K-1).

Alors, par H(K-1):

∃ i(K-1) ∈ IR+, (...) ^ (...)

.

.

Par H1:

$\exists$ il $\in$ IR+, ($\forall$ j $\in$ [0,il[, P1(wl(j))) $\wedge$

(P0(wl(il)) $\wedge$ (wl(il).t = tl'+£0) (6)

En appliquant ainsi les hypothèses HK,...,H1 successivement nous obtenons une suite de sous-traces de wK, imbriquées, (wk)k$\in${1,...,K} et une suite de K réels (ik)k$\in${1,...,K} marquant les états initiaux de ces sous-traces.

Posons i = iK+...+il et vérifions qu'on a bien la conclusion désirée:

* Soit j $\in$ [0,i[: dans ce cas,

$\exists!$ k $\in$ {1,...,K}, j $\in$ $\left[\overline{\underset{\eta=k+1}{\overset{K}{\backslash}{/}}} i_\eta , \overline{\underset{\eta=k}{\overset{K}{\backslash}{/}}} i_\eta \right[$ =

[tk',t(k-1)'[, avec t0' = tl'+£0.

Grace à Hk, nous savons que:

$\forall$ j' $\in$ [0,ik[, Pk(wk(j')) ce qui équivaut à dire que:

$\forall$ j' $\in$ [0,ik[, Pk(wk(tk'+j')) par définition de wk.

Donc,

pour j $\in$ [tk', t(k-1)'[, $\overset{K}{\underset{\eta=0}{\backslash/}} P_\eta(wK(j))$.

* L'assertion (6) nous informe que:

P0(wl(il)) $\wedge$ (wl(il).t = tl'+£0)

= P0(wK(il)) $\wedge$ (wK(i).t = tl'+£0) par définition de wl.

Or, les hypothèses nous ont permis de construire une suite de dates tl',...,tK' marquant le début des traces wl,...,wK respectivement et vérifiant:

tl' = sl.t = w2(i2).t = t2'+£1

t2' = s2.t = w3(i3).t = t3'+£2

.

.

.

t(K-1)' = s(K-1).t = wK(iK).t = tK'+£(K-1) donc nous pouvons encore écrire que:

P0(wK(il)) $\wedge$ (wK(i).t = $\overset{K-1}{\underset{\eta=0}{\overline{\backslash/}}} £_\eta$+tK').

C.Q.F.D.

4) La règle R-IMPL:

Supposons que |= P => Q,

|= Q --£--> R1, R2,

|= R1 => U1,

|= R2 => U2.

La seconde hypothèse s'écrit encore:

$\forall$ s $\in$ S[IP], $\exists$ t" $\in$ IR+, [Q(s) $\wedge$ s.t = t"] =>

[$\forall$ w $\in$ Te[IP](s), $\exists$ i' $\in$ Dom(w),

($\forall$ j $\in$ [0,i'[, R1(w(j)) $\wedge$

(R2(w(i'))) $\wedge$ w(i').t = t"+£)]. (7)

Soit s une état de IP, appliquons la seconde hypothèse à s, et soit t'=t". Supposons qu'on a P(s) $\wedge$ (s.t=t'): par la première hypothèse, on a donc Q(s) $\wedge$ (s.t=t'), donc selon (7),

$\forall$ w $\in$ Te[IP](s), $\exists$ i' $\in$ Dom(w),

($\forall$ j $\in$ [0,i'[, R1(w(j))) $\wedge$ (R2(w(i')) $\wedge$

w(i').t=t"+£). (8)

Soit w $\in$ Te[IP](s), et soit i' le réel associé à w par

la proposition ci-dessus. Posons i=i' et montrons que la conclusion est vérifiée:

* Soit j ∈ [0,i[: par (8) on sait que R1(w(j)) est vrai. Or la troisième hypothèse s'écrit:

∀ s ∈ S[IP], R1(s) => U1(s), donc on a encore U1(w(j)).

* La proposition (8) nous permet d'écrire que

R2(w(i')) ^ (w(i').t = t"+£)

= R2(w(i)) ^ (w(i).t = t'+£)

or la quatrième hypothèse affirme que ∀ s ∈ S[IP], R2(s) => U2(s). Donc U2(w(i)) ^ (w(i).t = t'+£).

D'où la conclusion.

C.Q.F.D.

5) La règle R-SIMPL:

Supposons que |= P --£-->Q, R, et soit £' ∈ [0,£[.

Soit s ∈ S[IP], par hypothèse, on a donc

∃ t" ∈ IR+, [P(s) ^ s.t = t"] =>

[∀ w ∈ Te[IP](s), ∃ i' ∈ Dom(w),

(∀ j ∈ [0,i'[, Q(w(j))) ^

(R(w(i')) ^ w(i').t = t"+£)] (9)

Posons t'=t", et supposons que P(s) ^ (s.t = t').

Soit w ∈ Te[IP], par la proposition (9), on obtient donc

∃ i' ∈ Dom(w), (∀ j ∈ [0,i'[, Q(w(j))) ^

(R(w(i')) ^ w(i').t = t"+£) (10)

Considérons que i=£', et vérifions que la conclusion est satisfaite:

* Soit j ∈ [0,i[: Comme l'état initial de w est s, w(0).t=t'; or nous savons que

∃ K ∈ IR+, ∀ j ∈ IR+, (w(j).t = j+K). Donc t'=K.

De plus selon (10), w(i').t = t'+£ = i'+K = i'+t', donc i'=K, ce qui entraîne encore que pour j<i, nous avons j<i = £' < £ = i'. Donc, grace à l'assertion (10), Q(w(j)) est vérifiée.

* Il est non moins évident que, puisque i<i', on a également Q(w(i)). Comme nous avons remarqué que t'=K, on peut évidemment affirmer que:

w(i).t = i+K = i+t' = £'+t'.

D'où la conclusion.

C.Q.F.D.

6) La règle R-IS:

Supposons que |= P --£-->Q, R, c'est à dire que

∀ s ∈ S[IP], ∃ t" ∈ IR+, [P(s) ^ s.t = t"] =>

[∀ w ∈ Te[IP](s), ∃ i' ∈ Dom(w),

(∀ j ∈ [0,i'[, Q(w(j))) ^

(R(w(i')) ^ w(i').t = t"+£)]. (11)

Soit s ∈ S[IP], prenons pour t' le réel t" associé à s par l'hypothèse ci-dessus. Supposons que

(P ^ is t0)(s) = (P(s) ^ (s.t = t0)).

Soit w ∈ Te[IP](s), alors grace à (11) on a :

∃ i' ∈ Dom(w), (∀ j ∈ [0,i'[, Q(w(j))) ^ (R(w(i'))

^ w(i').t = t"+£) (12)

prenons i=i' et montrons que la conclusion est vérifiée:

* Soit j ∈ [0,i[: la proposition (12) affirme que Q(w(j)). De plus, comme w ∈ Te[IP](s), on sait que: ∃ K ∈ IR+, ∀ x ∈ IR+, w(x).t = x+K. Donc w(j).t = j+K = w(0)+j = t'+j = t0+j.

Soit au total,

$Q(w(j)) \wedge (w(j).t = t0+j) = Q(w(j)) \wedge (\exists\ u \in [t0,t0+£[,\ w(j).t = u)$.

* De même, on a: $w(i).t = i+K = £+t' = £+t0$, et par (12) on obtient: $R(w(i)) \wedge (w(i).t = t0+£ = t'+£)$.

D'où le résultat désiré.

C.Q.F.D.

Ceci achève la démonstration de la correction du système A.R.T.

Fin de la démonstration.

IV.D CONCLUSION.

Nous avons ébauché un système axiomatique de preuve afin de démontrer des propriétés temps-réel de programmes ADA, fondé sur l'opérateur "until-temporisé" et exprimant à notre avis un maximum de propriétés sur un intervalle de temps compact donné. Quoique correct, notre système de preuve ne semble pas être sémantiquement complet. Ce problème pourrait être résolu, quant à lui de la manière suivante:

Pour démontrer, pour un programme IP donné, la proposition $(P \text{ --}t'\text{-->} Q, R)$, où $P, Q, R \in A$ un langage d'assertions de IP et $t' \in IR+$, il faut:

1) Trouver la suite strictement croissante d'instants critiques $t0,t1,\ldots,tq$ (pour un certain $q \in IN$) telle que $tq = t'+t0$ et $(\forall\ j \in \{0,\ldots,q\},\ tj \in [t0,t0+t'[)$ auxquels IP est observable. (Il s'agit en fait de la suite des dates associées aux états observables de IP).

2) A partir de cette suite, il faut construire une suite d'assertions $P0,P1,\ldots,Pq$ telle que

$$+\ \forall\ i \in \{0,\ldots,q-1\},\ \models\ Pi \text{ --}(t(i+1)-ti)\text{-->}\ Pi,\ P(i+1) \qquad (13)$$

$$+\ \models P \Rightarrow P0$$

$$+\ \models Pq \Rightarrow R$$

$$+\ \models \left(\bigvee_{i=0}^{q-1} Pi\right) \Rightarrow Q$$

à partir de laquelle nous avons alors par R-INDUCTION,

$$\text{A.R.T.}\ \vdash P0 \text{ --}(tq-t0)\text{-->} \bigvee_{j=0}^{q-1} Pj,\ Pq$$

et ensuite par R-IMPL,

$$\text{A.R.T.}\ \vdash P \text{ --}t'\text{-->}\ Q,\ R.$$

Les propriétés "until-temporisées" (13) ne peuvent être construites, dans tous les cas, dans notre système A.R.T. En effet, les assertions $(Pi), i \in \{0,\ldots,q\}$ peuvent contenir une composante relative à l'état mémoire de IP, mettant en jeu les variables locales des différentes tâches de celui-ci. Or nous ne pouvons établir (par R-COMP) comme assertion mémoire que les conjonctions d'assertions mémoire relatives aux différentes tâches. Il nous faudrait donc élargir notre système de preuve en autorisant l'emploi de variables auxiliaires et en introduisant une notion d'invariant global (tel que le fait APT dans [5]) ainsi qu'une règle de coopération assurant une bonne utilisation des variables auxiliaires de IP.

PROPRIETES DE FATALITE TEMPS-REEL
- EXEMPLES DE PREUVES

CHAPITRE V

PROPRIETES DE FATALITE TEMPS-REEL - EXEMPLES DE PREUVES.

Après avoir défini des opérateurs temporels susceptibles d'exprimer des notions temps-réel, nous nous proposons dans un premier temps, d'exposer les types de propriétés exprimables dans le système A.R.T. Puis nous évoquerons informellement une méthode à suivre afin de prouver les plus intéressantes d'entre elles, que nous illustrerons par deux exemples. Nous avons choisi pour celà deux problèmes concrets de simulation de phénomènes physiques, caractéristiques de ceux rencontrés en pratique.

V.A PROPRIETES TEMPORELLES EXPRESSIBLES DANS LE SYSTEME A.R.T.

N'ayant jusqu'à présent spécifié aucune propriété relative aux traces d'exécution d'un programme ADA donné, nous allons énoncer maintenant les propriétés les plus intéressantes exprimables dans notre langage d'assertions.

Il est possible de dégager, ainsi que l'ont fait MANNA et PNUELI, une hiérarchie "naturelle" des propriétés exprimables en logique temporelle:

i) Le premier niveau hiérarchique regroupe les propriétés d'invariance i.e. les propriétés de la forme P => []Q, et dont font partie les propriétés de correction partielle, d'absence d'erreur à l'exécution et d'absence d'interblocage. Cette classe a été longuement étudiée dans des papiers tels que [17] et [43].

ii) La seconde classe contient les propriétés de fatalité, i.e. de la forme P => <>Q et contient entre autres des propriétés telles que la terminaison, la correction totale, l'accessibilité en section critique, l'accessibilité et l'absence de famine. Cette classe est enrichie dans notre cas d'une propriété de "délai de réponse à un signal", de la forme P => <y>Q.

iii) Si la plupart des chercheurs sont d'accord quant à la définition des deux premières catégories, il n'en est pas de même pour la troisième. Cette dernière regroupe les propriétés de précédence, i.e. celles exprimables à l'aide d'un opérateur "until":

s|= P Until Q, s|= P --t-->Q, R , et s|= P => <<u> Q.

Le dernier type de propriétés, bien qu'ayant le forme d'une propriété de fatalité, doit à notre avis être classé dans ce niveau hiérarchique car il exprime la précédence de Q par rapport à la propriété "is s.t+u". Les opérateurs "until" sont fréquemment utilisés afin d'exprimer des propriétés d'équité, via des règles d'inférence telles qu'on les rencontre dans [51] et [43]. Dans notre modèle temps-réel nous avons mis en évidence un ensemble de propriétés de type "jusqu'à", nous apparaissant comme primordiales: la réception "instantanée" d'un signal, la sensibilité aux variations d'un signal continu et l'émission "continue" d'un signal.

Rappelons tout d'abord les propriétés les plus importantes appartenant aux deux premiers niveaux hiérarchiques (propriétés 1 à 5 ci-dessous), puis formalisons les propriétés temps-réel de la dernière classe, fondamentales pour la suite de notre étude, ainsi que la propriété d'équité sur les points d'entrée.

Supposons que le modèle sous-jacent soit défini par la sémantique abstraite.

Soient IP=(T1//...//Tn), (n ∈ IN*) un programme ADA,

A un langage d'assertions associé à IP,

P, Q ∈ A, y ∈ IR+,

⟨S[IP], I[IP], t[IP], Te[IP]⟩ la sémantique abstraite associée à IP, dans le modèle équitable.

1) Propriété: La correction partielle.

On dit que IP est partiellement correct relativement à P et Q si et seulement si

$$(P \wedge \underline{at}\ IP) \Rightarrow \Box(\underline{after}\ IP \Rightarrow Q).$$

2) Propriété: L'invariance globale.

P est globalement invariante pour IP si et seulement si

$$Init \Rightarrow \Box P$$

où l'on définira

s|= Init <=> ∀ w ∈ T[IP](s), w(0) ∈ I[IP].

3) Propriété: La terminaison.

On dit que le programme IP termine si et seulement si

$$\underline{at}\ IP \Rightarrow \langle\rangle\ \underline{after}\ IP$$

4) Propriété: L'absence d'interblocages.

On dit que IP est exempt d'interblocages si et seulement si

$$\neg(\ \exists\ (l1,\ldots,ln) \in \overset{n}{\underset{j=1}{X}} Label(Tj),$$

$$(\forall\, j \in \{1,\ldots,n\},\ lj \neq \underline{\text{after}}\ Tj) \wedge$$
$$(\underline{\text{at}}\ IP \Rightarrow \langle\rangle\_(/\backslash_{j=1}^{n}\ Tj\ \underline{\text{at}}\ lj)) \wedge$$
$$((/\backslash_{j=1}^{n}\ Tj\ \underline{\text{at}}\ lj) \Rightarrow [](/\backslash_{i=1}^{n}\ Ti\ \underline{\text{at}}\ li))).$$

5) Propriété: L'accessibilité potentielle.

L'assertion Q est accessible depuis P si et seulement si

$$P \Rightarrow \langle\rangle\_\ Q$$

6) Propriété: L'équité sur les points d'entrée.

Soit Te[IP] c T[IP] l'ensemble des traces équitables de IP.

