Institut National Polytechnique
de Lorraine

Centre de Recherche en
Informatique de Nancy

ETUDE ET REALISATION DE METHODES DE PREUVE PAR RECURRENCE EN LOGIQUE EQUATIONNELLE

THESE DE DOCTORAT
DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE
EN INFORMATIQUE

Présentée par
Azzeddine LAZREK

Soutenue le 30 novembre 1988
Devant la commission d'examen

Président :	J.P. JOUANNAUD
Rapporteurs :	M. DAUCHET
	J.P. FINANCE
Examinateurs :	D. COULON
	P. LESCANNE
	J.L. REMY

Institut National Polytechnique
de Lorraine

Centre de Recherche en
Informatique de Nancy

ETUDE ET REALISATION DE METHODES DE PREUVE PAR RECURRENCE EN LOGIQUE EQUATIONNELLE

THESE DE DOCTORAT
DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE
EN INFORMATIQUE

Présentée par

Azzeddine LAZREK

Soutenue le 30 novembre 1988
Devant la commission d'examen

Président :	J.P. JOUANNAUD
Rapporteurs :	M. DAUCHET
	J.P. FINANCE
Examinateurs :	D. COULON
	P. LESCANNE
	J.L. REMY

RESUME

L'objet de cette thèse est l'étude pratique de preuve par récurrence en logique équationnelle. Elle fournit ainsi un outil puissant pour la programmation logique et/ou fonctionnelle et pour la démonstration automatique.

Notre objectif principal est l'étude et la mise en œuvre d'une méthode de preuve de théorèmes inductifs dans l'algèbre initiale d'une variété équationnelle. La méthode étudiée est la synthèse de la méthode de preuve par consistance et de celle basée sur la réductibilité inductive.

Le deuxième objectif est l'étude de la complétude relative, souhaitable dans la conception et la correction des spécifications algébriques structurées et requise par la méthode de preuve par consistance.

Une partie de ce travail est consacrée à la présentation de la ω-complétude: elle constitue, à notre connaissance, la première tentative d'étude systématique de cette propriété.

Par ailleurs nous donnons une méthode de reconnaissance et de génération des formes normales closes d'un système de réécriture.

Enfin il est à noter que toutes ces méthodes de preuves de validité, de complétude relative ainsi que la reconnaissance et la génération des formes normales closes d'un système de réécriture ont été implantées dans le démonstrateur automatique REVE.

Mots-clés: algèbre initiale, spécification, consistance relative, complétude relative, ω-complétude, réécriture, procédure de complétion, récurrence, automate, grammaire.

Study and Implementation of Inductive Proof Methods in Equational Logic

ABSTRACT

The object of this thesis is the practical elaboration of the inductive proof in equational logic. It gives a strong tool for logic and functional programming and for theorem proving.

The first objectif is the study and implementation of a proving inductive theorems in the initial algebra of equational varety method. The method adopted is the method based on proof by consistency and this based on inductive reducibility synthesis.

The second objectif is the investigation of relative completeness propertie, desirable in structural algebrique specification conception and correction and exacting in the method based in proof by consistency.

A part of this work is consecrated to a systematic study of ω-completeness propertie useful in axiomatisation of theories.

Otherwise a method based on trees automata and grammar and on rewriting is explained for recognition and generation of closed normal formes with rapport to a term rewriting system.

Jot down that all these methods of proving theorems, relative completeness, recognition and generation of closed normal formes with rapport to a term rewriting system are implemented in the rewriting laboratory REVE.

Keywords: initial algebra, specification, relative consistency, relative completeness, ω-completeness, term rewriting, completion procedure, induction, automata, grammar.

A ma mère

a mon père

a ma famille

a mes ami(e)s.

Je tiens à remercier vivement tous les membres du jury:

Daniel Coulon, Professeur à l'INPL, et je lui exprime ma vive reconnaissance.

Max Dauchet, Professeur à l'université de Lille I, venu de Lille pour siéger dans ce jury en tant que rapporteur de ma thèse.

Jean-Pierre Finance, Professeur à l'université de Nancy I, qui a accépté d'être rapporteur de ma thèse. Et je le remercie de ses critiques positives.

Jean-Pierre Jouannaud, Professeur à l'université de Paris-sud, qui m'a fait l'honneur de présider ce jury et de ses remarques pertinentes.

Pierre Lescanne, directeur de recherche au CNRS, qui a bien su encadrer et diriger mon travail de recherche depuis le DEA. J'ai énormément bénéficié de ses précieux conseils et de ses encouragements qui ont marqué profondément cette thèse et m'ont permis de la mener à bien. Qu'il trouve ici l'expression de ma profonde gratitude.

Jean-luc Rémy, chargé de recherche au CNRS, à qui je suis particulièrement reconnaissant pour le grand intérêt qu'il a toujours porté à mes travaux et pour les nombreux encouragements qu'il m'a prodigués.

Je remercie aussi les enseignants de la faculté des sciences de Marrakech de toutes les connaissances de base qu'ils ont voulu bien me prodiguer, sans oublier le personnel qui m'a bien faciliter toutes choses.

Cette thèse a été réalisée au sein de l'équipe EURECA du CRIN et je tiens à remercier tous ceux qui de près ou de loin ont facilité ce travail. Je remercier également tous les membres de l'équipe en particulier Françoise Bellegarde, Miki Hermann, Claude Kirchner et Hélène Kirchner pour leur amical soutien. Mes remerciments s'adressent à Mohamed Adi, Abderrafia Koukam, Noureddine Lazrak, Mohamed Tajine ainsi que tous les membres du laboratoire CRIN pour leur amical appui et la création d'une agréable ambiance de travail.

En fin que tous mes proches trouvent ici le témoignage de ma plus sincère gratitude et sentiment pour leur soutien et appui ainsi que leurs encouragements de tous moments.

Table des matières

INTRODUCTION

Le souci d'une conception, de grands logiciels par équipe, plus correcte, moins coûteuse et plus rapide, a amené à des recherches fondamentales sur les types abstraits de données dans les langages de programmation et de spécification entre autres.

Un type abstrait de données est constitué, en plus de l'ensemble de ses objets. par la famille des opérateurs qui agissent sur ces objets. L'algèbre universelle constitue le fondement théorique des types abstraits de données. Une spécification algébrique d'un type abstrait de données est une algèbre qui est décrite par un ensemble de sortes, un ensemble d'opérateurs et un ensemble d'axiomes. Ce qui permet de la considérer comme une axiomatisation de la théorie du type de données qu'elle spécifie.

La syntaxe et la sémantique d'une spécification sont basées respectivement sur la signature d'algèbres et l'algèbre initiale de la classe d'algèbres.

Dans une spécification, le problème du mot consiste à tester si une équation est une conséquence de l'ensemble d'axiomes, c'est à dire si elle est vraie dans la théorie équationnelle associée.

Nous nous plaçons dans le cas de la logique équationnelle où la théorie est décrite par des axiomes forment un ensemble d'équations égalitaires entre des termes. Les termes sont des combinaisons bien formées d'un ensemble d'opérateurs et d'un ensemble de variables implicitement quantifiées universellement. En logique équationnelle le mécanisme de déduction se réduit à la seule règle du calcul équationnel qui est le remplacement d'égaux par des égaux. Sous certaines conditions, l'automatisation de la validation des identités est possible en utilisant la réécriture.

La réécriture est à l'origine une méthode de preuve en logique équationnelle mais son domaine d'utilisation n'a pas cessé de s'étendre depuis. La résolution d'équations, le calcul propositionnel et le calcul de prédicats en logique, la construction et la correction des spécifications algébriques et la transformation des programmes n'en sont que quelques exemples et ne forment pas une liste exhaustive. L'idée de base de la réécriture est d'orienter les équations dans un sens bien étudié, pour qu'elles puissent être utilisées comme des règles. Ainsi nous obtenons un processus déterministe et fini pour le calcul équationnel.

La théorie de la réécriture, basée sur la procédure de complétion de Knuth et Bendix

offre un mécanisme simple pour le problème du mot. Cette procédure complète l'ensemble des règles pour que l'égalité équationnelle et la relation de réécriture induite soient équivalentes. Si une spécification peut être transformée en un système de réécriture convergent, c'est à dire confluent et nœthérien, alors le test de la validité d'une équation, dans la variété équationnelle, se réduit au test de l'identité des formes normales des deux membres de l'équation.