Sous l'hypothèse d'équité, nous pouvons exprimer la propriété:

$\forall$ Tk $\in$ {T1,...,Tn}, $\forall$ a point d'entrée de Tk,

$\forall$ i $\in$ {1,...,n}, i $\neq$ k, $\exists$ N1,N2 $\in$ IN,

$\exists\ \vec{e} = (e1,\ldots,eN1) \in (\text{EXPRESSIONS})^{N1}$ vecteur d'expressions arithmétiques à variables dans $\vec{\mathcal{V}}i$, $\exists\ \vec{f} \in (\vec{\mathcal{V}}i)^{N2}$,

$Ti\ \underline{\text{at}}\ Tk.a(\vec{e}\ ;\vec{f}\ ) \Rightarrow \langle\rangle(Ti\ \underline{\text{after}}\ Tk.a(\vec{e}\ ;\vec{f}\ ))$.

7) Propriété: La réception "instantanée" d'un signal.

$\forall$ Tk $\in$ {T1,...,Tn}, $\forall$ a point d'entrée de Tk, on dit que Tk est susceptible de recevoir "instantanément" le signal a avec une précision y si et seulement si

$\forall$ Ti $\in$ {T1,...,Tn}\{Tk}, $\exists$ N1,N2 $\in$ IN,

$\exists\ \vec{e} \in (\text{EXPRESSIONS})^{N1}, \exists\ \vec{f} \in (\vec{\mathcal{V}}i)^{N2}$,

$Ti\ \underline{\text{at}}\ Tk.a(\vec{e}\ ;\vec{f}\ ) \Rightarrow \langle\langle y\rangle(Tk\ \underline{\text{after}}\ \underline{\text{accept}}\ a)$.

(Nous omettrons les paramètres formels de l'instruction d'acceptation).

V.A.i Définition:

Soient IP un programme ADA, Tk une tâche de IP, a un point d'entrée appartenant à Tk.

On dit que a est un signal continu de précision x $\in$ IR+ si et seulement si deux appels de l'entrée a sont espacés d'un délai inférieur à x, i.e.

$\exists$ N1,N2 $\in$ IN,

$$[](\exists\ Ti \in \{T1,\ldots,Tn\}, \exists\ \vec{e} \in (\text{EXPRESSIONS})^{N1}, \exists\ \vec{f} \in (\vec{\mathcal{V}}i)^{N2},$$
$$Ti\ \underline{\text{at}}\ Tk.a(\vec{e}\ ;\vec{f}\ )) \Rightarrow$$
$$\langle\langle x\rangle(\exists\ Tj \in \{T1,\ldots,Tn\}, \exists\ \vec{e}" \in (\text{EXPRESSIONS})^{N1},$$
$$\exists\ \vec{f}" \in (\vec{\mathcal{V}}j)^{N2},\ Tj\ \underline{\text{at}}\ Tk.a(\vec{e}",\vec{f}"))).$$

On dit que a est un signal continu s'il existe un réel positif x tel que a est un signal continu de précision x.

Fin de la définition.

8) Propriété: Sensibilité aux variations d'un signal "continu".

Soient Tk une tâche de IP, a un point d'entrée de Tk, nous dirons que Tk est sensible au signal continu a, de façon continue et avec une précision y si et seulement si

$\exists$ N1,N2 $\in$ IN,

$$[](\exists\ u \in (\vec{\mathcal{V}}k)^{N1}, \exists\ v \in (\vec{\mathcal{V}}k)^{N2}, Tk\ \underline{\text{at}}\ \underline{\text{accept}}\ a(\vec{u}\ ;\vec{v}\ ))$$

$\Rightarrow$

$$\langle\langle y\rangle(\exists\ u' \in (\vec{\mathcal{V}}k)^{N1}, \exists\ v' \in (\vec{\mathcal{V}}k)^{N2}, Tk\ \underline{\text{at}}\ \underline{\text{accept}}\ a(\vec{u}',\vec{v}'))).$$

9) Propriété: Emission "continue" d'un signal a.

Soient Tk une tâche de IP, a un point d'entrée d'une tâche Tj de IP.

On dit que Tk émet "continuellement" le signal a avec une précision de y si et seulement si $\exists$ N1,N2 $\in$ IN

$$[](\exists\ \vec{e} \in (\text{EXPRESSIONS})^{N1}, \exists\ \vec{f} \in (\vec{\mathcal{V}}k)^{N2}, Tk\ \underline{\text{at}}\ Tj.a(\vec{e}\ ;\vec{f}\ ))$$

=>

$$\langle y\rangle(\exists\ \vec{e}'' \in (\text{EXPRESSIONS})^{N1},\ \exists\ f'' \in (\mathcal{V}k)^{N2},\ Tk \text{ at } Tj.a(e'';f''))$$

i.e. deux appels de l'entrée a sont séparés d'un délai inférieur à y.

10) Propriété: Délai de réponse à un signal.

Soient Tk une tâche de IP, A un langage d'assertions de IP; si P est une assertion ne portant que sur les variables locales de Tk, indiquant l'arrivée d'un signal de Tk, et si Q est une assertion de A ne portant que sur les variables locales de Tk, indiquant qu'une réponse est donnée, alors Tk admet un délai de réponse à P de y, dont la réponse est Q, si et seulement si

$$[](P \Rightarrow \langle y\rangle Q).$$

Remarquons que Q peut être une assertion marquant l'émission d'un signal en guise de réponse à P.

Exemple:

Pour indiquer que la tâche T1 nécessite un délai de réponse de 10 secondes afin de répondre à un appel de son entrée a par l'appel du point d'entrée b de Tj, on écrira:

$$[](P \Rightarrow \langle 10\rangle Q)$$

avec P = ($\exists$ T' $\in$ {T1,...,Tn}, T' at T1.a)

Q = (T1 at Tj.b)

(Nous n'indiquerons pas les paramètres dans les instructions d'appel de point d'entrée, pour des raisons de lisibilité. De même, nous avons supposé tout au long de ce paragraphe qu'il y a compatibilité entre les paramètres effectifs et formels).

Fin de l'exemple.

V.B SYNTHESE D'UN PROGRAMME SATISFAISANT A UNE PROPRIETE DE FATALITE TEMPS-REEL.

Intéressons nous à la propriété 8 précédente et voyons comment associer à la spécification d'un type de problème temps-réel particulier un programme ADA. Toute la puissance d'expression du langage ADA nous sera nécessaire, c'est pourquoi nous nous permettrons ici d'exprimer les algorithmes dans la version complète de celui-ci (au sens de [19]).

Définissons les ensembles infinis suivants:

PROG = l'ensemble des programmes ADA, corrects au sens de [19],

SEQ = l'ensemble des programmes ADA, séquentiels corrects au sens de [19],

V(IP) = l'ensemble des propriétés temporelles validées par le programme IP,

$$\backslash V = \bigcup_{IP \in PROG} V(IP)$$

$$\backslash Vs = \bigcup_{IP \in SEQ} V(IP)$$

Supposons l'existence d'une fonction

g : $\mathcal{P}(\backslash Vs)$ ----> PROG, associant à un sous-ensemble de \Vs un programme ADA, vérifiant les propriétés de celui-ci et tel que Im(g) c SEQ.

Les éléments de \Vs sont évidemment des propriétés de type 1 à 3. Nous désirons trouver une extension de g à (V8 U \Vs), que nous noterons h, où V8 désigne le sous-ensemble de \V contenant les propriétés de "sensibilité aux variations d'un signal continu", i.e. de type 8:

Donnons nous un sous-ensemble fini, non-vide X de (V8 U \Vs) ne contenant qu'un seul élément de V8, noté X0:

$$X = \{X1,...,Xz,X0\}, \text{ pour un certain } z \in \mathbb{N}^*.$$

Il est alors possible d'associer à {X1,...,Xz} c \Vs un programme séquentiel, à savoir g({X1,...,Xz}). Supposons que X0 soit la propriété de "sensibilité au signal continu a, avec une précision f, dont les informations véhiculées par a sont du type $\vec{t}$.

* Une première solution triviale consiste à écrire h(X) de la manière suivante:

```
                task T1 is
                                -)      -)
                    entry a(v : in t);
                end T1;

                task body T1 is
                                                -)
                    tab : array(0..max) of t;
                    i : integer := 0;
                   -- variables supplémentaires de g({X1,...,Xz})
                   -- non encore mentionnées.
                begin
                <<10>>
                    loop
                <<11>>
                                        -)      -)
<<12>>                      accept a (v : in t ) do
                <<13>>                          -)
                                tab(i) := v;
                <<14>>
                            end a;
                <<15>>
                            i := i+1;
                <<16>>
                            g({X1,...,Xz});
                <<17>>
                    end loop;

                end T1;
```

- Figure 7 -

où nous avons choisi, de façon arbitraire, de ranger les données envoyées sur a dans un tableau suffisament grand, tab.

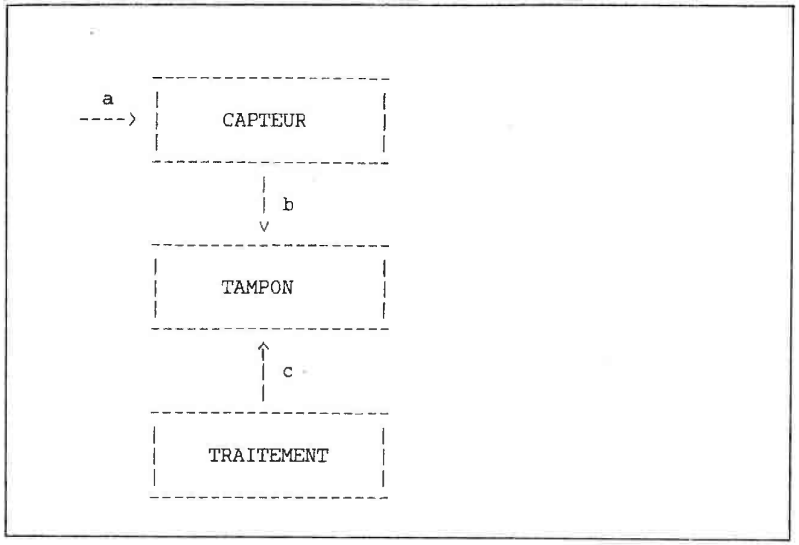
Il est très probable, dans une application concrète, que g({X1,...,Xz}) effectue le traitement des données contenues dans tab.

Cette solution ne convient au problème X que si elle vérifie X0, donc si la précision £ est garantie: si nous admettons que l'affectation située en 13 (cf. figure 7) nécessite un temps θ à l'exécution, alors il faut que

$$f(g(\{X1,...,Xz\})) + 7\theta \leq £$$

soit vérifié.

* Une seconde posibilité est d'écrire un programme parallèle, constitué de trois tâches: une tâche recevant les données sur a, une seconde jouant le rôle de tampon et une dernière effectuant le traitement séquentiel. Ces processus communiquent via deux points d'entrée, b et c, appartenant au processus tampon (cf. figure 8).



- Figure 8 -

Il est donc possible d'écrire h(X) sous la forme suivante:

```
                task capteur is
                                ->      ->
                    entry a( v : in t );

                end capteur;

                task tampon is
                                ->      ->
                    entry b( v : out t );
                                ->      ->
                    entry c( w : in t );

                end tampon;

                task traitement;

                task body capteur is
                    ->   ->
                    v' : t ;
                begin

                <<10>>
                    loop
                <<11>>                  ->      ->
                        accept a ( v : in t ) do
                <<12>>          ->      ->
                                v' := v ;
                <<13>>
                        end a;
                <<14>>                  ->
<<15>> <<16>>           tampon.b( v' );
                <<17>>
                    end loop;

                end capteur;
```

- Figure 9 -


```
                task body tampon is
                                                ->
                    tab : array(0..max) of t ;
                    in : integer := 0 ;
                    out : integer := 0 ;
                    count : integer := 0 ;
                begin
                <<m0>>
                    loop
                <<m1>>
                        select
<<m2>>                      when (count > 0) and (c'count = 0) =>
                                                ->      ->
                                accept b ( v : out t ) do
                <<m3>>                  ->
                                    v := tab(out) ;
                <<m4>>
                                end b;
                <<m5>>
                                out := (out + 1) mod (max + 1);
                <<m6>>
                                count := count - 1 ;
                        or
                            when count < max =>
                                                ->      ->
                                accept c( w : in t ) do
                <<m7>>                                  ->
                                    tab(in) := w ;
                <<m8>>
                                end c;
                <<m9>>
                                in := (in + 1) mod (max + 1);
                <<m10>>
                                count := count + 1 ;
                        end select;
                <<m11>>
                    end loop;
                end tampon;

                task body traitement is
                    ->  ->
                    v : t ;
                begin
                    loop
                                    ->
                        tampon.c(v );

                        g({X1,...,Xz});
                    end loop;
                end traitement;
```

- Figure 9 (suite) -

Ici encore, il est impératif de vérifier que la précision £ est respectée: si nous supposons que le tableau tab du processus "tampon" ne déborde jamais (dans le cas contraire il faudrait trouver une stratégie consistant à "écraser" certaines données du tableau tab, dont le temps d'exécution est minimal), il faut alors que

$$f(\text{tampon.b}(\vec{v'})) + 4\theta \leq £.$$

Calculons le premier terme de cette inégalité: $f(\text{tampon.b}(\vec{v'}))$ est égal à la somme des temps mis par le processus "capteur" pour:

- entrer dans l'état "wait" (14 --> 15),
- attendre un éventuel rendez-vous,
- exécuter le rendez-vous (15 --> 17).

Le premier d'entre eux vaut θ, par définition de notre sémantique opérationnelle; le second, quant à lui, peut être majoré par 11θ, c'est à dire le temps nécessaire pour transiter de m2 vers lui-même en exécutant la deuxième branche de l'instruction "select". Le dernier vaut évidemment 3θ, i.e. le temps nécessaire à l'exécution de la première instruction "accept" du buffer. Finalement, nous obtenons l'inégalité:

$$19\theta \leq £,$$

nous indiquant que dans ce cas la propriété X0 est garantie par h(X). Cette condition est nettement plus faible que celle obtenue dans la première solution.

Remarquons que la fonction partielle h peut aisément être étendue (pour tenir compte également des propriétés du type 9), de manière similaire à g.

V.C UNE METHODE DE PREUVE DE PROPRIETES TEMPS-REEL DANS A.R.T.

Soit IP un programme ADA, IP=(T1//...//Tn). Les propriétés 7 à 10 du paragraphe précédent, quantifiées relativement à IP constituent l'ensemble des propriétés de fatalité temps-réel de IP. Celles-ci s'écrivent à l'aide

des opérateurs temporels définis au chapitre IV, et sont, grace à la proposition du paragraphe IV.A, équivalentes à une proposition "until-temporisée". Nous pouvons ainsi exprimer toute propriété de fatalité temps-réel à l'aide de ce même opérateur temporel, nous permettant comme nous allons le voir, de raisonner dans le système axiomatique A.R.T.

Décrivons maintenant une méthode de raisonnement afin de prouver une telle propriété de fatalité.

Soit Q une propriété de fatalité temps-réel. Afin de prouver Q, il nous faut trouver dans un premier temps un découpage de IR+ en une suite infinie d'intervalles, dont les bornes marquent les instants auxquels le système IP est observable (ce sont les dates intéréssantes). Pour celà on associe, de manière purement syntaxique, à chaque instruction de chacune des tâches de IP sa durée d'exécution, et l'on projette les n axes de temps ainsi obtenus sur la demi-droite réelle IR+, obtenant ainsi l'ensemble des instants critiques pour IP. Dans la preuve de Q nous associons à chaque intervalle de temps élémentaire ainsi obtenu, une suite d'au plus n propositions "until-temporisées" décrivant chacune le comportement d'une tâche de IP pendant ce laps de temps. Ces propositions sont soit des axiomes du système A.R.T. ou des conséquences d'une règle de celui-ci. La règle de preuve R-SIMPL nous sera utile afin d'adapter un axiome à l'intervalle élémentaire considéré, en "tronquant" celui-ci à droite.