Dans l'algèbre initiale de la variété équationnelle d'une spécification, un raisonnement équationnel ne suffit plus pour tester la validité d'un théorème inductif, un raisonnement par récurrence devient obligatoire dans ce cas. Il se trouve que la procédure de complétion de Knuth et Bendix peut être encore utilisée moyennant un certain nombre de modifications. Dans ce sens, il y a la méthode de preuve par consistance et la méthode de preuve basée sur la réductibilité inductive.

Nous nous restreignons au cas des spécifications mono-sortes et dont toutes les équations peuvent être orientées en des règles de réécriture.

La construction et la correction d'une spécification structurée sont assurées par les deux propriétés fondamentales de consistance et de complétude relative.

Nous nous intéressons à la preuve automatique de ces deux propriétés en utilisant la réécriture.

Nous nous intéressons aussi à la propriété d'ω-complétude d'une spécification qui exprime que tout théorème inductif est un théorème équationnel.

Par ailleurs nous nous intéressons à la reconnaissance et à la génération des formes normales closes, ou avec des variables, d'un système de réécriture en utilisant la réécriture et les automates et les grammaires d'arbres.

Il est à noter que toutes ces méthodes de preuves de validité, de complétude relative ainsi que la reconnaissance et de génération des formes normales closes d'un système de réécriture ont été implantées en langage CLU, dans le logiciel de réécriture REVE, sur les machines VAX et SUN, sous le système d'exploitation UNIX.

L'organisation de cette thèse est la suivante:

- Dans le premier chapitre nous rappelons les concepts théoriques de base décrivant le cadre de notre travail.
  La première partie est consacrée à l'algèbre universelle et à la variété équationnelle.
  La deuxième partie aux spécifications algébriques.
  La troisième partie est réservée aux systèmes de réécriture de termes.

- Le deuxième chapitre est consacré à la preuve de validité dans l'algèbre initiale d'une variété équationnelle. Il représente ainsi le cœur de notre travail et c'est le noyau des autres chapitres.
  La première partie est consacrée à la présentation des différentes approches de preuve par récurrence. Ce qui permet de dégager la méthode adoptée dans notre réalisation. Cette méthode est la synthèse de celle de preuve par consistance et de celle de preuve par réductibilité inductive. Elles seront étudiées respectivement dans les deux parties qui suivent.
  La quatrième partie est réservée à la confluence close.

- Le troisième chapitre décrit la méthode adoptée pour tester la complétude relative d'une spécification par rapport à la spécification de base.
  Dans la première partie nous exposons la propriété de convertibilité relative d'un opérateur pour tester la complétude relative.
  Dans la deuxième partie nous étudions le concept de recouvrement qui est à la base de la méthode adoptée.
  La troisième partie est consacrée au théorème de décidabilité de cette propriété.

- Le quatrième chapitre présente une étude systématique de la ω-complétude d'une spécification.
  Dans la première partie nous donnons la définition de la ω-complétude clarifiée par des exemples.
  Dans la deuxième partie nous exposons les deux méthodes, dégagées d'un certain nombre d'exemples de spécifications ω-complètes, pour la tester. Nous énumérons les problèmes qui se posent lors de la construction d'un enrichissement ω-complet d'une spécification donnée. Puis nous nous restreignons à la ω-complétude relative par rapport à un ordre nœthérien.
  La troisième partie est consacrée à sa décidabilité dans certains cas où le domaine de l'algèbre initiale associée à la spécification est fini.

- Le cinquième chapitre expose une méthode permettant la reconnaissance et la génération du langage des formes normales closes d'un système de réécriture.
  Dans la première partie nous rappelons les notions de bases des automates et des

grammaires dans le cas des arbres. Nous donnons aussi les théorèmes de décidabilité de la finitude, de la vacuité et de l'équivalence des langages d'arbres.
La deuxieme partie est consacrée à la construction et correction d'un automate déterministe qui reconnait le langage des formes normales closes, ou avec des variables, d'un système de réécriture de termes.
La troisième partie est consacrée à la construction et correction d'une grammaire qui engendre ce langage.

- Le sixième chapitre est consacré au laboratoire de réécriture REVE.
  Dans la première partie nous donnons une présentation générale du logiciel REVE. Dans la deuxième partie nous regroupons les procédures principales qui réalisent les différents concepts introduits. Dans la troisième partie nous donnons une seance complète de notre réalisation.

- Dans la conclusion nous faisons un bilan de ce travail ainsi que les principaux problèmes résultants qui restent ouverts.

Définition 1.3 *Soient $P = (S_P, F_P)$ et $Q = (S_Q, F_Q)$ deux algèbres. Nous disons que Q est une sous-algèbre de P si et seulement si*

- $S_Q \subseteq S_P$,
- $\forall f \in F \quad f_Q = f_{P/Q}$ *où* $f_{P/Q}$ *est la restriction de* f_P *à* Q.

Définition 1.4 *Soit $(P_i = (S_{P_i}, F_{P_i}))_{i \in I}$ une famille indexée d'algèbres. Le produit des algèbres $(P_i = (S_{P_i}, F_{P_i}))_{i \in I}$ est l'algèbre $P = (S_P, F_P)$ où*

- $S_P = \prod_{i \in I} S_{P_i}$ *produit cartésien.*
- $\forall n \in N \quad \forall f \in F_n \quad \forall (a_1, \ldots, a_n) \in (\prod_{i \in I} S_{P_i})^n$
  $\prod_i (f_P(a_1, \ldots, a_n)) = f_{P_i}(\prod_i (a_1), \ldots, \prod_i (a_n))$
  où $\prod_i$ *est l'application i-ième projection.*

Définition 1.5 *Une classe C non vide d'algèbres est une variété si et seulement si elle est fermée par sous-algèbres, image homomorphe et produit.*

Algèbre libre

Définition 1.6 *Soient C une classe quelconque d'algèbres et X un ensemble dénombrable, éventuellement fini. Nous appelons C-algèbre libre engendrée par X, ou sur X, toute algèbre $P = (S_P, F_P)$ telle que:*

- $P \in C$,
- $X \subseteq S_P$,
- $\forall Q = (S_Q, F_Q) \in C \quad \forall h_o : X \longrightarrow S_Q \quad \exists h : P \longrightarrow Q$ *où h est un unique homomorphisme tel que* $h/_X = h_o$.

Convention 1.4
Nous parlerons d'algèbre libre sur X au lieu de C-algèbre libre sur X s'il n'y a pas d'ambiguïté sur C.
Les éléments de X sont appelés des variables.

L'algèbre libre sur X, si elle existe, est unique à un isomorphisme prés.

Nous supposons dans toute la suite que F est un ensemble fixe de fonctions et X est un ensemble de variables quelconque tels que $F \cap X = \emptyset$ et $F_0 \neq \emptyset$.

Le théorème suivant, qui est un cas particulier d'un théorème dû à G. Birkhoff [5], nous permet de définir l'ensemble des termes.

Théorème 1.1 *Soit C la classe de toutes les algèbres sur X. L'algèbre libre sur X existe.*

Nous avons vu que l'algèbre libre sur un ensemble X quelconque est unique à un isomorphisme prés. Dans le cas où X est vide nous obtenons la notion d'*algèbre initiale.* Dans le cas où X est dénombrable nous obtenons la notion d'*algèbre générique.*

Définition 1.7 *L'algèbre libre sur l'ensemble $\emptyset$ est appelée* algèbre initiale *et notée $I(F)$. L'algèbre libre sur un ensemble X dénombrable est appelée* algèbre générique *et notée $G(F, X)$.*

Dans toute la suite, nous prenons la classe de toutes les algèbres sur X.

Nous allons maintenant définir la notion de *terme* qui est l'élément de base de la réécriture comme nous le verrons dans le deuxième paragraphe.