Puis nous composons les n propositions afin de n'en obtenir qu'une seule, capable de décrire le comportement de IP dans l'intervalle choisi, celà moyennant la règle R-COMP.

Finalement, nous devons composer séquentiellement les propositions résultantes, afin de décrire IP sur un intervalle de la forme $[0,x[$ avec $x \in IR+$, tel qu'il est exigé dans Q. Nous utilisons donc les règles R-SEQ ou R-INDUCTION.

V.D PREMIER EXEMPLE: SIMULATION D'UN FILTRE DE COURANT ALTERNATIF.

V.D.i Enoncé Du Problème.

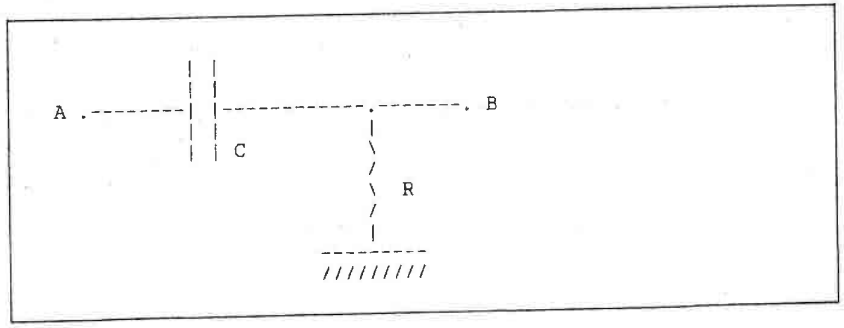
Soit le filtre de courant alternatif représenté sur la figure 10, comprenant un condensateur de capacité C une résistance de valeur R et de deux bornes A et B.

Le milieu exterieur émet un signal au point A (i.e. le passage de 0 à VA de la tension en A). Nous désirons fournir au milieu extérieur la valeur de la tension au point B: VB. Il s'agit donc d'évaluer VB en permanence, sachant qu'à la date t

$$VB = \exp(-t / (RC)).$$

En simulant ce phénomène, par un programme ADA, nous allons délivrer en B la tension VB, en permanence avec une précision $\eta \in IR+$.

Quelle est la plus petite valeur de η que nous pouvons garantir, sachant que le signal en A est quelconque et que la tâche simulant ce circuit électrique est la tâche T de la figure 11?



- Figure 10: Un filtre de courant alternatif. -

On notera par dt le plus petit délai que l'on peut demander à la primitive "delay", qui selon le manuel de Référence du langage ADA est l'attribut DURATION'DELTA.


```
                    task TA is
                        x : float ;
                    begin
                          .
                          .
                          .
                        T.a(x);
                          .
                          .
                          .
                    end TA;

                    task T is
                        dt, R, C, VA, VB : float;
                    begin
                    <<l1>>
                        while true loop
                    <<l2>>
<<l3>>                      select
                                when true => accept A( v : in float) do
<<l4>>
                                                 VA := v ;
<<l5>>
                                             end A;
                            or
                                when true => delay dt;
                            end select;
<<l6>>                      VB := VA * exp(-clock()/( R * C));
<<l7>>
<<l8>> <<l9>>               TB.B(VB) ;
<<l10>>
                        end loop;
                    end T ;

                    task TB is

                    begin
                          .
                          .
                          .
                        select
                            when true => accept B(y : in float) do
                                             null ;
                                         end B ;
                        end select;
                          .
                          .
                          .
                    end TB;
```

- Figure 11: Le système étudié. -

Modélisons complètement l'environnement dans lequel évolue T:

Le signal discontinu A étant émis par un dispositif physique, nous le décrivons par une tâche TA, désirant à certains instants entrer en rendez-vous avec T, par une instruction classique d'appel de l'entrée A (voir figure 11). Lors de l'exécution d'une telle instruction, on supposera que le paramètre x, réel positif, indique la tension au point A.

La tâche T étant censée transmettre, dans un environnement réel, sa réponse à un autre dispositif matériel (à un fil électrique, un écran cathodique,...), on supposera de plus qu'il existe une tâche réceptrice TB, toujours en attente sur un signal B (cf. figure 11), et vérifiant l'assertion temporelle:

```
|= [] (TB at wait (select
                      when true => accept B( y : in float)
                                   do
                                      null ;
                                   end B;
                   end select;)).
```

En résumé, nous devons considérer un programme ADA constitué de trois tâches, soit IP1=(TA//T//TB).

Remarquons également dans la figure 11 que nous utiliserons la primitive standard clock() indiquant l'heure absolue (relativement à une horloge globale): c'est une fonction ADA à résultat dans IR+.

V.D.ii Enoncé De La Propriété A Démontrer.

Prouvons que la tâche T est sensible au signal A de façon continue, avec une précision (dt+1).

Ceci équivaut à prouver que

A.R.T.|- [](T at 12 => ⟨⟩dt+1⟩ (T at 12)).

D'après les propriétés temporelles démontrées plus tôt, et de part la définition des opérateurs temporels (si on se place dans un modèle

équitable), on a les équivalences suivantes:

```
|= []( T at 12 => ⟨⟩dt+1⟩ (T at 12))
<=>
|= true => []( T at 12 => ⟨⟩dt+1⟩ (T at 12))
<=>
∀ t ∈ IR+, |= true --t-->(T at 12 => ⟨⟩dt+1⟩ (T at 12)),
                                    (T at 12 => ⟨⟩dt+1⟩ (T at 12))
<=>
∀ t ∈ IR+, |= true --t-->true,(T at 12 => ⟨⟩dt+1⟩ (T at 12))
<=>
∀ t ∈ IR+, |= (T at 12 ^ is t) => ⟨⟩dt+1⟩ (T at 12)
<=>
∀ t ∈ IR+, ∃ t' ∈ [0,dt+1], |= T at 12 ^ is t --t'-->true,T at 12
```

Si nous pouvons démontrer la proposition

```
(P1) | ∀ t ∈ IR+, ∃ t' ∈ [0,dt+1],                          |
     |                                                      |
     |      A.R.T.|- T at 12 ^ is t --t'-->true,T at 12    |
```

alors, par les équivalences ci-dessus et par la correction du système axiomatique A.R.T., nous aurons prouvé la propriété de fatalité temps-réel énoncée au début de ce paragraphe.

V.D.iii Preuve De La Propriété De Fatalité Temps-réel.

Soit t ∈ IR+, montrons qu'il existe t' ∈ [0,dt+1] tel que

(0) A.R.T.|- T at 12 ^ is t --t'-->true, T at 12.

Par l'axiome A-SEL1,

A.R.T.|- T at 12 --θ--> T at 12, T at 13

en appliquant R-IS et R-IMPL,

A.R.T.|- T at 12 ^ is t --θ--> T at 12, T at 13 ^ is t+θ

Distinguons deux cas, selon que TA envoit un signal au cours des dt unités de temps qui suivent l'instant t+θ ou non.

L'implication suivante est donc trivialement vérifiée.

A.R.T.|- is t+θ =>

[∃ £' ∈ [0,dt[, is t+θ --£'--> ~TA at wait(T.A), TA at wait(T.A)]

v

[∀ £' ∈ [0,dt[, is t+θ --£'--> ~TA at wait(T.A), ~TA at wait(T.A)]

V.D.iii.a Lemme 1:

Si ∃ £' ∈ [0,dt[,

A.R.T.|- is t+θ --£'-->~TA at wait(T.A), TA at wait(T.A),

alors

A.R.T.|- T at 13 ^ is t+θ --£'+(8+ϒ)θ-->true, T at 12 ^

is t+£'+(9+ϒ)θ

Fin du lemme.

Démonstration:

Soit £' ∈ [0,dt[vérifiant l'hypothèse.

On déduit que d(TA,T.A(x)) = £' = D. D'où D<dt, vérifiant ainsi les hypothèses de A-SEL2. On en déduit que:

A.R.T.|- T at 13 --£'+θ--> T at 13, T at 14 ^ TA at RV(A).

Soit par l'application des règles R-IS et R-IMPL,

(1) A.R.T.|- T at 13 ^ is t+θ --£'+θ-->true, T at ((VA:=v)[v/x]) ^ TA at RV(A) ^ is t+2θ+£'

Il est évident que (VA:=v)[v/x] est identique à (VA:=x).

Selon A-AFF, on a:

A.R.T.|- T at 14 --θ--> T at 14, T at 15

donc, de la même manière que précédement:

(2) A.R.T.|- T at 14 ^ is t+2θ+£' --θ--> true, T at 15 ^ is t+3θ+£'

De plus A-AP5 nous indique que:

A.R.T.|- ~T at 15 ^ TA at RV(A) --θ--> TA at RV(A), TA at RV(A)

or

A.R.T.|- T at 14 => ~T at 15

donc:

(3) A.R.T.|- T at 14 ^ TA at RV(A) --θ--> TA at RV(A), TA at RV(A)

Composons cette dernière avec (2) par R-COMP:

(4) A.R.T.|- T at 14 ^ is t+2θ+£' ^ TA at RV(A) --θ--> TA at RV(A),

TA at RV(A) ^ T at 15 ^ is t+3θ+£'

La règle A-SEL7 nous dit que:

A.R.T.|- T at 15 ^ TA at RV(A) --θ--> T at 15 ^ TA at RV(A),

T at 16 ^ T after RV(A)

Appliquons successivement R-IS et R-IMPL à cette dernière assertion:

(5) A.R.T.|- T at 15 ^ TA at RV(A) ^ is t+3θ+£' --θ--> true, T at 16 ^ is t+4θ+£'

Or le temps d'exécution de l'affectation située en 16 est de:

θ + f(VA*exp(-clock()/(R*C)))

= θ + 2θ + f(VA) + f(exp(-clock()/(R*C)))

= 3θ + te + f(-clock()/(R*C))

= 3θ + te + θ + 2θ + 2θ + θ = 9θ + te = ϒθ

(pour un certain ϒ ∈ IR+), où te est le temps d'évaluation d'une fonction exponentielle, et où nous avons supposé que la fonction "clock()" nécessite un temps de θ. Selon A-AFF on a donc:

A.R.T.|- T at 16 --ϒθ--> T at 16, T at 17

Ce qui donne, grace à R-IS et R-IMPL:

(6) A.R.T.|- T at 16 ^ is t+4θ+£' --ϒθ--> true,T at 17 ^ is t+£'+(4+ϒ)θ

Grace aux règles A-AP1, R-IS et R-IMP on peut écrire que:

(7) A.R.T.|- T at 17 ^ is t+£'+(4+ϒ)θ --θ--> true, T at 18 ^ is t+£'+(5+ϒ)θ

Or nous avons supposé dans les hypothèses de cette preuve que

|= [] (TB at wait(select ... end select))

Ce qui permet d'affirmer que:

A.R.T.|- is t+£'+(5+ϒ)θ => TB at wait(select ... end select)

Or selon A-AP2:

A.R.T.|- T at 18 ^ TB at wait(select...end) --θ--> T at 18 ^ TB at wait(select...end), T at 19

et selon A-SEL4 (où D=0):

A.R.T.|- TB at wait(select...end select) --θ--> TB at wait(select...end select), TB at null ^ T at 19

Donc, en composant ces deux dernières par R-COMP puis en appliquant R-IS et R-IMPL au résultat:

(8) A.R.T.|- T at 18 ^ is t+£'+(5+ϒ)θ --θ--> true, T at 19 ^ TB at null ^ is t+£'+(6+ϒ)θ

Par l'axiome A-NULL:

(9) A.R.T.|- TB at null --θ--> TB at null, TB after null

Or, par A-AP5:

A.R.T.|- T at 19 ^ ~TB at (end B) --θ--> T at 19, T at 19

et comme

A.R.T.|- TB at null => ~TB at (end B), on a

A.R.T.|- T at 19 ^ TB at null --θ--> T at 19, T at 19

Composons cette dernière avec (9), puis appliquons R-IS et R-IMPL successivement au résultat:

(10) A.R.T.|- TB at null ^ T at 19 ^ is t+£'+(6+ϒ)θ --θ--> true, T at 19 ^ TB after null ^ is t+f'+(7+ϒ)θ

Selon A-SEL8,

A.R.T.|- TB at (end B) ^ T at 19 --θ--> TB at (end B) ^ T at 19, TB after (end B) ^ T at 110

Appliquons une fois de plus R-IS et R-IMPL,

(11) A.R.T.|- TB at (end B) ^ T at 19 ^ is t+£'+(7+ϒ)θ --θ--> true, TB after (select...end select) ^ T at 110 ^ is t+£'+(8+ϒ)θ

L'axiome A-WHILE3 nous permet d'écrire encore que:

A.R.T.|- T at 110 --θ--> T at 110, T at 12

d'où on obtient par R-IS at R-IMPL:

(12) A.R.T.|- T at 110 ^ is t+£'+(8+ϒ)θ --θ--> true, T at 12 ^ is t+£'+(9+ϒ)θ

Composons successivement les assertions (1), (4)-(8), (10)-(12) par la règle R-SEQ et appliquons R-IMPL au résultat final afin de simplifier le deuxième membre:

A.R.T.|- T at 13 ^ is t+θ --£'+(8+ϒ)θ--> true, T at 12 ^ is t+£'+(9+ϒ)θ

C.Q.F.D.

V.D.iii.b Lemme 2:

Si ∀ £' ∈ [0,dt[,

A.R.T.|- is t+θ --£'--> ~TA at wait(T.A(x)), ~TA at wait(T.A(x)),

alors

A.R.T.|- T at 13 ^ is t+θ --dt+(5+ϒ)θ--> true, T at 12 ^ is dt+(6+ϒ)θ.

Fin du lemme.

Démonstration:

Appliquons l'axiome A-SEL3 en sachant par hypothèse que D≥dt:

A.R.T.|- T at 13 --dt--> T at 13, T at 16

Grace aux règles R-IS et R-IMPL, on obtient:

(13) A.R.T.|- T at 13 ^ is t+θ --dt--> true, T at 16 ^ is t+θ+dt

Si nous supposons que l'affectation en 16 dure un temps égal à ϒθ:

A.R.T.|- T at 16 --ϒθ--> T at 16, T at 17, d'où

(14) A.R.T.|- T at 16 ^ is t+θ+dt --ϒθ--> true, T at 17 ^ is t+dt+(1+ϒ)θ

Grace à A-AP1, R-IS et R-IMPL on a:

(15) A.R.T.|- T at 17 ^ is t+dt+(1+ϒ)θ --θ--> true, T at 18 ^ is t+dt+(2+ϒ)θ

Grace à l'hypothèse

|= [] (TB at wait(select...end select)), on a:

A.R.T.|- is t+dt+(2+ϒ)θ => (TB at wait(select...end select))

Selon A-AP2:

A.R.T.|- T at 18 ^ TB at wait(select...end select) --θ--> T at 18 ^ TB at wait(select...end select) , T at 19

Par A-SEL4,

A.R.T.|- TB at wait(select...end select) --θ--> TB at wait(select...end select), TB at null ^ T at 19

Ce qui donne par R-COMP, R-IS et R-IMPL:

(16) A.R.T.|- T at 18 ^ is t+dt+(2+ϒ)θ --θ--> true, T at 19 ^ TB at null ^ is t+dt+(3+ϒ)θ

De même que dans le premier lemme, par A-NULL at A-AP5 on a:

A.R.T.|- TB at null --θ--> TB at null, TB after null

A.R.T.|- T at 19 ^ ~TB at (end B) --θ-->T at 19, T at 19

or

A.R.T.|- TB at null => ~TB at (end B),

donc on a par R-IMPL et R-COMP:

A.R.T.|- T at 19 ^ TB at null --θ--> true, T at 19 ^ TB after null

Par R-IS et R-IMPL celà donne:

(17) A.R.T.|- T at 19 ^ TB at null ^ is t+dt+(3+ϒ)θ --θ--> true, T at 19 ^ TB after null ^ is t+dt+(4+ϒ)θ

Par R-SEL8, R-IS et R-IMPL on a maintenant:

(18) A.R.T.|- TB at (end B) ^ T at 19 ^ is t+dt+(4+ϒ)θ --θ--> true, T at 110 ^ is t+dt+(5+ϒ)θ

Par A-WHILE3, R-IS et R-IMPL:

(19) A.R.T. |- T at 110 ^ is t+dt+(5+ϒ)θ --θ--> true, T at 12 ^ is t+dt+(6+ϒ)θ

Si nous composons par R-SEQ les assertions (13) à (19), nous avons finalement:

A.R.T.|- T at 13 ^ is t+θ --dt+(5+ϒ)θ--> true, T at 12 ^ is t+dt+(6+ϒ)θ

C.Q.F.D.