1.1.2 Termes

Définition 1.8 *Nous appelons ensemble des termes de premier ordre sur X, noté $T(F, X)$, le domaine de l'algèbre libre sur X.*

Convention 1.5
Nous parlerons de terme au lieu de terme de premier ordre.
L'ensemble des termes $T(F, \emptyset)$ sans variables, noté $T(F)$, est le domaine de l'algèbre initiale $I(F)$ et appelé ensemble des termes clos.
Par abus de notation, nous désignerons par $T(F, X)$ l'algèbre libre sur X et par $T(F)$ l'algèbre initiale.
Si $X = \{x_1, x_2, \ldots, x_n, \ldots\} = \{x_i \mid i \in N\}$ noté X_ω, $T(F, X_\omega)$ est appelé l'algèbre générique.

Exemple 1.3 *Soient $F = \{0, s, +\}$ et $X = \{x_1, \ldots, x_n, \ldots\}$. Les expressions suivantes sont des termes, éléments de $T(F, X)$: $0, s(0), s(x_1), x_1 + (0 + x_2), x_1 + s(x_2), s(x_1 + x_2)$.*

Le concept de terme est identique à celui d'expression formelle que nous pouvons construire d'après l'ensemble X et la signature $(F.arité)$. Formellement, ce concept est défini de façon constructive comme suivant:

Lemme 1.1 *L'ensemble $T(F, X)$ des termes est le plus petit ensemble tel que*

- $X \subseteq T(F, X)$,
- $\forall n \in N \quad \forall f \in F_n \quad \forall (t_1, \ldots, t_n) \in T(F, X)^n \quad f(t_1, \ldots, t_n) \in T(F, X)$.

Cette définition des termes permet de disposer d'un principe d'induction structurelle sur $T(F, X)$.

Définition 1.9 *Soit P une propriété.*
si

- *induction structurelle de base:* $\forall x \in X \quad P(x)$ *et* $\forall c \in F_0 \quad P(c)$.
- *une étape d'induction structurelle:* $\forall n \in N^+ \quad \forall f \in F_n \quad \forall (t_1, \ldots, t_n) \in T(F.X)^n$ $P(t_1) \wedge \cdots \wedge P(t_n) \implies P(f(t_1, \ldots, t_n))$,

alors $\forall t \in T(F, X) \quad P(t)$.

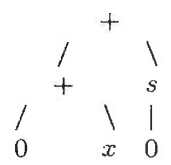
Nous allons donner une représentation arborescente aux termes qu'il est d'usage d'utiliser concurremment à la notation habituelle sous forme fonctionnelle. Soit $(N^*, .)$ le monoïde libre engendré par N^+, ensemble des entiers strictement positifs. '.' est l'opération de concaténation, elle sera omise si elle n'est pas ambiguë.

Définition 1.10 *Un terme t est une arborescence ordonnée étiquetée par $F \cup X$. C'est à dire une application partielle de N^* dans $F \cup X$ dont le domaine, noté $O(t)$, vérifie:*

- *le mot vide* $\varepsilon \in O(t)$,
- *si* $u \in O(t_i)$ *alors* $iu \in O(f(\ldots, t_i, \ldots))$ *si et seulement si* $1 \leq i \leq$ *arité*(f).

Convention 1.6
Les éléments de $O(t)$ sont appelés occurrences de t et elles représentent les différents chemins d'accés dans l'arborescence de t.
Dans la suite, sauf mention contraire, nous ne ferons pas de distinction entre un terme et sa représentation arborescente.

Exemple 1.4 *Etant donné le terme* $t = (0 + x) + s(0)$ *alors* $O(t) = \{\varepsilon, 1, 11, 12, 2, 21\}$ *et sa représentation arborescente est:*



Définition 1.11 *Soit t un terme. Nous disons que t' est un sous-terme de t si et seulement si il existe $u \in O(t)$ telle que t à l'occurrence u, noté t/u, soit égal à t'. Si $u \neq \varepsilon$, t' est un sous-terme strict.*

Définition 1.12 *Soient t et t' deux termes et u une occurrence de t. $t[u \longleftarrow t']$ est le terme obtenu en remplaçant le terme t/u par t' dans t.*

Convention 1.7
Dans la suite, nous désignons par $V(t)$ l'ensemble des variables de t défini par:
$V(t) = \{x \in X \mid \exists u \in O(t)\ \ t/u = x\}$.
L'ensemble des occurrences closes de t, qui ne correspondent pas à des variables, est désigné par $G(t)$ et défini par: $G(t) = \{u \in O(t) \mid \ t/u \notin X\}$.

Définition 1.13 *Un terme t est dit linéaire si et seulement s'il ne contient qu'une occurrence au plus de chaque variable.*

Exemple 1.5 *Soient $t = (0 + x) + s(0)$ et $t' = 0 + x$,*
t' est un sous-terme de t à l'occurrence 1.
$t'' = t[21 \longleftarrow t'] = (0 + x) + s(0 + x)$.
$V(t) = \{x\}$ et $G(t) = \{\varepsilon, 1, 11, 2, 21\}$,
t est linéaire mais t'' ne l'est pas.

Définition 1.14 *Une substitution σ est une application de X dans $T(F, X)$. Le domaine de σ, noté $D(\sigma)$, est le sous-ensemble fini de X défini par: $D(\sigma) = \{x \in X \mid \sigma(x) \neq x\}$.*

Convention 1.8
Remarquons que le domaine de σ est vide si et seulement si σ est l'identité.
L'ensemble des substitutions est noté $\Sigma(F, X)$.
L'ensemble des substitutions closes, c'est à dire à valeur dans $T(F)$, est noté $\Sigma(F)$.
Une substitution σ sera représentée par l'ensemble: $\{x \longleftarrow \sigma(x) \mid x \in D(\sigma)\}$.

L'algèbre $T(F, X)$ étant libre, une telle application se prolonge de façon unique en un endomorphisme de $T(F, X)$. Elle vérifie donc:

$$\forall n \in N \quad \forall f \in F_n \quad \forall (t_1, \ldots, t_n) \in T(F, X)^n \quad \sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n)).$$

Définition 1.15 *Soient σ et ρ deux substitutions, le composé de σ et ρ, noté $\sigma\rho$, est la substitution définie par: $\forall x \in X \quad \sigma\rho(x) = \sigma(\rho(x))$.*

1.1.3 Théorie équationnelle

Variété équationnelle

Définition 1.16 *Nous appelons équation un couple de termes (t_1, t_2), notée $t_1 = t_2$.*

Définition 1.17 *Soient $t_1 = t_2$ une équation et $P = (S_P, F_P)$ une algèbre. P valide, ou satisfait, $t_1 = t_2$ si et seulement si $\forall h : T(F, X) \longrightarrow S_P \quad h(t_1) = h(t_2)$.*

Convention 1.9
Nous disons aussi que l'équation $t_1 = t_2$ est valide dans P, ou P est un modèle de $t_1 = t_2$.

et noté $P \models t_1 = t_2$.
P valide un ensemble d'équations A si et seulement si P valide chaque équation de A.
Une classe d'algèbres C valide A si et seulement si chaque algèbre de C valide A.

Définition 1.18 *Soit A un ensemble d'équations. Nous appelons variété équationnelle d'algèbres engendrée par A, que nous nous notons $M(A)$, la classe de toutes les algèbres validant A.*

Convention 1.10
$M(A)$ est appelée aussi classe des modèles de A.
Les équations de A sont appelées les axiomes d'une axiomatisation de la variété équationnelle $M(A)$.

Théorème 1.2 *Soit C une classe d'algèbres. C est une variété équationnelle d'algèbres si et seulement si C est une variété.*

Par conséquent, la $M(A)$-algèbre libre sur X existe, pour tout ensemble X. Nous dirons aussi que $M(A)$ est la variété équationnelle engendée par A sur X.

Nous allons maintenant construire la $M(A)$-algèbre libre sur X, à partir de l'ensemble des termes $T(F, X)$ et de la congruence engendrée par A.

Définition 1.19 *Soit $P = (S_P, F_P)$ une algèbre. Une relation $\sim$ sur S_P est une congruence sur P si et seulement si $\sim$ est une relation d'équivalence stable par les opérateurs de F_P.*
C'est à dire: $\forall n \in N \quad \forall f \in F_n \quad \forall (t_1, \ldots, t_n, t'_1, \ldots, t'_n) \in S_P^{2\star n}\ t_1 \sim t'_1 \wedge \cdots \wedge t_n \sim t'_n \implies f(t_1, \ldots, t_n) \sim f(t'_1 \ldots, t'_n)$.

La relation $\sim$ étant une relation d'équivalence, nous pouvons définir l'algèbre quotient comme étant égale à $(S_P/_\sim, F_P)$ et noté $P/_\sim$.

Exemple 1.6 *Reprenons l'exemple 1.2. Les opérateurs* 0 *et s engendrent les termes* 0 *et* $s^n(0)$ *avec* $n \geq 1$. *qui correspondent aux entiers naturels. Tout autre terme contenant* 0. *s et* + *est équivalent à l'un des termes suivants:* 0 . $s^n(0)$ *avec* $n \geq 1$. *Par conséquent l'algèbre quotient* $(\overline{N}, \overline{F})$ *est définie par:*

$\overline{N} = \{0\} \cup \{s^n(0) \mid n \geq 1\}$
$\overline{F} = \{\overline{0}, \overline{s}, \overline{+}\}$ *tels que :*

$\overline{0}$	=	0
$\overline{s}(0)$	=	$s(0)$
$\overline{s}(s^n(0))$	=	$s^{n+1}(0)$
$0\overline{+}0$	=	0
$0\overline{+}s^n(0)$	=	$s^n(0)$
$s^n(0)\overline{+}0$	=	$s^n(0)$
$s^n(0)\overline{+}s^m(0)$	=	$s^{n+m}(0)$.

et $h : \overline{N} \longrightarrow N$ *est un isomorphisme défini par:*
$h(0) = 0$
$h(s^n(0)) = n$ *si* $n \geq 1$.

Problème de la validité

Soient A un ensemble d'équations et $t_1 = t_2$ une autre équation. Nous allons maintenant présenter le problème de la validité, ou problème de mots, dans la variété équationnelle $M(A)$ qui consiste à décider la validité de l'équation $t_1 = t_2$ dans toute algèbre de $M(A)$.

Définition 1.20 *La plus petite congruence sur* $T(F, X)$ *contenant tous les couples* $(\sigma(s_1), \sigma(s_2))$ *où* $s_1 = s_2$ *est une équation quelconque de* A *et* σ *une substitution, est appelée la congruence engendée par* A, *ou la* A*-égalité, et notée* $=_A$.

Convention 1.11
Si une équation $t_1 = t_2$ *est valide dans* $M(A)$, *nous notons* $t_1 =_A t_2$ *et nous lisons* $t_1 = t_2$ *modulo* A, *ou* t_1 *est* A*-égal à* t_2.

Définition 1.21 *La relation symétrique* A*-égalité en une étape, notée* $\vdash\!\dashv_A$, *est définie de la manière suivante:* $t_1 \vdash\!\dashv_A t_2$ *si et seulement si*
$\exists g = d \in A \quad \exists u \in O(t_1) \quad \exists \sigma \in \Sigma(F, X)$ *tels que:* $t_1/u = \sigma(g)$ *(ou resp.* $t_1/u = \sigma(d)$*) et* $t_2 = t_1[u \longleftarrow \sigma(d)]$ *(ou resp.* $t_2 = t_1[u \longleftarrow \sigma(g)]$*).*

Nous donnons ci-dessous une définition constructive de la congruence engendrée par un ensemble d'équations A.

Définition 1.22 *Soient* t *et* t' *deux termes.* $t =_A t'$ *si et seulement si* $\exists (t_1, \ldots, t_n) \in T(F, X)^n$ *tel que:*

- $t = t_1$,
- $t_i \vdash\!\dashv_A t_{i+1}$ *pour* $1 \leq i < n$,
- $t_n = t'$.

Définition 1.23 *Nous appelons théorie équationnelle engendrée par* A *l'algèbre quotient* $(T(F, X)/_{=_A}, F)$ *et par abus de langage la congruence elle même.*

Lorsque A est l'ensemble vide nous parlerons de la théorie vide.

Théorème 1.3 *La* $M(A)$*-algèbre libre sur* X *est l'algèbre quotient* $(T(F, X)/_{=_A}, F)$.

L'algèbre libre d'une variété équationnelle a une très grande importance car la validité d'une équation se ramène à une preuve de A-égalité. Elle a donc une caractérisation syntaxique, comme nous allons le voir dans le théorème suivant, dû à G. Birkhoff [5], et qui est à la base de la logique équationnelle.

Théorème 1.4 *Soit A un ensemble d'équations. Une équation $t_1 = t_2$ est valide dans $M(A)$ si et seulement si $t_1 =_A t_2$.*

Ce théorème montre qu'une équation $t_1 = t_2$ est valide dans la variété équationnelle $M(A)$ si et seulement si elle peut être déduite des équations de A par un raisonnement équationnel, c'est à dire un nombre fini de substitutions et de remplacements de termes par des égaux. Nous disons aussi que $t_1 = t_2$ est une conséquence de A et nous le notons $A \vdash t_1 = t_2$.

Définition 1.24 *Dans le cas où X est vide, la $M(A)$-algèbre libre sur $\emptyset$ est l'algèbre initiale de la variété équationnelle engendrée par A, notée $I(F, A)$, dont le domaine est l'ensemble quotient $T(F)/_{=_A}$.*
Dans le cas où le cardinal de X est dénombrable, la $M(A)$-algèbre libre sur X est l'algèbre générique de la variété équationnelle engendrée par A, notée $G(F, X, A)$, dont le domaine est l'ensemble quotient $T(F, X)/_{=_A}$.

Le problème de la validité dans l'algèbre initiale se résume dans la proposition suivante.

Proposition 1.1 *Une équation $t_1 = t_2$ est valide dans l'algèbre initiale $I(F)$ (resp. $I(F, A)$) si et seulement si $\forall \sigma \in \Sigma(F) \quad \sigma(t_1) = \sigma(t_2)$ (resp. $\sigma(t_1) =_A \sigma(t_2)$).*

Exemple 1.7 *Soient $F = \{0, s, +\}$ et A l'ensemble d'équations suivant:*

$$A = \{ \begin{array}{lll} x+0 & == & x \\ x+s(y) & == & s(x+y) \end{array} \}.$$

L'équation:

$$x + (y + s(z)) = s(x + (y + z))$$

est valide dans $I(F)$.
Les deux équations suivantes:

$$\begin{array}{lll} x+y & = & y+x \\ x+(y+z) & = & (x+y)+z \end{array}$$

sont valides dans $I(F, A)$ mais pas dans $I(F)$ car elles ne peuvent pas être prouvées uniquement par un raisonnement équationnel à partir des équations de A. La preuve nécessite un raisonnement par récurrence, une induction structurelle, sur les variables.

1.1.4 Satisfiabilité

La notion de satisfiabilité est la notion duale de celle de validité.

Définition 1.25 *Soient t_1 et t_2 deux termes de $T(F,X)$ et $P = (S_P, F_P)$ une algèbre. L'équation $t_1 = t_2$ est satisfiable (resp. semi-satisfiable) dans P s'il existe un homomorphisme h de $T(F,X)$ dans S_P tel que $h(t_1) = h(t_2)$ (resp. $t_1 = h(t_2)$).*

Certains problèmes sont reliés avec la notion générale de la satisfiabilité (resp. semi-satisfiabilité) c'est le cas pour les problèmes suivants:

- le problème de l'unification (resp. filtrage): c'est la notion de satisfiabilité (resp. semi-satisfiabilité) dans $T(F,X)$. C'est à dire étant donnés deux termes t_1 et t_2, il faut trouver une substitution σ telle que $\sigma(t_1) = \sigma(t_2)$ (resp. $t_1 = \sigma(t_2)$).

- le problème de l'unification modulo A (resp. filtrage modulo A): c'est la notion de satisfiabilité (resp. semi-satisfiabilité) dans l'ensemble quotient $T(F,X)/_{=_A}$. C'est à dire étant donnés deux termes t_1 et t_2, il faut trouver une substitution σ telle que $\sigma(t_1) =_A \sigma(t_2)$ (resp. $t_1 =_A \sigma(t_2)$).

Convention 1.12
Dans le cas de l'unification nous disons que σ unifie t_1 et t_2, ou σ est l'unificateur de t_1 et t_2.
Dans le cas de filtrage, nous disons que σ filtre t_2 vers t_1, ou σ est le filtre de t_2 vers t_1. ou que t_2 schématise t_1, ou t_1 est une instance de t_2.