Finalement, nous pouvons dire que par R-SEL1:

A.R.T.|- T at 12 --θ--> T at 12, T at 13

d'où par R-IS et R-IMPL:

A.R.T.|- T at 12 ^ is t --θ--> true, T at 13 ^ is t+θ

En considérant les deux cas discutés dans les deux lemmes précédents, on obtient par R-SEQ et R-IMPL les deux assertions suivantes:

[Si ∃ £' ∈ [0,dt[,

A.R.T.|- is t+θ --£'--> ~TA at wait(T.A(x)), TA at wait(T.A(x))

alors

A.R.T.|- T at 12 ^ is t --£'+(9+ϒ)θ--> true, T at 12].

[Si ∀ £' ∈ [0,dt[,

|- A.R.T.|- is t+θ --£'--> ~TA at wait(T.A(x)), ~TA at wait(T.A(x))

alors

A.R.T.|- T at 12 ^ is t --dt+(6+ϒ)θ--> true, T at 12].

Dans le premier cas nous pouvons donc poser

$$t' = £' + (9+ϒ)\theta,$$

et dans le second, on posera

$$t' = dt + (6+ϒ)\theta.$$

Comme nous pouvons prendre te = 10θ (voir la définition de f au chapitre III), alors nous avons dans les deux cas: $t' \in [0,dt+1]$, ce qui achève la démonstration de la proposition (P1) et permet de répondre à la question initiale en disant que la plus petite valeur de ϒ que nous pouvons garantir est

$$ϒ = \max(£'+(9+ϒ)\theta, dt+(6+ϒ)\theta) \leq dt + (9+ϒ)\theta = dt +$$

28θ

V.E SECOND EXEMPLE: UN ADDITIONNEUR.

V.E.i Enoncé Du Problème.

Considérons un additionneur réalisant l'addition de deux mots de n bits, constitué de n additionneurs élémentaires fonctionnant en parallèle. Les mots de données parviennent à l'additionneur par deux bus de n fils et le résultat est déposé sur un troisième bus de n fils.
Modélisons les constituants de notre problème par des tâches ADA:

1) les étage-additionneurs seront représentés par n tâches A1,...,An respectivement;

2) les deux bus de données quant à eux, sont modélisés par les tâches bus_a1,...,bus_an, bus_b1,...,bus_bn;

3) le bus véhiculant le résultat de l'addition sera représenté par les tâches bus_s1,...,bus_sn. (Voir figure 12).

Globalement, nous devons considérer le système suivant de 4n tâches: IP2=(A1//...//An//bus_a1//...//bus_bn//bus_s1//...//bus_sn).
Cet additionneur opère de manière classique:

1) Les signaux de données sont stables pendant une durée ϒ∈ IR+ (suffisament longue), pendant laquelle ils peuvent être lus par les étage-additionneurs. Ce comportement est simulé par une instruction d'appel de point d'entrée limité dans le temps.

2) De même, les tâches (bus_si)i∈{1,...,n} attendent pendant une durée ϒ' leur résultat respectif, ce qui se modélise aisément par une instruction "select" temporisée.

3) Les étage-additionneurs quant à eux, lisent successivement leurs données puis la retenue provenant de l'étage précédent (ce qui ne concerne bien entendu pas le premier étage). Ensuite, ils calculent le résultat de l'addition qu'ils émettent immédiatement au bus des résultats, puis la retenue qu'ils transmettent alors à l'étage suivant (cette dernière opération n'est pas réalisée par An).

Ayant anoté notre programme IP2 de manière cohérente et simple, posons nous le problème de mesurer le temps de réponse de l'additionneur. En d'autres termes, quel est le temps mis par IP2 pour calculer le résultat de l'addition de deux mots de n bits, depuis l'instant où toutes les données sont présentes sur les bus, jusqu'au moment où tous les bits du mot résultat sont émis?

```
        -- ∀ i ∈ {2,...,n-1}, Ai a la forme suivante:

        task Ai is
              vi,vai,vbi,vci,vsi : integer;
        begin
        <<10>>    select
<<11>>              when true =>  accept first_data( vi: in integer ) do
        <<12>>                              vai:=vi;
        <<13>>                    end first_data;
                  end select;
        <<14>>    select
<<15>>              when true =>  accept second_data( vi: in integer ) do
        <<16>>                              vbi:=vi;
        <<17>>                    end second_data;
                  end select;
        <<18>>    select
<<19>>              when true =>  accept carry( vi : in integer ) do
        <<110>>                             vci:=vi;
        <<111>>                   end carry;
                  end select;
        <<112>>
                  vsi:= (vai + vbi + vci) mod 2 ;
        <<113>>
<<114>> <<115>>   bus_si.send(vsi);
        <<116>>
                  vci:= (vai + vbi + vci) / 2 ;
        <<117>>
<<118>> <<119>>   A(i+1).carry(vci);
        <<120>>

        end Ai;
```

- Figure 12: L'additionneur. -


```
        -- La tâche A1 est légèrement différente, car elle ne reçoit
        -- pas de retenue de l'étage-additionneur précédent.

        task A1 is
              v1,va1,vb1,vc1,vs1 : integer ;
        begin
        <<10>>  select
        <<11>>        accept first_data ( v1 : in integer ) do
        <<12>>              va1:= v1;
        <<13>>        end first_data;
                  end select;
        <<14>>    select
        <<15>>        accept second_data ( v1: in integer ) do
        <<16>>              vb1:=v1;
        <<17>>        end second_data;
                  end select;
        <<18>>    vs1 := (va1 + vb1 + vc1) mod 2 ;
        <<113>>
<<114>> <<115>> bus_s1.send(vs1);
        <<116>> vc1:=(va1 + vb1 + vc1) / 2 ;
        <<117>>
<<118>> <<119>> A2.carry(vc1);
        <<120>>
        end A1;

        -- La tâche An est également différente des Ai précédentes
        -- car elle ne calcule pas de retenue.

        task An is
              van,vbn,vcn,vsn : integer;
        begin
        <<10>>        select
        <<11>>              accept first_data( vn: integer ) do
        <<12>>                    van:=vn;
        <<13>>              end first_data;
                      end select;
        <<14>>        select
        <<15>>              accept second_data( vn: integer ) do
        <<16>>                    vbn:=vn;
        <<17>>              end second_data;
                      end select;
        <<18>>        select
        <<19>>              accept carry( vn : integer ) do
        <<110>>                   vcn:=vn;
        <<111>>             end carry;
                      end select;

        <<112>>   vsn:= (van + vbn + vcn) mod 2 ;
        <<113>>
<<114>> <<115>>   bus_sn.send(vsn);
        <<116>>
        end Ai;
```

- Figure 12 (suite): L'additionneur -

```
        -- ∀ i ∈ {1,...,n},la tâche bus_ai sera la suivante:

        task bus_ai is
              xi : integer;
        begin
        ⟨⟨10⟩⟩
⟨⟨11⟩⟩
              select
        ⟨⟨12⟩⟩          Ai.first_data(xi);
              or
                        delay  ¥;
              end select;
        ⟨⟨13⟩⟩
        end bus_ai;

        -- ∀ i ∈ {1,...,n},la tâche bus_bi sera écrite de
        -- manière  suivante:

        task bus_bi is
              yi : integer;
        begin
        ⟨⟨10⟩⟩
⟨⟨11⟩⟩
              select
        ⟨⟨12⟩⟩          Ai.second_data(yi);
              or
                        delay¥ ;
              end select;
        ⟨⟨13⟩⟩
        end bus_bi;

        -- ∀ i ∈ {1,...,n}, bus_si possèdera la composition
        -- suivante:

        task bus_si is
              wi,zi : integer ;
        begin
        ⟨⟨10⟩⟩
⟨⟨11⟩⟩        select

                  when true =>  accept send( wi : integer) do
                                    zi:=wi;
        ⟨⟨12⟩⟩
        ⟨⟨13⟩⟩                end send;
                     or
                  when true =>  delay   ¥ ';

              end select;
        ⟨⟨14⟩⟩
        end bus_si;
```

- Figure 12 (suite): L'additionneur -

V.E.ii Enoncé De La Propriété A Démontrer.

Prouvons la propriété de "délai de réponse" suivante:

$$\text{Si } P = \bigwedge_{i=1}^{n} (bus\_ai \text{ at } 10 \wedge bus\_bi \text{ at } 10),$$

et

$$Q = \bigwedge_{i=1}^{n} (Ai \text{ after } bus\_si.send(vsi))$$

alors

$$\models \Box (P \Rightarrow \langle\langle (17n-1)\theta \rangle\rangle Q)$$

Ceci équivaut encore à prouver que:

(P2) $\forall t \in IR+, \models P \wedge \text{is } t \Rightarrow \langle\langle (17n-1)\theta \rangle\rangle Q.$

V.E.iii Preuve De La Propriété De Fatalité Temps-réel.

Nous allons démontrer une propriété plus forte que (P2) à savoir la suivante:

V.E.iii.a Propriété Forte:

$\models$ at IP2 --(17n-1)θ--> true, FINISH

avec

$$FINISH = \left(\bigwedge_{i=1}^{n-1} Ai \text{ at } 120\right) \wedge (An \text{ at } 116) \wedge (\text{is } (17n-1)\theta)$$

Fin de la propriété.

Montrons dans un premier lemme que si cette dernière propriété est démontrable dans A.R.T., alors (P2) est verifiée.

V.E.iii.b Lemme 1:

Si A.R.T.$\vdash$ at IP2 --(17n-1)θ--> true, FINISH,

alors la propriété (P2) est valide.

Fin du lemme.

Démonstration:

* Tout d'abord nous remarquons que at IP2 n'est vérifiée que pendant l'intervalle de temps [0,θ[, ceci car par l'axiome A-SEL1 on a:

∀ i ∈ {1,...,n},

A.R.T.|- Ai at 10 --θ--> Ai at 10, Ai at 11

de même on a

∀ i ∈ {1,...,n},

A.R.T.|- bus_si at 10 --θ--> bus_si at 10, bus_si at 11

Grace à A-AP1 nous savons de même que

∀ i ∈ {1,...,n},

(1) A.R.T.|- bus_ai at 10 --θ--> bus_ai at 10, bus_ai at 11

(2) A.R.T.|- bus_bi at 10 --θ--> bus_bi at 10, bus_bi at 11

d'où, par R-IS, R-COMP et R-SIMPL

$$A.R.T.|- \text{at IP2} \wedge \text{is } 0 \text{ --}\theta\text{--> at IP2, is } \theta \wedge \bigwedge_{i=1}^{n} (Ai \text{ at } 10 \wedge bus\_ai \text{ at } 10 \wedge bus\_bi \text{ at } 10 \wedge bus\_si \text{ at } 10)$$

Vu la structure de IP2, où l'on remarque que l'on n'atteint plus jamais le point de contrôle (10,...,10), on en déduit que

(3) A.R.T.|- (∃ t ∈ [0,θ[, is t) => at IP2

Si on ne considère que les assertions (1) et (2), on déduit également que:

$$A.R.T.|- P \wedge \text{is } 0 \text{ --}\theta\text{--> } P, \text{ is } \theta \wedge \bigwedge_{i=1}^{n} (bus\_ai \text{ at } 11 \wedge bus\_bi \text{ at } 11)$$

donc que P est vérifiée uniquement dans l'intervalle de temps [0,θ[.

(4) A.R.T.|- P => (∃ t ∈ [0,θ[, is t).

D'où finalement, par (3) et (4):

(5) A.R.T.|- P => at IP2

* Il est évident que

$$\text{FINISH} \Rightarrow \left(\bigwedge_{i=1}^{n-1} Ai \text{ after } bus\_si.send(vsi)\right) \wedge (An \text{ after } bus\_sn.send(vsn))$$

$$= \bigwedge_{i=1}^{n} Ai \text{ after } bus\_si.send(vsi) = Q.$$

D'où par R-HYP,

(6) A.R.T.|- FINISH => Q

* Appliquons la règle R-IMPL à l'hypothèse de ce lemme ainsi qu'à (5) et (6):

A.R.T.|- P --(17n-1)θ--> true, Q.

Soit t ∈ IR+, on a par R-IS:

A.R.T.|- P ∧ is t --(17n-1)θ--> true, Q

ce qui équivaut à

∀ t ∈ IR+, A.R.T.|- P ∧ is t => ⟨(17n-1)θ⟩ Q

c'est à dire la proposition (P2), par correction du système A.R.T.

C.Q.F.D.

Afin de démontrer la proposition forte, nous prouverons les trois assertions suivantes:

(7) A.R.T.|- at IP2 --x0--> true, P(n-2)

(8) A.R.T.|- P(n-2) --x1--> true, P0

(9) A.R.T.|- P0 --x2--> true, FINISH

avec x0 = 22θ, x1 = 17(n-2)θ, x2 = 11θ,

∀ k ∈ {0,...,n-2},

$$Pk = \left(\bigwedge_{i=1}^{n-k-2} Ai \text{ at } 120\right) \wedge (A(n-k-1) \text{ at } 118) \wedge \left(\bigwedge_{i=n-k}^{n} Ai \text{ at } 19\right) \wedge \left(\bigwedge_{i=1}^{n-k-1} bus\_si \text{ at } 14\right) \wedge \left(\bigwedge_{i=n-k}^{n} bus\_si \text{ at } 11\right) \wedge AB \wedge (\text{is } tk),$$

$$AB = \bigwedge_{i=1}^{n} (bus\_ai \text{ at } 13 \wedge bus\_bi \text{ at } 13)$$

∀ k ∈ {0,...,n-2}, tk = (17(n-k-1)+5)θ.

Une fois que nous aurons démontré celà, il suffira d'appliquer R-SEQ à (7), (8) et (9):

A.R.T.|- at IP2 --x0+x1+x2--> true, FINISH

d'où nous déduisons, par définition de x0, x1 et x2, ainsi que par la correction de A.R.T. que:

|= at IP2 --(17n-1)θ--> true, FINISH

c'est à dire la propriété forte.

Démontrons maintenant les assertions (7) à (9) dans les trois lemmes suivants.

V.E.iii.c Lemme 2:

A.R.T.|- at IP2 --x0--> true, P(n-2)

Fin du lemme.

Démonstration:

Nous allons démontrer la proposition que voici:

$$A.R.T.|\text{- at IP2 --2}\theta\text{--> true, A1 at 118} \wedge (\wedge_{i=2}^{n} \text{Ai at 19}) \wedge (\text{bus\_sl at 14}) \wedge (\wedge_{i=2}^{n} \text{bus\_si at 11}) \wedge \wedge_{i=1}^{n} (\text{bus\_ai at 13} \wedge \text{bus\_bi at 13}) \wedge \text{is 2}\theta$$

Ce faisant nous traduisons le comportement du programme IP2 au cours de l'intervalle de temps [0,2θ[.

Au cours de ce dernier, les tâches A2,...,An se déplacent depuis leur point de contrôle 10 à 19, en lisant les deux bits de données, puis attendront leur retenue. Pendant ce même temps, A1 se déplacera depuis 10 jusqu'en 118, lisant ses données et émettant le résultat de l'addition. Une fois arrivée en 118, A1 sera capable de transmettre sa retenue à A2, afin que l'étage suivant puisse commencer ses calculs. De cette manière, il est évident que les retenues se propagent d'étage en étage et qu'ainsi chacun d'entre eux est obligé d'attendre la fin des calculs de son prédécesseur avant de pouvoir commencer les siens. Au cours de ce même segment temporel, il est évident que les tâches (bus_ai)i∈{1,...,n} et (bus_bi)i∈{1,...,n} progressent depuis leur

point de contrôle 10 jusqu'en 13, celà afin de communiquer leurs données. Leur travail sera alors achevé. On remarquera aussi que les tâches (bus_si)i∈{2,...,n} attendront leurs résultats respectifs en 11.

Afin de faciliter la présente preuve, découpons l'intervalle [0,22θ[en trois sous-intervalles: [0,8θ[, [8θ,9θ[et [9θ,22θ[.

* L'intervalle de temps [0,8θ[:

Découpons cet intervalle en huit sous-intervalles de longueur θ.

- Pendant le premier sous-intervalle, toutes les tâches se mettent en état d'attente: par l'axiome A-SEL1,

(0) ∀ i ∈ {1,...,n}, A.R.T.|- Ai at 10 --θ--> Ai at 10, Ai at 11

et

A.R.T.|- bus_si at 10 --θ--> bus_si at 10, bus_si at 11.

Grace à A-SA1,

∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at 10 --θ--> bus_ai at 10, bus_ai at 11 et

A.R.T.|- bus_bi at 10 --θ--> bus_bi at 10, bus_bi at 11.

Composons ces 4n assertions par R-COMP et appliquons R-IS et R-IMPL au résultat, de la même manière que d'habitude:

$$(1)\ A.R.T.|\text{- at IP2} \wedge \text{is 0 --}\theta\text{--> true, is } \theta \wedge \wedge_{i=1}^{n} (\text{bus\_ai at 11} \wedge \text{bus\_bi at 11} \wedge \text{Ai at 11} \wedge \text{bus\_si at 11}).$$

- Pendant le second sous-intervalle [θ,2θ[, les tâches Ai et bus_ai entrent en rendez-vous et les 2n tâches restantes se maintiennent en 11. Selon A-SEL4 où l'on a D=d(bus_ai,Ai.first_data(xi))=0,

∀ i ∈ {1,...,n},

(1') A.R.T.|- Ai at 11 --θ--> Ai at 11, Ai at (vai:=vi)[vi/xi]

^ bus_ai at RV(first_data).

L'axiome A-SA2 nous renseigne sur le comportement des tâches

(bus_bi)i∈{1,...,n}, qui peuvent entrer en rendez-vous avant la date 5θ:

∀ i ∈ {1,...,n},

A.R.T.|- bus_bi at l1 --ϒ-->

[(bus_bi at l1 ^ ~(Ai at (accept second_data...end)) v

(bus_bi at l2 ^ Ai in (accept second_data...end))] v

[bus_bi at l3 ^ (Ai at l{8,...,20})],

[(bus_bi at l2 ^ (Ai in (accept second_data...end)) v

(bus_bi at l3)]

ce qui équivaut à

(2) ∀ i ∈ {1,...,n},

A.R.T.|- bus_bi at l1 --ϒ-->

[(bus_bi at l1 ^ ~Ai at l{6,7}) v (bus_bi at l2 ^ Ai at l{6,7})]

v [bus_bi at l3 ^ Ai at l{8,...,20}],

[(bus_bi at l2 ^ Ai at l{6,7}) v (bus_bi at l3)]

Par R-SIMPL on en déduit que

(3) ∀ i ∈ {1,...,n},

A.R.T.|- bus_bi at l1 --5θ-->

[(bus_bi at l1 ^ ~Ai at l{6,7}) v (bus_bi at l2 ^ Ai at l{6,7})

v (bus_bi at l3 ^ Ai at l{8,...,20})],

[(bus_bi at l1 ^ ~Ai at l{6,7}) v (bus_bi at l2 ^ Ai at l{6,7})

v (bus_bi at l3 ^ Ai at l{8,...,20})].

Nous verrons ultérieurement qu'il est possible de préciser le comportement des tâches (bus_bi)i∈{1,...,n}, en connaissant celui des tâches (Ai)i∈{1,...,n}; nous déduirons à partir de (3) que les tâches (bus_bi)i∈{1,...,n} se trouvent en l1 pendant le segment de temps [θ,5θ].

L'axiome A-SEL4 nous permet d'écrire au sujet des tâches

(bus_si)i∈{1,...,n} que:

(4) ∀ i ∈ {1,...,n},

A.R.T.|- bus_si at l1 --t(n-i)+8θ--> bus_si at l1, bus_si at l2 ^ Ai at l15

donc, nous en déduisons, par R-SIMPL que

(5) ∀ i ∈ {1,...,n},

A.R.T.|- bus_si at l1 --7θ--> bus_si at l1, bus_si at l1

Cette assertion explique que les tâches (bus_si)i∈{1,...,n} se maintiennent en l1 pendant 7θ secondes à partir de la date θ. Nous savons finalement que par A-SA2:

∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at l1 --ϒ-->

[(bus_ai at l1 ^ ~(Ai at (accept first_data...end)) v

(bus_ai at l2 ^ Ai in (accept first_data...end))] v

[bus_ai at l3 ^ (Ai at l{4,...,20})],

[(bus_ai at l2 ^ (Ai in (accept first_data...end)) v

(bus_ai at l3)]

soit encore,

∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at l1 --ϒ -->

[(bus_ai at l1 ^ ~(Ai at l{2,3})) v (bus_ai at l2 ^ (Ai at l{2,3}))

v (bus_ai at l3 ^ Ai at l{4,...,20})],

[(bus_ai at l2 ^ Ai at l{2,3}) v (bus_ai at l3)]

Donc, comme ϒ est suffisament grand par rapport à θ, on a par R-SIMPL:

(6) ∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at l1 --θ-->

[(bus_ai at l1 ^ ~Ai at l{2,3}) v (bus_ai at l2 ^ Ai at l{2,3})

v (bus_ai at 13 ^ Ai at 1[4,...,20})],

[(bus_ai at 11 ^ ~Ai at 1[2,3}) v (bus_ai at 12 ^ Ai at 1[2,3})

v (bus_ai at 13 ^ Ai at 1[8,...,20})].

Composons (1') et (6) par R-COMP puis simplifions par R-IMPL:

(7) ∀ i ∈ {1,...,n},

A.R.T.|- (Ai at 11 ^ bus_ai at 11) --θ--> true, (Ai at 12 ^ bus_ai at 12)

(Nous avons en fait utilisé implicitement les assertions

Ai at 12 ^ ~(Ai at 1[2,3}) = false

Ai at 12 ^ (Ai at 1[2,3}) = Ai at 12

Ai at 12 ^ (Ai at 1[4,...,20}) = false,

ce qui a notablement simplifié certaines assertions intermédiaires).

Composons par R-COMP les n assertions obtenues en (7), puis appliquons R-IS et R-IMPL:

(8) $A.R.T.|- \bigwedge_{i=1}^{n} (Ai \text{ at } 11 \wedge bus\_ai \text{ at } 11) \wedge \text{is } \theta \text{ --}\theta\text{--> true}, \bigwedge_{i=1}^{n} (bus\_ai \text{ at } 12 \wedge Ai \text{ at } 12) \wedge \text{is } 2\theta.$

- Les tâches (bus_ai)i∈{1,...,n} attendent maintenant la fin du rendez-vous, ce qui est traduit par la règle A-SA4:

(9) ∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at 12 ^ ~Ai at 13 --θ--> bus_ai at 12, bus_ai at 12

Il est évident que

(10) A.R.T.|- Ai at 12 => ~Ai at 13,

et de plus par A-AFF,

(11) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 12 --θ--> Ai at 12, Ai at 13

En appliquant R-IMPL à (9) et (10), puis en composant les n résultats obtenus en (11) par la règle R-COMP,

(12) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 12 ^ bus_ai at 12 --θ--> Ai at 12 ^ bus_ai at 12, Ai at 13 ^ bus_ai at 12

Appliquons R-IS à cette dernière, puis R-IMPL et composons les n assertions résultantes par R-COMP:

(13) $A.R.T.|- \bigwedge_{i=1}^{n} (Ai \text{ at } 12 \wedge bus\_ai \text{ at } 12) \wedge \text{is } 2\theta \text{ --}\theta\text{--> true}, \bigwedge_{i=1}^{n} (Ai \text{ at } 13 \wedge bus\_ai \text{ at } 12) \wedge \text{is } 3\theta$

- De plus, par A-SEL8,

(14) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 13 ^ bus_ai at 12 --θ--> Ai at 13 ^ bus_ai at 12, Ai at 14 ^ bus_ai at 13

En composant ces n assertions par R-COMP et en appliquant R-IS et R-IMPL,

(15) $A.R.T.|- \bigwedge_{i=1}^{n} (Ai \text{ at } 13 \wedge bus\_ai \text{ at } 12) \wedge \text{is } 3\theta \text{ --}\theta\text{--> true}, \bigwedge_{i=1}^{n} (Ai \text{ at } 14 \wedge bus\_ai \text{ at } 13) \wedge \text{is } 4\theta.$

Dorénavant les tâches (bus_ai)i∈{1,...,n} resteront en 13 et nous utiliserons l'assertion, due à A-END que voici:

(16) ∀ i ∈ {1,...,n}, ∀ t' ∈ IR+,

A.R.T.|- bus_ai at 13 --t'--> bus_ai at 13, bus_ai at 13

- Maintenant les tâches (Ai)i∈{1,...,n} se préparent à lire leur seconde donnée. De part l'axiome A-SEL1 on a

(17) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 14 --θ--> Ai at 14,Ai at 15

Prenons t'=θ dans (16) et écrivons encore

(18) ∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at 13 --θ--> bus_ai at 13, bus_ai at 13

En composant (17) et (18), puis en composant les n résultats obtenus ainsi en faisant varier i de 1 à n, on a

(19) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 14 ^ bus_ai at 13) --θ--> $\bigwedge_{i=1}^{n}$ (Ai at 14 ^ bus_ai at 13), $\bigwedge_{i=1}^{n}$ (Ai at 15 ^ bus_ai at 13)

appliquons R-IS et R-IMPL,

(20) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 14 ^ bus_ai at 13) ^ is 4θ --θ--> true, $\bigwedge_{i=1}^{n}$ (Ai at 15 ^ bus_ai at 13) ^ is 5θ.

- Les tâches Ai et bus_bi peuvent, pour tout i de {1,...,n}, entrer en rendez-vous immédiatement à partir de la date 5θ. Par A-SEL4 où D=0, on peut écrire

(21) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 15 --θ--> Ai at 15,

Ai at 16 ^ bus_bi at RV(second_data).

De même que précédement l'assertion (16) nous dit que

(22) ∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at 13 --θ--> bus_ai at 13, bus_ai at 13

soit en composant les 2n assertions indiquées en (21) et (22) puis en appliquant R-IS et R-IMPL,

(23) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 15 ^ bus_ai at 13) ^ is 5θ --θ--> true, $\bigwedge_{i=1}^{n}$ (Ai at 16 ^ bus_bi at RV(second_data) ^ bus_ai at 13) ^ is 6θ.

Appliquons R-SEQ aux assertions (8), (13), (15), (20) et (23),

(24) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 11 ^ bus_ai at 11) ^ is θ --5θ--> true, $\bigwedge_{i=1}^{n}$ (Ai at 16 ^ bus_bi at RV(second_data) ^ bus_ai at 13) ^ is 6θ.

Composons par R-COMP les n assertions indiquées en (3), spécifiant ainsi le comportement des tâches (bus_bi)i∈{1,...,n} pendant un intervalle de temps de durée 5θ, puis composons le résultat avec (24), en appliquant R-IMPL et R-COMP:

(25) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 11 ^ bus_ai at 11 ^ bus_bi at 11) ^ is θ --5θ-->true, $\bigwedge_{i=1}^{n}$ (Ai at 16 ^ bus_bi at RV(second_data) ^ bus_ai at 13) ^ is 6θ.

- Pendant les θ secondes qui suivent la date 6θ, l'intruction d'acceptation du rendez-vous est exécutée. Grace à A-AFF,

(26) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 16 --θ--> Ai at 16, Ai at 17

L'assertion (16) permet encore d'écrire que:

(27) ∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at 13 --θ--> bus_ai at 13, bus_ai at 13

Par A-SA4,

(28) ∀ i ∈ {1,...,n},

A.R.T.|- bus_bi at RV(second_data) ^ ~Ai at 17 --θ--> bus_bi at RV(second_data), bus_bi at RV(second_data)

En composant les 3n assertions ci-dessus par R-COMP, puis en utilisant R-IS, R-IMPL et l'implication que voici

(29) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 16 => ~Ai at 17, on obtient donc

(30) A.R.T.|- $\bigwedge_{i=1}^{n}$ (bus_ai at 13 ^ bus_bi at 12 ^ Ai at 16) ^ is 6θ --θ--> true, $\bigwedge_{i=1}^{n}$ (bus_ai at 13 ^ bus_bi at 12 ^ Ai at 17) ^ is 7θ

- Finalement, le rendez-vous s'achève à la date 8θ. Par A-SEL8, il est évident que

(31) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 17 ^ bus_bi at 12 --θ--> Ai at 17 ^ bus_bi at 12, Ai at 18

Par A-SA3 on a encore

(32) ∀ i ∈ {1,...,n},

A.R.T.|- bus_bi at 12 ^ Ai at 17 --θ--> bus_bi at 12 ^ Ai at 17, bus_bi at 13

Soit en composant les 3n assertions indiquées en (31), (32) et (27), puis en appliquant R-IS et R-IMPL au résultat:

(33) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 17 ^ bus_ai at 13 ^ bus_bi at 12) ^ is 7θ --θ--> true, $\bigwedge_{i=1}^{n}$ (Ai at 18 ^ bus_ai at 13 ^ bus_bi at 13) ^ is 8θ

Composons séquentiellement les assertions (25), (30) et (33)

(34) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 11 ^ bus_ai at 11 ^ bus_bi at 11) ^ is θ --7θ--> $\bigwedge_{i=1}^{n}$ (Ai at 18 ^ bus_ai at 13 ^ bus_bi at 13) ^ is 8θ

Composons les n assertions indiquées en (5) et simplifions le résultat,

(35) A.R.T.|- $\bigwedge_{i=1}^{n}$ (bus_si at 11) --7θ--> true, $\bigwedge_{i=1}^{n}$ (bus_si at 11)

Appliquons R-COMP à (34) et (35),

(36) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 11 ^ bus_ai at 11 ^ bus_bi at 11 ^ bus_si at 11) ^ is θ --7θ--> true, $\bigwedge_{i=1}^{n}$ (Ai at 18 ^ bus_ai at 13 ^ bus_bi at 13 ^ bus_si at 11) ^ is 8θ

Enfin, composons séquentiellement cette dernière avec (1):

(37) A.R.T.|- at IP2 ^ is 0 --8θ--> true, $\bigwedge_{i=1}^{n}$ (Ai at 18 ^ bus_ai at 13 ^ bus_bi at 13 ^ bus_si at 11) ^ is 8θ

Ceci spécifie le comportement de IP2 pendant l'intervalle de temps [0,8θ[.