Définition 1.26 *Soient σ et ρ deux substitutions. La relation $\ll$ sur les substitutions est définie par:* $\forall(\sigma,\rho) \in \Sigma(F,X)^2 \quad \sigma \ll \rho \iff \exists\varphi \in \Sigma(F,X) \quad \forall x \in X \quad \varphi\sigma(x) = \rho(x)$.

S'il existe un unificateur de t_1 et t_2 alors il existe un unificateur minimal, ou dit principal, par rapport à $\ll$. Dans ce cas il est unique à un isomorphisme près des noms des variables de t_1 et t_2.

Exemple 1.8 *Soient $t_1 = (x+0)+s(y)$ et $t_2 = (x+0)+s(0)$. $\sigma = \{y \longleftarrow 0\}$ filtre t_2 vers t_1. Soit $t_3 = z + s(0)$, $\sigma = \{z \longleftarrow x+0, y \longleftarrow 0\}$ unifie t_1 et t_3, c'est aussi l'unificateur principal de t_1 et t_3.*

Dans le paragraphe suivant, nous présenterons les notions fondamentales de la réécriture qui vont être utilisées pour la preuve de la validité dans une variété équationnelle et dans l'algèbre initiale associée.

1.2 Spécification

Un type abstrait de données spécifie une structure de donnée abstraite, en précisant seulement les données, ou dit les objets, et les opérateurs qui agissent sur ces données. Les propriétés caractéristiques des opérateurs sont précisées sans se préoccuper des algorithmes qui les réalisent. C'est la raison pour la quelle nous le qualifions d'algébrique.

Une spécification représente une description syntaxique des types abstraits de données. La sémantique d'une spécification algébrique est donnée par l'algèbre initiale de la variété équationnelle associée à la spécification [16].

Définition 1.27 *Nous appelons spécification algébrique d'un type abstrait de donnée un triplet* (S, F, A) *où*

- S *est un ensemble de sortes,*
- F *un ensemble d'opérateurs,*
- A *un ensemble d'axiomes.*

Dans toute la suite, nous nous restreindrons aux spécifications mono-sortes, c'est à dire aux cas où S est un singleton. Nous nous restreindrons aussi aux axiomes équationnels, c'est à dire de la forme $t_1 = t_2$ où t_1 et t_2 sont deux termes construits uniquement à partir des opérateurs de F et de variables implicitement quantifiées universellement.

Définition 1.28 *Nous appelons type abstrait de donnée spécifié par* (S, F, A) *la sous-classe de* F*-algèbres modèles de* A *dont les éléments sont isomorphes.*

Chaque élément de la sous-classe est une représentation des données du type abstrait de donnée.

La construction d'une spécification se fait à partir d'une spécification primitive, de base, à la quelle nous appliquons les opérations suivantes:

- l'*enrichissement*: ajout de nouveaux opérateurs et de nouveaux axiomes.
- l'*extension*: ajout de nouvelles sortes,

qui expriment les deux concepts fondamentaux en programmation structurée.

Chaque spécification (S, F, A) est associée à l'algèbre initiale $I(F, A)$. Nous voulons que l'algèbre initiale de départ soit isomorphe à la restriction de la nouvelle algèbre initiale aux anciennes sortes et opérateurs. Ce qui introduit les deux propriétés fondamentales des spécifications que sont la *consistance* et la *complétude relative*, pour exprimer la *cohérence relative* des spécifications.

Définition 1.29 *Une spécification (S_o, F_o, A_o) est une sous-spécification d'une spécification (S, F, A) si et seulement si S_o, F_o et A_o sont inclus respectivement dans S, F et A. Lorsque $S = S_o$ nous parlons d'enrichissement sinon nous parlons d'extension.*

Définition 1.30 *Une extension (S, F, A) de (S_o, F_o, A_o) est relativement consistante par rapport à (S_o, F_o, A_o) si et seulement si la restriction de la congruence $=_A$ à $T(F_o)$ coïncide avec la congruence $=_{A_o}$. C'est à dire $\forall (t, t') \in T(F_o)^2 \quad t =_A t' \implies t =_{A_o} t'$.*

Dans le chapitre 2, nous présenterons une méthode basée sur la réécriture pour tester la consistance relative d'une spécification donnée par rapport à la spécification de base.

Définition 1.31 *Une extension (S, F, A) de (S_o, F_o, A_o) est relativement complète par rapport à (S_o, F_o, A_o) si et seulement si $\forall t \in T(F) \quad \exists t' \in T(F_o) \quad t =_A t'$.*

Dans le chapitre 3, nous présenterons une méthode basée sur l'unification et la réécriture pour tester la complétude relative d'une spécification donnée par rapport à la spécification de base.

Lorsque la spécification (S, F, A) est un enrichissement de (S_o, F_o, A_o), c'est à dire $S = S_o$, (S, F, A) est relativement consistante (resp. complète) par rapport à (S_o, F_o, A_o) si et seulement si il existe un homomorphisme injectif (resp. surjectif) de $T(F)/_{=_A}$ dans $T(F_o)/_{=_{A_o}}$.

1.3 Réécriture

Le problème de la validité dans une variété équationnelle est indécidable en général. Mais ce n'est pas le cas pour un grand nombre de théories équationnelles classiques lorsque nous utilisons la méthode de la réécriture.

Dans la théorie équationnelle nous avons défini la congruence $=_A$ engendrée par l'ensemble d'équations A. Et comme nous avons vu, une équation $t_1 = t_2$ est valide dans $M(A)$ si et seulement si $t_1 =_A t_2$. L'idée de base de cette méthode consiste à déterminer pour chaque classe d'équivalence, de la congruence $=_A$, un représentant canonique. Ce qui ramène le problème de la A-égalité des deux termes t_1 et t_2 à celui de l'identité de leurs représentants canoniques.

Le principe de base de la réécriture consiste à orienter les équations de A et à les appliquer toujours dans la même direction, ce qui diffère du raisonnement équationnel où les équations sont utilisées de façon symétrique et indéterministe dans les deux sens. Cela évitera les retours arrières, *backtraking*, et tous ses inconvénients éventuels. Ainsi les équations orientées, appelées *règles*, forment un *système de réécriture de termes* R et la congruence $=_A$ devient la relation de réécriture $\longrightarrow_R$ associée à R. Cette méthode impose

l'existence et l'unicité de la forme normale. C'est à dire à chaque terme est associé un et un seul terme qui n'est plus réductible par $\longrightarrow_R$. Ce que nous exprimons respectivement par les propriétés de terminaison et de confluence du système de réécriture. Cette condition restreint le champ d'application de cette méthode.

1.3.1 Système de réécriture

Définition 1.32 *Un système de réécriture de termes dans $T(F, X)$ est un ensemble fini R de paires ordonnées de termes tel que: $\forall (g, d) \in R \quad V(d) \subseteq V(g)$.*

Convention 1.13
Une paire (g, d) de R sera noté $g \rightarrow d$ et appelée règle de réécriture, avec g le membre gauche et d le membre droit de la règle.
Un système de réécriture est dit linéaire lorsque le membre gauche et le membre droit de chaque règle sont linéaires.

Etant donné R un ensemble de paires ordonnées de $T(F, X)$, un système de réécriture peut être considéré comme la spécification explicite d'une relation finie dans $T(F, X)$.

Exemple 1.9 *Soit A l'ensemble d'équations suivant:*

$$A = \{ \begin{array}{ll} x+0 & == \ x \\ x+s(y) & == \ s(x+y) \end{array} \}.$$

définissant l'opérateur $+$ dans les entiers naturels. Ces équations peuvent être orientées en un ensemble de règles R construisant le système de réécriture de termes suivant:

$$R = \{ \begin{array}{ll} x+0 & \rightarrow \ x \\ x+s(y) & \rightarrow \ s(x+y) \end{array} \}.$$

Définition 1.33 *Soit R un système de réécriture. Nous appelons relation de réécriture induite par R dans $T(F, X)$ la plus petite relation $\longrightarrow_R$ définie dans $T(F, X)$ qui contient les règles de R et qui est fermée par les opérations de l'algèbre, ou de remplacement, et par substitutions.*

Formellement, la relation $\longrightarrow_R$ associée à un système de réécriture est la plus petite relation définie dans $T(F, X)$ et telle que:

- $\forall g \rightarrow d \in R \quad g \longrightarrow_R d$,
- $s \longrightarrow_R t \implies \forall r \in T(F, X) \ \forall u \in O(r) \ r[u \longleftarrow s] \longrightarrow_R r[u \longleftarrow t]$,
- $s \longrightarrow_R t \implies \forall \sigma \in \Sigma(F, X) \ \sigma(s) \longrightarrow_R \sigma(t)$.