* L'intervalle de temps [8θ,9θ[:

Pendant ce dernier, les tâches A2,...,An se déplacent de 18 en 19 où elles restent en attente de rendez-vous, afin d'obtenir la retenue de l'étage-additionneur précédent. La tâche A1 quant à elle, entame l'exécution d'une affectation dont le temps total est de 4θ. Les autres processus conservent le même état qu'à la date 8θ. Soit formellement,

(38) ∀ i ∈ {1,...,n},

A.R.T.|- Ai at 18 --θ--> Ai at 18, Ai at 19 (par A-SEL1),

(39) A.R.T.|- A1 at 18 --4θ--> A1 at 18, A1 at 113 (par A-AFF),

(40) A.R.T.|- A1 at 18 --θ--> A1 at 18, A1 at 18

(par R-SIMPL et (39)),

(41) ∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at 13 --θ--> bus_ai at 13, bus_ai at 13

(il s'agit de l'assertion (16) prise en t'=θ)

(42) ∀ i ∈ {1,...,n},

A.R.T.|- bus_bi at 13 --θ--> bus_bi at 13, bus_bi at 13

(A-END appliquée à la tâche bus_bi).

Si nous composons les assertions (38), et (40)-(42), nous avons en tout

(43) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 18 ^ bus_ai at 13 ^ bus_bi at 13) --θ--> $\bigwedge_{i=1}^{n}$ (Ai at 18 ^ bus_ai at 13 ^ bus_bi at 13), (A1 at 18) ^ $\bigwedge_{i=2}^{n}$ (Ai at 19) ^ $\bigwedge_{i=1}^{n}$ (bus_ai at 13 ^ bus_bi at 13)

d'où par R-IS et R-IMPL,

(44) A.R.T.|- $\bigwedge_{i=1}^{n}$ (Ai at 18 ^ bus_ai at 13 ^ bus_bi at 13) ^ is 8θ--θ--> true, (A1 at 18) ^ $\bigwedge_{i=2}^{n}$ (Ai at 19) ^ $\bigwedge_{i=1}^{n}$ (bus_ai at 13 ^ bus_bi at 13) ^ is 9θ

* L'intervalle de temps [9θ,22θ[:

En vertu de l'assertion (16) on a

(46) ∀ i ∈ {1,...,n},

A.R.T.|- bus_ai at l3 --l3θ--> bus_ai at l3, bus_ai at l3

Grace à A-END, on a encore

(47) ∀ i ∈ {1,...,n},

A.R.T.|- bus_bi at l3 --l3θ--> bus_bi at l3, bus_bi at l3

comme A-SEL4 appliqué aux tâches A2,...,An conduit à

(48) ∀ i ∈ {2,...,n},

A.R.T.|- Ai at l9 --t(n-i)-9θ--> Ai at l9, Ai at ll0 ^ A(i-l) at ll9

Ce qui entraine qu'on a par R-SIMPL

(49) ∀ i ∈ {2,...,n},

A.R.T.|- Ai at l9 --l3θ-->Ai at l9, Ai at l9

d'où en composant (45), (46), (47) et (49) par R-COMP et en appliquant R-IMPL au résultat,

(50) A.R.T.|- $\bigwedge_{i=2}^{n}$ (Ai at l9) ^ $\bigwedge_{i=1}^{n}$ (bus_ai at l3 ^ bus_bi at l3) --l3θ--> true, $\bigwedge_{i=2}^{n}$ (Ai at l9) ^ $\bigwedge_{i=1}^{n}$(bus_ai at l3 ^ bus_bi at l3).

Donc nous possédons un invariant pour l'intervalle de temps considéré:

GIl = $\bigwedge_{i=2}^{n}$ (Ai at l9) ^ $\bigwedge_{i=1}^{n}$ (bus_ai at l3 ^ bus_bi at l3),

indiquant que

- les tâches (bus_ai)i∈{1,...,n}, et (bus_bi)i∈{1,...,n} sont terminées,
- les tâches (Ai)i∈{2,...,n} attendent leur retenue (cette dernière ne pouvant arriver avant la date à laquelle Al a transmis la sienne à la tâche A2).

Voyons comment se comportent les tâches Al et bus_sl.

(51) A.R.T. |- Al at l8 ^ is 8θ --4θ--> true, Al at ll3 ^ is l2θ

(par A-AFF, R-IS et R-SIMPL)

(52) A.R.T. |- Al at ll3 ^ is l2θ --θ--> true, Al at ll4 ^ is l3θ

(selon A-APl, R-IS et R-IMPL)

(53) A.R.T |- Al at l8 ^ is 8θ --5θ-->true, Al at ll4 ^ is l3θ

(par R-SEQ, (51) et (52)))

(53') A.R.T |- Al at ll ^ is θ --7θ--> true, Al at l8 ^ is 8θ

(par R-IMPL appliquée à (36))

(53") A.R.T |- Al at ll ^ is θ --l2θ--> true, Al at ll4 ^ is l3θ

(par R-SEQ, (53) et (53'))

(54) A.R.T |- Al at ll4 ^ bus_sl at ll --θ--> true, Al at ll5 ^ bus_sl at l2

(par A-AP2)

(55) A.R.T |- bus_sl at ll --l3θ--> bus_sl at ll, bus_sl at l2 ^ Al at ll5

(par (4) et R-SIMPL)

(56) A.R.T |- Al at ll ^ is θ ^ bus_sl at ll --l3θ--> true, Al at ll5 ^ bus_sl at l2 ^ is l4θ

(par R-SEQ, (53"), (54), R-COMP et (55))

(57) A.R.T |- Al at ll5 ^ bus_sl at l2 ^ is l4θ --θ--> true, Al at ll5 ^ bus_sl at l3 ^ is l5θ

(par A-AP5, A-AFF, R-IS, R-IMPL et R-COMP)

(58) A.R.T |- Al at ll5 ^ bus_sl at l3 ^ is l5θ --θ--> true, Al at ll6 ^ bus_sl at l4 ^ is l6θ

(par A-SEL8, A-AP4, R-IS, R-IMPL et R-COMP)

(59) ∀ t' ∈ IR+,

A.R.T.|- bus_sl at l4 --t'--> bus_sl at l4, bus_sl at l4

(par A-END)

(60) A.R.T |- Al at ll6 ^ bus_sl at l4 ^ is l6θ --5θ--> true, Al at ll7 ^ bus_sl at l4 ^ is 2lθ

(par A-AFF, (59), R-IS et R-IMPL)

(61) A.R.T |- Al at 117 ^ bus_sl at 14 ^ is 21θ --θ--> true, Al at 118 ^ bus_sl at 14 ^ is 22θ

(par A-AP1, (59), R-COMP, R-IS et R-IMPL)

(62) A.R.T |- Al at 11 ^ bus_sl at 11 --21θ--> true, Al at 118 ^ bus_sl at 14 ^ is 22θ

(par R-SEQ appliquée à (56), (57), (58), (60) et (61))

(63) A.R.T.|- at IP2 ^ is 0 --9θ--> /\(i=2 n) (Ai at 19) ^ /\(i=1 n) (bus_ai at 13 ^ bus_bi at 13) ^ is 9θ

(par R-SEQ, (37), (44) et R-IMPL)

(64) A.R.T.|- /\(i=2 n)(Ai at 19) ^ /\(i=1 n) (bus_ai at 13 ^ bus_bi at 13) ^ is 9θ --13θ--> true, /\(i=2 n)(Ai at 19) ^ /\(i=1 n) (bus_ai at 13 ^ bus_bi at 13) ^ is 22θ

(par R-IS appliquée à (50) et R-IMPL)

(65) A.R.T.|- at IP2 ^ is 0 --22θ--> true, is 22θ ^ /\(i=2 n)(Ai at 19) ^ /\(i=1 n) (bus_ai at 13 ^ bus_bi at 13)

(par R-SEQ appliquée à (63) et (64))

(66) A.R.T.|- Al at 10 ^ is 0 ^ bus_sl at 10 --22θ--> true, Al at 118 ^ bus_sl at 14 ^ is 22θ (par (0) et (62))

(67) ∀ i ∈ {2,...,n},

A.R.T.|- bus_si at 11 --21θ--> true, bus_si at 11

(par R-IMPL et (4))

(68) ∀ i ∈ {2,...,n},

A.R.T.|- bus_si at 10 --22θ--> true, bus_si at 11

(69) A.R.T.|- at IP2 ^ is 0 --22θ--> true, is 22θ ^ Al at 118 ^ /\(i=2 n)(Ai at 19 ^ bus_si at 11) ^ /\(i=1 n) (bus_ai at 13 ^ bus_bi at 13) ^ bus_sl at 14

(par R-COMP appliquée à (65), (66) et (68)).

C.Q.F.D.

V.E.iii.d Lemme 3:

A.R.T.|- P(n-2) --xl--> true, P0

Fin du lemme.

Nous allons prouver cette assertion à l'aide de la règle d'induction, en appliquant cette dernière aux assertions suivantes:

(**) ∀ k ∈ {1,...,n-2},

A.R.T.|- Pk --17θ--> true, P(k-1)

Il ne nous reste qu'à prouver (**), c'est à dire encore

A.R.T.|-(/\(i=1 n-k-2) Ai at 120) ^ (A(n-k-1) at 118) ^ (/\(i=n-k n) Ai at 19) ^ (/\(i=1 n-k-1) bus_si at 14) ^ (/\(i=n-k n) bus_si at 11) ^ AB ^ is tk --17θ--> true, (/\(i=1 n-k-1) Ai at 120) ^ (A(n-k) at 118) ^ (/\(i=n-k+1 n) Ai at 19) ^ (/\(i=1 n-k) bus_si at 14) (/\(i=n-k+1 n) bus_si at 11) ^ AB ^ is t(k-1)

Expliquons d'abord le comportement de IP2 entre les dates tk et t(k-1). Le (n-k)-ième étage-additionneur reçoit la retenue de l'étage précédent, effectue son addition, dont il envoit le résultat au bus de sortie, puis après avoir calculé sa retenue, se prépare à envoyer celle-ci à l'additionneur suivant. Ce cadencement intrinsèque de l'additionneur nous a conduit tout naturellement à découper l'axe des temps en tranches de 17θ seconde, entre les instants t=22θ et t=(17(n-1)+5)θ. En ce qui concerne les tâches Ai d'indice inférieur à n-k-2, elles se maintiennent dans le même état qu'à la date tk, ayant atteintes la fin de leurs instructions. Pour la même raison que celle invoquée dans le lemme précédent, les tâches A(n-k+1),...,An se trouvent en 19 tout au long de l'intervalle de temps [tk,t(k-1)]. Mentionnons le comportement des tâches simulant le bus des résultats: seule la tâche bus_s(n-k) se déplace car elle reçoit le résultat de A(n-k); les autres processus

restent dans le même état qu'auparavant, en l1 ou en l4.

Nous pouvons considérer l'invariant suivant, pour Ik=[tk,t(k-1)]:

$$GI2 = AB \wedge \bigwedge_{i=1}^{n-k-1} (bus\_si \text{ at } l4) \wedge \bigwedge_{i=1}^{n-k-2} (Ai \text{ at } l20) \wedge \bigwedge_{i=n-k+1}^{n} (bus\_si \text{ at } l1)$$

justifié par les assertions que voici:

∀ i ∈ {1,...,n}, ∀ t' ∈ IR+,

A.R.T.|- bus_ai at l3 --t'--> bus_ai at l3, bus_ai at l3

A.R.T.|- bus_bi at l3 --t'--> bus_bi at l3, bus_bi at l3

(selon A-END)

∀ i ∈ {1,...,n-k-1}, ∀ t' ∈ IR+,

A.R.T.|- bus_si at l4 --t'--> bus_si at l4, bus_si at l4

(selon A-END)

∀ i ∈ {1,...n-k-2}, ∀ t' ∈ IR+,

A.R.T.|- Ai at l20 --t'--> Ai at l20, Ai at l20

(selon A-END)

∀ i ∈ {n-k+1,...,n},

A.R.T.|- bus_si at l1 --t(n-i)+9θ--> bus_si at l1, bus_si at l2 ^ Ai at l15

(par l'assertion (4) du lemme précédent)

Il ne nous reste plus qu'à démontrer l'assertion suivante, extraite de (**):

(0) A.R.T.|- A(n-k-1) at l18 ^ A(n-k) at l9 ^ bus_s(n-k) at l1 ^ is tk --17θ--> true,

A(n-k-1) at l20 ^ A(n-k) at l18 ^ bus_s(n-k) at l4 ^ is t(k-1)

Tout d'abord nous remarquons que

A.R.T.|- bus_s(n-k) at l1 ^ is θ --tk+9θ--> bus_s(n-k) at l1, bus_s(n-k) at l2 ^ A(n-k) at l15

(par l'assertion (4) du lemme précédent et R-IS)

Donc J = (bus_s(n-k) at l1) est un invariant pour l'intervalle

[tk,tk+8θ].

(1) A.R.T.|- A(n-k) at l9 ^ A(n-k-1) at l18 ^ J ^ is tk --θ--> true, A(n-k) at l10 ^ A(n-k-1) at l19 ^ J ^ is tk+θ

(par A-SEL4, A-AP2, R-IS et R-IMPL)

(2) A.R.T.|- A(n-k) at l10 ^ A(n-k-1) at l19 ^ J ^ is tk+θ --θ--> true, A(n-k) at l11 ^ A(n-k-1) at l19 ^ J ^ is tk+2θ

(par A-AFF, A-AP5, R-IS et R-IMPL)

(3) A.R.T.|- A(n-k) at l11 ^ A(n-k-1) at l19 ^ J ^ is tk+2θ --θ--> true, A(n-k) at l12 ^ A(n-k-1) at l20 ^ J ^ is tk+3θ

(par A-AP4, A-SEL8, R-IS et R-IMPL)

(4) A.R.T.|- A(n-k) at l12 ^ A(n-k-1) at l20 ^ J ^ is tk+3θ --4θ--> true, A(n-k) at l13 ^ A(n-k-1) at l20 ^ J ^ is tk+7θ

(par A-AFF, A-END, R-IS et R-IMPL)

Posons I = (A(n-k-1) at l20). Selon A-END, I est invariant pour l'intervalle [tk+3θ,+infini[, donc à fortiori pour [tk+3θ,t(k-1)].

Nous composerons donc, à la fin de ce lemme par l'assertion:

(5) A(n-k-1) at l20 --14θ--> true, A(n-k-1) at l20

(obtenue par A-END, R-SIMPL et R-IMPL)

(6) A.R.T|- A(n-k) at l13 ^ J ^ is tk+7θ --θ--> true, A(n-k) at l14 ^ J ^ is tk+8θ

(par A-AP1, R-IS et R-IMPL)

(7) A.R.T|- A(n-k) at l14 ^ bus_s(n-k) at l1 ^ is 8θ --θ--> true, A(n-k) at l15 ^ bus_s(n-k) at l2 ^ is tk+9θ

(par A-SEL4 avec D=0, A-AP2, R-IS et R-IMPL)

(8) A.R.T|- A(n-k) at l15 ^ bus_s(n-k) at l2 ^ is tk+9θ --θ--> true, A(n-k) at l15 ^ bus_s(n-k) at l3 ^ is tk+10θ

(par A-AFF, A-AP5, R-IS et R-IMPL)

(9) A.R.T.|- A(n-k) at l15 ^ bus_s(n-k) at l3 ^ is tk+10θ --θ-->

true, A(n-k) at 116 ^ bus_s(n-k) at 14 ^ is tk+110

(par A-SEL8, A-AP4, R-IS et R-IMPL)

Grace à A-END, nous pouvons dire que l'assertion K = (bus_s(n-k) at 14) est invariante pour [tk+110,tk+170].

Nous composerons à la fin de ce lemme l'assertion finale par

(10) A.R.T.|- bus_s(n-k) at 14 --60--> bus_s(n-k) at 14, bus_s(n-k) at 14

(11) A.R.T.|- A(n-k) at 116 ^ is tk+110 --50--> true, A(n-k) at 117 ^ is tk+160

(par A-AFF, R-IS et R-IMPL)

(12) A.R.T.|- A(n-k) at 117 ^ is tk+160 --0--> true, A(n-k) at 118 ^ is tk+170

(par A-AP1, R-IS et R-IMPL)

(13) A.R.T.|- A(n-k) at 116 ^ bus_s(n-k) at 14 ^ is tk+110 --60--> true, A(n-k) at 118 ^ bus_s(n-k) at 14 ^ is tk+170

(par R-SEQ, (11), (12), R-COMP et (10))

(14) A.R.T.|- A(n-k) at 113 ^ bus_s(n-k) at 11 ^ is tk+70 --100--> true, A(n-k) at 118 ^ bus_s(n-k) at 14 ^ is tk+170

(par R-SEQ, (6)-(9), (13) et (14))

(15) A.R.T.|- A(n-k) at 113 ^ A(n-k-1) at 120 ^ bus_s(n-k) at 11 ^ is tk+70 --100--> true, A(n-k) at 118 ^ A(n-k-1) at 120 ^ bus_s(n-k) at 14 ^ tk+170

(16) A.R.T.|- A(n-k) at 19 ^ A(n-k-1) at 118 ^ bus_s(n-k) at 11 ^ is tk --170--> true, A(n-k) at 118 ^ A(n-k-1) at 120 ^ bus_s(n-k) at 14 ^ is t(k-1)

(par R-SEQ, (1)-(4), (15) et R-IMPL)

C'est à dire l'assertion (0).

Finalement, à l'aide de (16) et de GI2, nous obtenons (**), et par

R-INDUCTION appliquée à cette dernière, nous avons démontré le résultat désiré.

C.Q.F.D.

V.E.iii.e Lemme 4:

A.R.T.|- P0 --110--> true, FINISH

Fin du lemme.

Démonstration: Il nous faut prouver la propriété:

$$\text{A.R.T.}\vdash \left(\bigwedge_{i=1}^{n-2} Ai \text{ at } 120\right) \wedge (A(n-1) \text{ at } 118) \wedge (An \text{ at } 19) \wedge \left(\bigwedge_{i<n} bus\_si \text{ at } 14\right) \wedge (bus\_sn \text{ at } 11) \wedge AB \wedge \text{is } (17n-2)\theta \text{ --110--> true,}$$

$$\left(\bigwedge_{i=1}^{n-1} Ai \text{ at } 120\right) \wedge (An \text{ at } 116) \wedge \text{is } (17n-1)\theta$$

Dans cette étape du calcul l'additionneur effectue les opérations suivantes:

- La tâche A(n-1) communique sa retenue à l'étage suivant et termine ainsi son travail.
- An effectue son addition et dépose son résultat sur le n-ième fil du bus résultat.
- Les autres tâches constituant IP2 ont un comportement inchangé.

En vertu des quatre assertions suivantes, nous pouvons distinguer l'invariant GI3 pour l'intervalle de temps [t0,t0+110]:

$$GI3 = AB \wedge \left(\bigwedge_{i=1}^{n-1} bus\_si \text{ at } 14\right) \wedge \left(\bigwedge_{i=1}^{n-2} Ai \text{ at } 120\right)$$

(A) $\forall i \in \{1,\ldots,n\}, \forall t' \in IR+,$

A.R.T.|- bus_ai at 13 --t'--> bus_ai at 13, bus_ai at 13

(B) A.R.T.|- bus_bi at 13 --t'--> bus_bi at 13, bus_bi at 13

(selon A-END)

(C) $\forall i \in \{1,\ldots,n-1\}, \forall t' \in IR+,$

A.R.T.|- bus_si at 14 --t'--> bus_si at 14, bus_si at 14

(selon A-END)

(D) ∀ i ∈ {1,...,n-2}, ∀ t' ∈ IR+,

A.R.T.|- Ai at 120 --t'--> Ai at 120, Ai at 120

(selon A-END)

Il nous reste ainsi à prouver que

(0) A.R.T.|- A(n-1) at 118 ^ An at 19 ^ bus_sn at 11 ^ is t0 --11θ--> true, A(n-1) at 120 ^ An at 116 ^ bus_sn at 14 ^ is (17n-1)θ

Pour celà, découpons l'intervalle de temps [t0,t0+11θ] en huit sous-intervalles, à chacun desquels nous associons une assertion A.R.T. Voici la preuve de (0):

(1) A.R.T.|- A(n-1) at 118 ^ An at 19 ^ is t0 --θ--> A(n-1) at 118 ^ An at 19, A(n-1) at 119 ^ An at 110 ^ is t0+θ

(par A-SEL4 où D=0, A-AP2, R-IS, R-IMPL et R-COMP)

(2) A.R.T.|- A(n-1) at 119 ^ An at 110 ^ is t0+θ --θ--> true, A(n-1) at 119 ^ An at 111 ^ is t0+2θ

(par A-AFF, A-AP5, R-IS, R-IMPL et R-COMP)

(3) A.R.T.|- A(n-1) at 119 ^ An at 111 ^ is t0+2θ --θ--> true, A(n-1) at 120 ^ An at 112 ^ is t0+3θ

(par A-AP4, A-SEL8, R-IS, R-IMPL et R-COMP)

(4) A.R.T.|- An at 112 ^ is t0+3θ --4θ--> true, An at 113 ^ is t0+7θ

(par A-AFF, R-IS et R-IMPL)

(5) A.R.T.|- An at 113 ^ is t0+7θ --θ--> true, An at 114 ^ is t0+8θ

(par A-AP1, R-IS et R-IMPL)

(6) A.R.T.|- An at 114 ^ bus_sn at 11 ^ is t0+8θ --θ--> true, An at 115 ^ bus_sn at 12 ^ is t0+9θ

(par A-AP2, A-SEL4 avec D=0)

(7) A.R.T.|- An at 115 ^ bus_sn at 12 ^ is t0+9θ --θ--> true, An at 115 ^ bus_sn at 13 ^ is t0+10θ

(par A-AFF, A-AP5, R-IS, R-IMPL et R-COMP)

(8) A.R.T.|- An at 115 ^ bus_sn at 13 ^ is t0+10θ --θ--> true, An at 116 ^ bus_sn at 14 ^ is t0+11θ

(par A-SEL8, A-AP4, R-IS et R-IMPL)

(9) A.R.T.|- A(n-1) at 118 ^ An at 19 ^ is t0 --3θ--> true, A(n-1) at 120 ^ An at 112 ^ is t0+3θ

(par R-SEQ, et (1)-(3))

(10) A.R.T.|- bus_sn at 11 ^ is θ --t0+8θ--> true, bus_sn at 12 ^ is t0+9θ

(par l'assertion (4) du lemme précédent, R-IS et R-IMPL)

Selon (10), l'assertion "bus_sn at 11" est invariante entre les dates t=θ et t=t0+8θ. Donc, selon (9) on a

(11) A.R.T.|- A(n-1) at 118 ^ An at 19 ^ bus_sn at 11 ^ is t0 --3θ--> true, A(n-1) at 120 ^ An at 112 ^ bus_sn at 11 ^ is t0+3θ

(12) ∀ t' ∈ IR+,

A.R.T.|- A(n-1) at 120 --t'--> true, A(n-1) at 120

(par A-END et R-IMPL)

(13) A.R.T.|- A(n-1) at 120 ^ An at 112 ^ bus_sn at 11 ^ is t0+3θ --5θ--> true, A(n-1) at 120 ^ An at 14 ^ is t0+8θ ^ bus_sn at 11

(par (4), (5), R-SEQ, (10), (12) et R-COMP)

(14) A.R.T.|- An at 114 ^ is t0+8θ ^ bus_sn at 11 --3θ--> true, An at 116 ^ bus_sn at 14 ^ is t0+11θ

(par R-SEQ, (6), (7) et (8))

(15) A.R.T.|- A(n-1) at 120 ^ An at 112 ^ bus_sn at 11 ^ is t0+3θ --8θ--> true, An at 116 ^ bus_sn at 14 ^ is t0+11θ

(par (13), R-IMPL, (14) et R-SEQ)

(16) A.R.T.|- A(n-1) at 120 ^ An at 112 ^ bus_sn at 11 ^ is t0+3θ --8θ--> true, An at 116 ^ A(n-1) at 120 ^ bus_sn at 14 ^ is t0+11θ

(par (12), (15) et R-COMP)

(17) A.R.T.|- A(n-1) at 118 ^ An at 19 ^ bus_sn at 11 ^ is t0 --11θ--> true, An at 116 ^ A(n-1) at 120 ^ bus_sn at 14 ^ is t0+11θ

(par (11), (16) et R-SEQ)

Comme t0+11θ = (17(n-1)+5)θ+11θ = (17n-1)θ, l'assertion (17) équivaut à (0). Il suffit de composer (0) avec les quatre assertions (A) à (D), par R-COMP, afin d'obtenir le résultat désiré au début de ce lemme.

C.Q.F.D.

Ainsi s'achève la preuve de la propriété forte, relative à l'additionneur.

Fin de la preuve de la propriété de fatalité temps-réel.

CHAPITRE VI

CONCLUSION.

Des éléments de conclusion ayant été donnés dans chaque chapitre, nous discutons maintenant quelques axes de travail.

Notre étude des concepts temps-réel du langage ADA nous a permis de donner une première formalisation de ceux-ci, très fortement basée sur une mesure du temps. La complexité de celle-ci est due au mélange des outils de manipulation du temps et de communication synchrone entre tâches en un nombre réduit de primitives; l'instruction "select" côté appelée en constitue le meilleur exemple car elle regroupe en une seule construction trois utilisations distinctes.

A partir d'une sémantique informelle et officielle de ADA, nous avons élaboré une sémantique opérationnelle respectant l'hypothèse de parallélisme maximal et imprégnée d'un modèle exécutif à parallélisme véritable. Chaque instruction élémentaire de notre langage a été découpée en un ensemble d'actions atomiques dont l'enchaînement contitue son exécution, et auxquelles nous avons adjoint, de manière purement syntaxique, un réel qui est leur temps d'exécution. En projetant sur un même axe temporel les chronologies associées aux différentes taches d'un programme il est possible, en un état donné, de décider de l'ensemble des actions atomiques $\mathcal{C}[s]$ à exécuter dans la prochaine transition ainsi que la durée de cette

dernière.

Cette première sémantique, aisément extensible au langage ADA tout entier, nous a guidé vers une méthode et un système de preuve de propriét'es de fatalité temps-réel, basés sur la logique temporelle classique. Dans la même lignée que l'approche considérée par PNUELI et DE ROEVER dans [51], nous avons rajouté à cette logique une dimension temporelle, selon laquelle sont dirigées toutes les preuves, ainsi que nous l'avons constaté au chapitre précédent. L'opérateur "until-temporisé" ainsi que l'assertion de datation autorisent la preuve d'une propriété de fatalité en exprimant ce qui se produit à tout instant dans le programme, avec un degré d'observation suffisament fin.

Dans les exemples tels ceux du chapitre V, nous avons toutefois remarqué une certaine lourdeur d'utilisation de celui-ci. Il nous faut donc dégager une meilleure méthodologie afin de prouver des propriétés temps-réel, dans laquelle le temps semble tenir un rôle prépondérant. Il nous semble également intéressant d'assurer une plus grande modularité dans ces preuves ainsi qu'une meilleure vue d'ensemble de ces processus; les treillis de preuve semblent être un cadre adéquat.

Nous nous acheminerons ainsi vers la résolution du second problème, à savoir la complétude du système axiomatique A.R.T.

Nos preuves ont été longues et difficiles; il convient donc de construire un outil d'aide à la preuve, semi-automatique tout au moins. Les premiers pas réalisés par MOSKOWSKI (cf. [49]) par exemple, nous suggèrent une approche par le calcul symbolique.

En dernier lieu, il faut s'assurer de l'utilité d'A.R.T. en l'appliquant à un plus grand nombre d'exemples de programmes.