Convention 1.14 *Nous disons que s se réécrit en t lorsque $s \longrightarrow_R t$.*

Lemme 1.2 *Soient R un système de réécriture et $\longrightarrow_R$ la relation de réécriture associée à R. La relation $\longrightarrow_R$ peut être définie comme suit:*
$\forall (s,t) \in T(F,X)$ $s \longrightarrow_R t$ *si et seulement si*

- $s \rightarrow t \in R$
  ou
- $\exists u \in G(s)$ $\exists g \rightarrow d \in R$ $\exists \sigma \in \Sigma(F,X)$ *tels que* $s/u = \sigma(g)$ *et* $t = s[u \longleftarrow \sigma(d)]$
  noté parfois $s \longrightarrow_R t_{[u,g \rightarrow d,\sigma]}$.

Convention 1.15
Le sous-terme s/u est appelé un radical lorsque $s \longrightarrow_R t_{[u,g \rightarrow d,\sigma]}$.
Soit $\longrightarrow_R$ une relation arbitraire définie dans $T(F,X)$. Les notations $\xrightarrow{+}_R$, $\xrightarrow{*}_R$ et $\xleftrightarrow{*}_R$ représentent respectivement les fermetures transitive, réflexive-transitive et réflexive-symétrique-transitive de la relation $\longrightarrow_R$ définie dans $T(F,X)$. R sera omis quand il n'y a pas d'ambiguïté.

Définition 1.34 *Nous disons qu'un terme t est réductible par rapport à un système de réécriture R si et seulement si $\exists t' \in T(F,X)$ tel que $t \longrightarrow t'$. Une forme normale de t par rapport à R, notée $t\downarrow$, est un terme t' irréductible tel que $t \xrightarrow{*} t'$.*

Exemple 1.10 *Reprenons l'exemple 1.9. Le terme $0 + (0 + 0)$ est un terme réductible dont la forme normale est 0. $s(0)$ est irréductible.*

Convention 1.16
L'ensemble des formes normales des termes de $T(F,X)$, noté $T(F,X)\downarrow_R$, est le domaine de l'algèbre générique de la variété équationnelle engendrée par R sur X.
L'ensemble des formes normales closes des termes de $T(F)$, noté $T(F)\downarrow_R$, est le domaine de l'algèbre initiale de la variété équationnelle engendrée par R.

Pour assurer l'existence et l'unicité de la forme normale des termes, nous avons besoin d'introduire deux propriétés essentielles d'un système de réécriture de termes qui sont la *terminaison* et la *confluence* [39].

Nous allons présenter maintenant la propriété de terminaison qui exprime que tout calcul d'un terme par réécriture est fini.

1.3.2 Terminaison

Définition 1.35 *Un système de réécriture est dit nœthérien, ou termine, ou à terminaison finie, s'il n'existe pas de chaine infinie de termes de la forme $t_1 \longrightarrow t_2 \longrightarrow \cdots \longrightarrow t_n \longrightarrow \cdots$. Nous disons aussi dans ce cas que la relation $\longrightarrow$ est nœthériene.*

En d'autre termes, un système de réécriture est nœthérien si et seulement si la relation $\longrightarrow$ associée définit un ordre partiel bien fondé sur $T(F, X)$.

Quand un système de réécriture est nœthérien, tout terme a au moins une forme normale.

La preuve de terminaison d'un système de réécriture est un problème indécidable en général [15], [27]. Nous ne nous étendrons pas sur les diverses méthodes existantes pour prouver la terminaison, mais nous renvoyons aux travaux suivants: [3], [14], [30], [32], [37], [51], [62], [64]. Néanmoins nous présentons un résumé des ordres utilisés dans la suite pour assurer la terminaison des systèmes de réécriture. Ces ordres sont basés sur la vérification que toute instance du membre droit de chaque règle de réécriture est strictement plus petit que l'instance correspondante du membre gauche.

Définition 1.36 *Soit $\prec$ un ordre défini sur $T(F, X)$. Nous disons que $\prec$ est ordre de simplification si et seulement si $\forall (s,t) \in T(F,X)^2 \quad \forall f \in F$*

- *compatibilité:* $s \prec t \implies \forall r \in T(F,X) \ \forall u \in O(r) \ r[u \longleftarrow s] \prec r[u \longleftarrow t]$.
- *sous-terme:* $t \prec f(\ldots, t, \ldots)$.

Définition 1.37 *Nous disons que $\prec$ est un ordre de réduction si et seulement si*

- *$\prec$ est un ordre de simplification.*
- *stabilité:* $\forall (s,t) \in T(F,X)^2 \quad \forall \sigma \in \Sigma(F,X) \quad s \prec t \implies \sigma(s) \prec \sigma(t)$.

Théorème 1.5 *[14] Soient $\prec$ un ordre de réduction et R un système de réécriture. Si $\forall g \to d \in R \quad d \prec g$ alors R est nœthérien.*

Les ordres de réductions sur les termes que nous utiliserons pour prouver la terminaison d'un système de réécriture sont:

- l'ordre proposé par D. Plaisted et N. Dershowitz, *Recursive Path Ordering*, qui est construit à partir d'un ordre partiel sur les opérateurs, [14], [58].
- l'ordre de S. Kamin et J. J. Levy, *Lexicographic Recursive Path Ordering*, [34].
- l'ordre proposé par P. Lescanne, *Recursive Decomposition Ordering*, [33], [48].
- l'ordre proposé par M. Rusinowitch, *Improved Recursive Decomposition Ordering with Status*, [64].
- l'ordre étudié par A. B. Cherifa basé sur les interprétations polynomiales, [3].

La réécriture est un processus indéterministe, et un terme pourra donc avoir plusieurs formes normales selon le chemin du calcul choisi. Nous allons maintenant présenter la propriété de confluence qui exprime que le résultat du calcul d'un terme par réécriture est unique, indépendamment du choix de la règle appliquée au terme à un moment donné.

1.3.3 Confluence

La propriété de Church-Rosser exprime comment nous utilisons les règles de réécriture pour faire des déductions équationnelles.

Définition 1.38 *Un système de réécriture possède la propriété de Church-Rosser si et seulement si*

$$\forall (t_1, t_2) \in T(F,X)^2 \quad t_1 \overset{*}{\longleftrightarrow} t_2 \quad \Longrightarrow \quad \exists t' \in T(F,X) \quad \begin{array}{ccc} & \overset{*}{\searrow} & \\ t_1 & & t' \\ t_2 & \overset{*}{\nearrow} & \end{array}$$

Ce qui exprime que l'équation $t_1 = t_2$ est valide dans $M(A)$ si et seulement si t_1 et t_2 ont la même forme normale par le système de réécriture associé à A. Rappelons que $\overset{*}{\longleftrightarrow}_R$ est équivalente à $=_A$ où R est le système de réécriture associé à A.

Définition 1.39 *Un système de réécriture est confluent si et seulement si*

$$\forall (t, t_1, t_2) \in T(F,X)^3 \quad \begin{array}{ccc} & \overset{*}{\nearrow} t_1 & \\ t & & \\ & \underset{*}{\searrow} t_2 & \end{array} \quad \Longrightarrow \quad \exists t' \in T(F,X) \quad \begin{array}{ccc} & \overset{*}{\searrow} & \\ t_1 & & t' \\ t_2 & \overset{*}{\nearrow} & \end{array}$$

Remarquons que la propriété de confluence implique la propriété de Churche-Rosser.

Quand un système de réécriture est confluent, tout terme a au plus une forme normale.

Le test de la confluence d'un système de réécriture est un problème indécidable en général. Mais ce n'est pas le cas pour les systèmes de réécriture nœthériens, comme nous allons le voir ci-dessous. L'idée de base de la décidabilité de la confluence est de localiser le test de la confluence.

Définition 1.40 *Un système de réécriture est localement confluent si et seulement si*

$$\forall (t, t_1, t_2) \in T(F,X)^3 \quad \begin{array}{ccc} & \nearrow t_1 & \\ t & & \\ & \underset{*}{\searrow} t_2 & \end{array} \quad \Longrightarrow \quad \exists t' \in T(F,X) \quad \begin{array}{ccc} & \overset{*}{\searrow} & \\ t_1 & & t' \\ t_2 & \overset{*}{\nearrow} & \end{array}$$

Lemme 1.3 *[55] Un système de réécriture nœthérien est confluent si et seulement si il est localement confluent.*

Ce lemme trouve son importance dans le fait qu'il existe un moyen simple, basé sur la notion de *paire critique*, pour tester la confluence locale d'un système de réécriture.

La notion de paire critique exprime l'indéterminisme du calcul par réécriture du système de réécriture. Nous supposons dans toute la suite que les variables des membres gauches des règles d'un système de réécriture R sont distinctes:

$$\forall (g \to d, l \to r) \in R^2 \quad V(g) \cap V(l) = \emptyset.$$

Condition qui n'est pas une restriction, car elle peut toujours être réalisée par un renommage des variables.

Définition 1.41 *Soient t et t' deux termes de $T(F, X)$. Nous disons que t' se superpose sur t à l'occurrence $u \in G(t)$ avec une substitution σ si et seulement si σ est l'unificateur principal de t' et t/u.*

Définition 1.42 *Soient $g \to d$ et $l \to r$ deux règles de R telles que l se superpose sur g à l'occurrence u avec la substitution σ. Le couple $(\sigma(g[u \longleftarrow r]).\sigma(d))$ est appelé paire critique de $l \to r$ sur $g \to d$ à l'occurrence u.*

Définition 1.43 *Nous appelons paire critique de R, toute paire critique obtenue par superposition de $l \to r$ sur $g \to d$ pour tout choix possible de couple de règles $g \to d$ et $l \to r$ de R et tout choix possible d'occurrence u de g.*

Exemple 1.11 *Soient $F = \{e, .\}$ et R le système de réécriture suivant:*

$$R = \{ \begin{array}{lll} 1 & x.e & \to x \\ 2 & (x.y).z & \to x.(y.z) \end{array} \}.$$

Les paires critiques de R sont:

- *paire critique de la superposition de la règle 1 sur la règle 2 à l'occurrence ε avec la substitution $\sigma = \{z \longleftarrow e\}$ est $(x.y.x.(y.e))$.*
- *paire critique de la superposition de la règle 1 sur la règle 2 à l'occurrence 1 avec la substitution $\sigma = \{y \longleftarrow e\}$ est $(x.z.x.(e.z))$.*
- *paire critique de la superposition de la règle 2 sur la règle 2 à l'occurrence 1 avec la substitution $\sigma = \{x \longleftarrow x'.x''\}$ est $(x'.(x''.(y.z)).(x'.(x''.y)).z)$.*

L'intérêt des paires critiques vient du théorème suivant qui décide la notion de la confluence locale.

Théorème 1.6 *Un système de réécriture est localement confluent si et seulement si pour toute paire critique (t_1, t_2) de R il existe un terme t' tel que:* $t_1 \overset{*}{\searrow}$ t', $t_2 \overset{*}{\nearrow}$.

Autrement dit, un système de réécriture est localement confluent si chaque paire critique est convergente, c'est à dire ses deux membres ont la même valeur par réécriture.

Ce théorème a tout d'abord été prouvé par D. Knuth et P. Bendix [39] en utilisant l'hypothèse de terminaison du système de réécriture. G. Huet [25] en a donné ensuite une preuve sans cette hypothèse.

Dans le cas où le système de réécriture R est fini et nœthérien, ce résultat donne une procédure de décision pour la confluence de R. Comme il n'existe qu'un nombre fini de paires critiques, il suffit pour chacune d'elles de calculer les formes normales des deux membres et de tester l'égalité.

Définition 1.44 *Un système de réécriture est dit convergent si et seulement si il est nœthérien et confluent.*

Exemple 1.12 *Reprenons l'exemple précédent. R est convergent si nous lui ajoutons la règle suivante:* $3 \quad e.x \rightarrow x$.

Quand un système de réécriture est convergent tout terme admet une forme normale unique.

Soient R un système de réécriture déduit d'un ensemble d'équations A et $=_R$ la fermeture réflexive-symétrique-transitive, $\stackrel{*}{\longleftrightarrow}$, de la relation de réécriture $\longrightarrow$ associée à R. Si R est convergent alors

$$\forall (t_1, t_2) \in T(F, X)^2 \quad t_1 =_R t_2 \implies t_1 \downarrow = t_2 \downarrow .$$

Ainsi l'identité des formes normales fournira un processus de décision de la A-égalité.

Nous allons maintenant présenter la procédure de complétion de Knuth et Bendix qui permet de trouver quand c'est possible, un système de réécriture R convergent déduit d'un ensemble d'axiomes A tel que les congruences $=_A$ et $=_R$ coïncident.

1.3.4 Procédure de complétion

Soient A un ensemble d'axiomes, $\prec$ un ordre de réduction sur les termes, R, initialement vide, l'ensemble qui contiendra les règles déduites de A et P, initialement égal à A, l'ensemble qui contiendra les paires critiques de R.

Pour assurer la propriété de la confluence locale du système de réécriture, la procédure de complétion va introduire les équations de P non convergentes comme nouvelles règles de réécriture, tout en gardant la propriété de terminaison, par rapport à $\prec$, de manière à garantir le calcul des formes normales et la confluence éventuelle du système de réécriture résultant. Il faut alors calculer les paires critiques additionnelles dûes à ces nouvelles règles et itérer ce processus.

Nous allons présenter ci-dessous une version récursive de l'algorithme de complétion de Knuth et Bendix. La correction de cette procédure se trouve dans [39] et [25].

Procédure 1.1

Initialisation:
$P \longleftarrow A$
$R \longleftarrow \emptyset$
$n \longleftarrow 0$
Procédure complétion$(P.R.\prec.n)$ *Signale*$(succès.échec)$

début
  si $P \neq \emptyset$
    alors choisir une paire (p,q) *dans* P
      $p' \longleftarrow p\downarrow$
      $q' \longleftarrow q\downarrow$
      si $p' = q'$
        alors complétion$(P \ominus \{(p,q)\}, R, \prec, n)$
        sinon cas $p' \prec q'$ *alors* $g \longleftarrow q' : d \longleftarrow p'$
                $q' \prec p'$ *alors* $g \longleftarrow p' : d \longleftarrow q'$
                autre-cas *signaler(échec)*
        fcas
      fsi
      $P, R \longleftarrow$ *simplification*$(P.R.g \rightarrow d)$
      complétion$(P \ominus \{(p,q)\}. R \oplus \{n : g \rightarrow d\}, \prec, n+1)$
    sinon si toutes les règles de R *sont marquées*
        alors signaler(succès)
        sinon choisir une règle $m : l \rightarrow r$ *non marquée dans* R
          $P \longleftarrow$ *paire_critique*$(R, m : l \rightarrow r)$
          marquer $l \rightarrow r$
          complétion$(P, R, \prec. m)$
      fsi
  fsi
fin.

- *simplification*: procédure qui simplifie les autres règles et équations en utilisant la nouvelle règle ajoutée, ce qui permet d'obtenir à la fin du processus un système inter-réduit.
- *paire_critique*: procédure qui calcule les paires critiques de la règle de label n avec toutes les règles de label inférieur et les ajoutées aux équations.
- les règles de R sont *étiquetées* par des entiers qui décrivent l'ordre d'apparition des règles dans le système, ce qui permet d'assurer l'hypothèse d'équité sur les règles.
- une règle est *marquée* dès que toutes ses paires critiques avec les règles précédemment

introduites, celles de label inférieur, ont été calculées.