GRAMMAIRE DU SOUS-ENSEMBLE DE ADA ETUDIE

APPENDICE A

GRAMMAIRE DU SOUS-ENSEMBLE DE ADA ETUDIE

```
⟨program⟩                    ::= ⟨sequence of tasks⟩

⟨sequence of tasks⟩          ::= ⟨task⟩
                               | ⟨sequence of tasks⟩ ⟨task⟩

⟨task⟩                       ::= task ⟨identifier⟩ is ⟨declarative part⟩
                                 begin ⟨sequence of statements⟩ end ;

⟨declarative part⟩           ::= NIL
                               | ⟨list of declarations⟩

⟨list of declarations⟩       ::= ⟨identifier list⟩ : ⟨type⟩ ;
                               | ⟨identifier list⟩ : ⟨type⟩ ; ⟨list of declarations⟩

⟨identifier list⟩            ::= ⟨identifier⟩
                               | ⟨identifier⟩ , ⟨identifier list⟩

⟨identifier⟩                 ::= ⟨letter⟩ ⟨list of letter or digits⟩

⟨list of letter or digits⟩   ::= ⟨letter⟩
                               | ⟨digit⟩
                               | ⟨letter⟩ ⟨list of letter or digits⟩
                               | ⟨digit⟩ ⟨list of letter or digits⟩

⟨type⟩                       ::= integer
                               | float

⟨sequence of staements⟩      ::= ⟨statement⟩
                               | ⟨sequence of statements⟩ ⟨statement⟩

⟨statement⟩                  ::= ⟨label⟩ ⟨simple statement⟩
                               | ⟨simple statement⟩
                               | ⟨label⟩ ⟨compound statement⟩
                               | ⟨compound statement⟩

⟨label⟩                      ::= << ⟨identifier⟩ >>
```

```
⟨simple statement⟩        ::= ⟨assignment statement⟩
                            | ⟨delay statement⟩
                            | ⟨entry call⟩
                            | ⟨null statement⟩

⟨compound statement⟩      ::= ⟨if statement⟩
                            | ⟨loop statement⟩
                            | ⟨select statement⟩

⟨assignment statement⟩    ::= ⟨identifier⟩ := ⟨expression⟩ ;

⟨delay statement⟩         ::= delay ⟨real⟩ ;

⟨entry call⟩              ::= ⟨identifier⟩ . ⟨identifier⟩  ( ⟨input parameters⟩ ,
                              ⟨output parameters⟩ ) ;
                            | ⟨identifier⟩ . ⟨identifier⟩ ;

⟨input parameters⟩        ::= ⟨expression list⟩
                            | NIL

⟨output parameters⟩       ::= ⟨identifier list⟩
                            | NIL

⟨null statement⟩          ::= null ;

⟨if statement⟩            ::= if ⟨condition⟩ then ⟨sequence of statements⟩ end if;

⟨loop statement⟩          ::= while ⟨condition⟩ loop
                              ⟨sequence of statements⟩ end loop;

⟨select statement⟩        ::= ⟨selective wait⟩
                            | ⟨timed entry call⟩

⟨selective wait⟩          ::= select
                              ⟨list of branchs⟩
                              end select ;

⟨list of branchs⟩         ::= ⟨branch⟩ or ⟨list of branchs⟩
                            | ⟨branch⟩

⟨branch⟩                  ::= when ⟨condition⟩ => ⟨select alternative⟩

⟨select alternative⟩      ::= ⟨accept statement⟩ ⟨sequence of statements⟩
                            | ⟨delay statement⟩ ⟨sequence of statements⟩
                            | ⟨accept statement⟩
                            | ⟨delay statement⟩

⟨accept statement⟩        ::= accept ⟨identifier⟩ ⟨formal part⟩ do
                              ⟨sequence of statements⟩ end ⟨identifier⟩ ;

⟨formal part⟩             ::= ( ⟨formal input parameters⟩ ;
                                ⟨formal output parameters⟩ )
                            | ( ⟨formal input parameters⟩ )
                            | ( ⟨formal output parameters⟩ )
                            | ()
```



```
⟨formal input parameters⟩::= ⟨identifier list⟩ : in ⟨type⟩
                           | ⟨identifier list⟩ : in ⟨type⟩ ;
                             ⟨formal input parameters⟩

⟨formal output parameters⟩::= ⟨identifier list⟩ : out ⟨type⟩
                            | ⟨identifier list⟩ : out ⟨type⟩ ;
                              ⟨formal input parameters⟩

⟨timed entry call⟩        ::= select
                              ⟨entry call⟩ ⟨sequence of statements⟩
                              or
                              ⟨sequence of statements⟩
                              end select ;
                            | select
                              ⟨entry call⟩
                              or
                              ⟨sequence of statements⟩
                              end select ;

⟨condition⟩               ::= ⟨relation⟩ ⟨list of and relations⟩
                            | ⟨relation⟩ ⟨list of or relations⟩

⟨list of and relations⟩   ::= NIL
                            | and ⟨relation⟩ ⟨list of and relations⟩

⟨list of or relations⟩    ::= NIL
                            | or ⟨relation⟩ ⟨list of or relations⟩

⟨relation⟩                ::= ⟨expression⟩
                            | ⟨expression⟩ ⟨relational operator⟩  ⟨expression⟩

⟨expression list⟩         ::= ⟨expression⟩
                            | ⟨expression⟩ , ⟨expression list⟩

⟨expression⟩              ::= ⟨unary operator⟩ ⟨simple expression⟩
                            | ⟨simple expression⟩

⟨simple expression⟩       ::= ⟨term⟩ ⟨list of added terms⟩
                            | ⟨term⟩

⟨list of added terms⟩     ::= ⟨adding operator⟩ ⟨term⟩ ⟨list of added terms⟩

⟨term⟩                    ::= ⟨factor⟩ ⟨list of multiplyed factors⟩
                            | ⟨factor⟩

⟨list of multiplyed factors⟩ ::= ⟨multiplying operator⟩ ⟨factor⟩
                                 ⟨list of multiplied factors⟩

⟨factor⟩                  ::= ⟨primary⟩
                            | ⟨primary⟩ ** ⟨primary⟩

⟨primary⟩                 ::= ⟨literal⟩
                            | ⟨identifier⟩
                            | (⟨condition⟩)
```

⟨logical operator⟩ ::= and | or

⟨relational operator⟩ ::= = | /= | < | <= | > | >=

⟨unary operator⟩ ::= - | + | not

⟨adding operator⟩ ::= + | -

⟨multiplying operator⟩ ::= * | / | mod | rem

REFERENCES BIBLIOGRAPHIQUES

APPENDICE B

REFERENCES BIBLIOGRAPHIQUES

[1] M. ABADI, Z. MANNA
"Nonclausal temporal Deduction"
Dept. of computer Science, Stanford University, Report no. STAN-CS-85-1056, June 1986.

[2] K.R. APT
"A static analysis of CSP programs"
Workshop on logics of programs, Pittsburgh, 6-8 June 1983, Springer Verlag.

[3] K.R. APT, P. CLERMONT
"Two normal forms theorems for CSP programs"
I.B.M. Watson Research Center, Report no. RC-10975, July 1985.

[4] K.R. APT, C. DELPORTE
"Syntax directed analysis of liveness properties"
LITP, Université PARIS 7, Rapport no. 83-42, Octobre 1983.

[5] K.R. APT, N. FRANCEZ, W.P. DE-ROEVER
"A proof system for C.S.P."
A.C.M. Transactions on Programming Languages and Systems, Vol.2, no.3, July 1980, pp.359-385.

[6] K.R. APT, E.R. OLDEROG
"Proof rules and transformations dealing with fairness"
L.N.C.S. no. 131, "Logics of programs", Springer Verlag 1982.

[7] P. BELLOT, A. PUCCETTI
"Conception et réalisation de programmes de Calcul Numérique en Langage ADA: le système CANADA"
Proceedings of the International Conference "Future trends in Computing", Grenoble, 2-6 Decembre 1985.

[8] A. BERNSTEIN, P.K. HARTER Jr.
"Proving Real-time properties of programs with temporal logic"
C.A.C.M., pp.1-81, 1981.

[9] G. BERRY, L. COSSERAT
"The Esterel Synchronous Programming Language and its Mathematical Semantics",
Rapport I.N.R.I.A. no.327, Septembre 1984.

[10] S.D. BROOKES
"On the axiomatic treatment of concurrency"
Proc. 1984, NSF-SERC, Seminar on Concurrency, L.N.C.S. no.164, "Logic of programs", Springer Verlag, 1983.

[11] S.D. BROOKES
"A semantics and proof system for communicating processes"
L.N.C.S. no. 197, pp.68-85.

[12] R.S. BOYER, J. STROTHER-MOORE
"Proof checking, theorem proving and program verification"
Technical report no. 35, University of TEXAS at Austin, January 1983.

[13] O.S.F. CARVALHO, G. ROUCAIROL
"Assertion decomposition and partial correctness of distibuted control algorithms"
Institut de programmation, Université PARIS 6.

[14] P. CASPI, N. HALBWACHS
"A functional model for describing and reasoning about time behaviour of computing systems",
Acta Informatica.

[15] S.A. COOK
"Soudness and completeness of an axiom system for program verification",
Technical report, Department of Computer Science, Univerity of TORONTO, march 31, 1977.

[16] P. COUSOT
"Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes",
Thèse d'état, U.S.M.G.-I.N.P.G., GRENOBLE, mars 1978.

[17] P. COUSOT, R. COUSOT
"Invariance proof methods and analysis techniques for parallel programs",
Rapport de recherche C.R.I.N., NANCY, no. 82-P067, mai 1982.

[18] P. COUSOT, R. COUSOT
"On the soundness and completeness of generalized Hoare-logic",
Rapport de recherche no.82-P093, C.R.I.N. NANCY, decembre 1982.

[19] U.S. DEPARTMENT OF DEFENSE
"Ada Reference Manual",
MIL-STD-1815, 10 december 1980.

[20] T. DESPEYROUX
"Executable specification of static semantics",
Rapport de recherche I.N.R.I.A., no. 295, mai 1984.

[21] D.L. DILL, E.M. CLARKE
"Automatic verification of asynchronous circuits using temporal logic",
Dept. of Computer Science, Report no. CMU-CS-85-125, CARNEGIE-MELLON University.

[22] H.R. GAREY, D.S. JOHNSON
"Computers and intractability, a guide to the theory of NP-completeness",
W.H.FREEMAN and Co., SAN FRANCISCO, Victor KLEE editor.

[23] R. GERTH
"A sound and complete Hoare-axiomatization of the ADA rendez-vous",
Report no. RUU-CS-82-5, University of UTRECHT, april 1982.

[24] R. GERTH, W.P. DE-ROEVER
"A proof system for concurrent ADA programs",
Science of Computer programming 4(1984), pp. 159-204, North Holland.

[25] D.I. GOOD
"Mechanical proofs about computer programs",
Technical Report no. 41, University of TEXAS at AUSTIN, march 1, 1984.

[26] D. GRUMBERG, N. FRANCEZ, S. KATZ
"A complete proof rule for strong equifair termination",
L.N.C.S. no. 164, "Logic of programs", pp. 257-278.

[27] N. HALBWACHS
"Analyse statique de propriétés linéaires des systèmes indéterministes et parallèles",
Rapport de recherche no. 367, Laboratoire I.M.A.G., Université de GRENOBLE, avril 1983.

[28] N. HALBWACHS
"Modélisation et analyse du comportement des systèmes informatiques temporisés",
Thèse d'état, Université de GRENOBLE, 1984.

[29] J. HALPERN, Z. MANNA, B. MOSZKOWSKI
"A hardware semantics based on temporal intervals",
Proceedings of the 10th International Colloquium on Automata, Languages and Programming, BARCELONA, Spain, july 1983.

[30] D. HAREL, D. KOZEN, R. PARIKH
"Process logic: expressiveness, decidability, completeness",
Proc. of the 21st symposium on foundations of Computer Science, SYRACUSE, N.Y., october 1980.

[31] C.A.R. HOARE
"Communicating sequential processes",
C.A.C.M. no. 21, volume 8, august 1978, pp. 666-677.

[32] D.E. KNUTH
"The art of computer programming, Vol 2: seminumerical algorithms",
Addison Wesley Pub. Co.

[33] R. KOYMANS, R.K. SHYAMASUNDAR, W.P. DE-ROEVER, R.GERTH, S. ARUN-KUMAR
"Compositional semantics for real-time distributed computing",
L.N.C.S. no. 193, "Logics of programs", BROOKLYN, june 1985.

[34] R. KUIPER, W.P. DE-ROEVER
"Fairness assumptions for C.S.P. in a temporal logic framework",
TC2 working conference on the formal desciption of programming concepts, GARMISCH-PARTENKIRCHEN, june 1982.

[35] L. LAMPORT, F.B. SCHNEIDER
"The Hoare-logic of C.S.P. and all that",
Dept. of Computer Science, Cornell University, Report no. TR-82-490, may 1982.

[36] L. LAMPORT
"Time, clocks and the ordering of events in a distributed system",
C.A.C.M. volume 21, no. 7, pp. 558-565, july 1978.

[37] G.M. LEVIN, D. GRIES
"A proof technique for C.S.P.", Acta informatica 15, pp. 281-302, Springer Verlag, 1981.

[38] W.A. MADDEN, R.Y. RONE
"Design, development, integration: space shuttle primary flight software system",
C.A.C.M. 27, 9, pp. 914-925, september 1984.

[39] Z. MANNA, A. PNUELI
"Verification of concurrent programs: temporal proof principles",
Dept. of Computer Science, University of STANFORD, Report no. STAN-CS-81-843.

[40] Z. MANNA, A. PNUELI
"Verification of concurrent programs: proving eventualities by well-founded ranking",
Dept. of Computer Science, STANFORD University, Report no. STAN-CS-82-15, may 1982.

[41] Z. MANNA, A. PNUELI
"Proving precedence properties: the temporal way",
Dept. of Computer Science, STANFORD University, Report no. STAN-CS-83-964, april 1983.

[42] Z. MANNA, A. PNUELI
"Verification of concurrent programs: a temporal proof system",
Dept. of Computer Science, STANFORD University, Report no.

STAN-CS-83-967, june 1983.

[43] Z. MANNA, A. PNUELI
"Adequate proof principles for invariance and liveness properties of concurrent programs",
Science of Computer programming 4(1984), pp. 257-289, North Holland.

[44] D. MERY
"Une méthode axiomatique de preuves de propriétés de fatalité de programmes parallèles avec hypothèse d'éxécution équitable",
Thèse de 3ème cycle, I.N.P.L., NANCY, mai 1983.

[45] D. MERY
"Une méthode axiomatique pour les propriétés de fatalité des programmes parallèles",
A paraitre dans RAIRO.

[46] D. MERY, A. PUCCETTI
"An operational approach for proving real-time ADA programs",
Proceedings of the sixth conference of the Chilean Computer Science society, july 28-30, 1986.

[47] "Real-time data handling and process control",
H. MEYER ed., North Holland Pub. Co. ECSC, EEC, EAEC, BRUSSELS and LUXEMBOURG, pp.25-29, 209-241, 751-754, 1980.

[48] B. MOSZKOWSKI
"A temporal logic for multi-level reasoning about hardware",
Computer Hardware Description Languages and their applications, T. UEHARA and M. BABACCI (editors), North Holland Pub. Co., I.F.I.P. 1983.

[49] B. MOSZKOWSKI
"Executing temporal logic programs",
Technical report no. 55, University of CAMBRIDGE, Computer Science Laboratory.

[50] B. MOSZKOWSKI, Z. MANNA
"Reasoning in interval temporal logic",
L.N.C.S. no. 164, "Logic of programs", pp. 371-382, 1983.

[51] A. PNUELI, W.P. DE-ROEVER
"Rendez-vous with ADA - a proof theretical view",
Rijksuniversiteit UTRECHT, Vakgroep Informatica, Report RUU-CS-82-12, july 1982.

[52] A. PUCCETTI
"Un compilataur de langage parallèle: APASCAL",
Rapport de recherche no. LRIM-84-03, Université de METZ, octobre 1984.

[53] H. ROGERS Jr.
"Theory of recursive functions and effective computability", Mc. GRAW HILL Book Co.

[54] W.C. ROUNDS
"Applications of topology to semantics of communicating sequential processes",
University of MICHIGAN, Computer Research Laboratory, Report no. CRL-TR-32-84, july 1984.

[55] R.L. SCHWARTZ, P.M. MELLIAR-SMITH, F.H. VOGT
"An interval-based temporal logic",
L.N.C.S. no. 164, "Logic of programs", pp.443-457.

[56] T. SHIMIZU, K. SAKAMURA
"Automatic tuning of multi-tasks program for real-time embedded systems",
Dept. of Computer Science, Faculty of Sciences, University of TOKYO, Report no. 85-11, may 1985.

[57] A. SPECTOR, D. GIFFORD
"The space-shuttle primary computer system",
C.A.C.M. Vol. 27,no. 9, september 1984 pp. 873-900.

[58] J.E. STOY
"Denotational semantics: the Scott-Strachey approach to programming language theory",
M.I.T. Press series in Computer Science.

[59] B.A. TRAKHENBROT, J.Y. HALPERN, A.R. MEYER
"From denotational to operational and axiomatic semantics for Algol-like languages: an overview",
Laboratory of Computer Science, M.I.T., Report no. MIT/LCS/TM-246, october 1983.

[60] D. LE VERRAND
"Le langage ADA, Manuel d'évaluation",
Dunod Informatique.

[61] P. WOLPER
"Temporal logic can be more expressive",
Dept. of Computer Science, University of STANFORD.