La procédure de complétion peut s'arrêter en échec, en succès ou ne pas terminer et engendrer indéfiniement de nouvelles règles. Dans le premier cas, l'échec est provoqué par l'impossibilité d'orienter une paire critique avec l'ordre $\prec$. Il est possible de l'orienter avec un autre ordre plus "puissant" et l'arrêt en échec ne se produit plus. Dans les deux autres cas, le système de réécriture R retourné, ou théoriquement calculé "à l'infini" est localement confluent, de plus $=_R$ et $=_A$ sont équivalentes.

2
PREUVE PAR RECURRENCE

Le but de ce chapitre est de présenter la méthode adoptée pour prouver la validité d'une équation dans l'algèbre initiale d'une variété équationnelle.

Dans un premier temps, nous allons exposer les différentes approches existantes, ce qui permet de dégager la méthode réalisée et de faire le lien avec les chapitres 3, 4 et 5. Cette méthode est la synthèse des différentes approches.

2.1 Introduction

Dans le cadre des spécifications algébriques, nous ne nous intéressons pas uniquement à prouver équationnellement des théorèmes, mais à prouver des théorèmes qui sont valides pour tout terme clos, c'est à dire des théorèmes valides dans l'algèbre initiale. La méthode classique consiste à utiliser des preuves par récurrence. C'est le problème de la validité dans l'algèbre initiale d'une variété équationnelle.

Le problème de la validité dans l'algèbre initiale d'une variété équationnelle est indécidable en général. Plusieurs approches ont été proposées pour le décider en posant des restrictions plus ou moins pertinentes sur la variété équationnelle. La première direction est fondée sur la preuve par récurrence explicite. L'approche de R. S. Boyer et J. S. Moore [6] est orientée dans ce sens. La deuxième direction regroupe toutes les autres approches qui utilisent la méthode de la réécriture et plus précisement de la procédure de complétion de Knuth et Bendix. Cette méthode que nous appelons *induction sans induction* ou *preuve par consistance* n'utilise pas explicitement de récurrence.

Comme nous l'avons vu dans la troisième partie du chapitre 1, la procédure de complétion fournit une procédure de décision pour la preuve de validité dans la variété équationnelle $M(A)$ engendrée par un ensemble d'équations A, en effet:

- la procédure de complétion essaye de transformer l'ensemble des équations A en un système de réécriture R convergent,
- une équation $t_1 = t_2$ est valide dans $M(A)$ si et seulement si t_1 et t_2 ont les mêmes formes normales par rapport à R.

La procédure de complétion, légérement modifiée, dite procédure de complétion inductive, fournit aussi une procédure de décision pour la preuve de validité dans l'algèbre initiale $I(F, A)$ de la varitété équationnelle $M(A)$ dans certaines conditions:

- la procédure de complétion essaye de transformer l'ensemble des équations A en un système de réécriture R convergent, qui décrive l'algèbre initiale à travers les formes normales closes,
- une équation $t_1 = t_2$ est valide dans $I(F, A)$ si et seulement si $R \cup \{t_1 = t_2\}$ est relativement consistant par rapport à R, c'est à dire ne change pas l'ensemble des formes normales closes.

Exemple 2.1 *Le type abstrait LISTE peut être construit à partir des éléments a et b, de la liste vide $[]$, de l'opérateur de fabrication d'une liste à un seul élément $[-]$ et de la concaténation $@$. Soient donc l'ensemble des constructeurs $C = \{[], a, b, [-], @\}$, l'ensemble des relations entre ces constructeurs est:*

$$\begin{aligned} []@x &= x \\ x@[] &= x \\ (x@y)@z &= x@(y@z). \end{aligned}$$

Un opérateur "flatten" sur LISTE peut être défini comme suit:

$$\begin{aligned} flatten([]) &= [] \\ flatten([x]) &= flatten(x) \\ flatten(a) &= a \\ flatten(b) &= b \\ flatten(a@x) &= a@flatten(x) \\ flatten(b@x) &= b@flatten(x) \\ flatten([x]@y) &= flatten(x)@flatten(y). \end{aligned}$$

Nous pouvons vouloir prouver que $flatten$ est une involution.

$$flatten(flatten(x)) = flatten(x).$$

ou que $flatten$ est un morphisme par rapport à $@$.

$$flatten(x@y) = flatten(x)@flatten(y).$$

Nous pouvons aussi vouloir prouver que $flatten$ n'est pas idempotent.

$$flatten(flatten(x)) = x.$$

2.2 Différentes approches

Nous pouvons classer les différentes approches, basées sur la réécriture, proposées pour tester la validité d'une équation dans l'algèbre initiale d'une variété équationnelle suivant la façon d'axiomatiser l'inégalité:

- explicitement par prédicat d'égalité, ou prédicat d'inégalité, méthode proposée par D. R. Musser puis améliorée par J. A. Goguen, G. Huet et D. Oppen,
- implicitement, avec deux directions différentes:
  - par le fait que deux termes ont des symboles de tête qui sont des constructeurs distincts, méthode fondée sur la consistance et proposée par G. Huet et J. M. Hullot, pui améliorée par D. S. Lankford, N. Dershowitz, J. L. Rémy, E. Paul, L. Puel, H. Kirchner,
  - par le fait que deux termes, clos, ont des formes normales distinctes, méthode basée sur la réductibilité inductive et proposée par J. P. Jouannaud et E. Kounalis puis par D. Kapur, P. Narendran et H. Zhang, pui améliorée par L. Fribourg, W. Küchlin.

2.2.1 Approche basée sur la consistance

L'idée d'utiliser la procédure de complétion pour faire la preuve de validité dans l'algèbre initiale d'une variété équationnelle, a été proposée par D. R. Musser [54]. Il montre que la preuve de validité dans l'algèbre initiale associée à une spécification, se ramène à la preuve de la consistance de cette spécification augmentée de l'équation à prouver. Il impose une restriction à savoir que chaque type de données possède une spécification de l'égalité complètement définie.

- J. A. Goguen [20] suppose l'existence d'une spécification relativement complète de l'égalité et ainsi il donne un rôle essentiel au type booléen. D. R. Musser et J. A. Goguen raisonnent avec une seule sorte d'inconsistance qui est $vrai = faux$.
- G. Huet et D. Oppen [28] imposent que chaque type de données possède une spécification de la négation de l'égalité complètement définie au lieu de l'égalité imposée par D. R. Musser.
- G. Huet et J. M. Hullot [26] améliorent la technique de D. R. Musser en prenant en compte la partition de l'ensemble des opérateurs entre constructeurs et opérateurs définis. Ces derniers doivent satisfaire le principe de définition, ou la complétude relative, c'est à dire tout terme clos est congruent, modulo l'ensemble des axiomes, à un et un seul terme clos sur les constructeurs. L'inconsistance dans cette méthode se ramène alors à la présence de relations entre les constructeurs. La restriction imposée est l'interdiction d'existence de relations entre les constructeurs.
- D. S. Lankford [43] et N. Dershowitz [13] donnent une vue simplifiée de l'approche de G. Huet et J. M. Hullot.

- J. L. Rémy [63] et E. Paul [57] acceptent la présence de relations entre les constructeurs lorsque celles-ci peuvent être associées à un système de réécriture convergent et inductivement complet, ou ω-complet, c'est à dire que l'égalité inductive coïncide avec l'égalité équationnelle. L. Puel [61] accepte aussi les relations entre les constructeurs, mais utilise la récurrence classique pour les preuves des théorèmes entre les constructeurs.

- H. Kirchner [38] unifie ces différentes approches dans le cadre des spécifications hiérarchiques, en prenant en compte la construction structurelle d'une spécification. La construction hiérarchique des spécifications forme une relation de dépendance entre les différents niveaux comme suit: au niveau 0, il y a la spécification de base qui contient les relations entre les constructeurs, si elles existent, et qui possède une procédure de décision des théorèmes inductifs. A chaque niveau, correspond la définition d'un nouvel opérateur utilisant les opérateurs de niveaux inférieurs et les constructeurs. Pour prouver la validité d'un théorème inductif, qui contient un opérateur complètement défini au niveau n par exemple, l'algorithme de complétion procède de la manière suivante: supposons que la procédure engendre une paire critique, lemme du théorème à prouver, qui contient des opérateurs de niveau m, avec m inférieur ou égal à n, alors nous devons prouver sa validité dans la spécification de niveau m. Si m est égal à 0 alors la procédure de décision pour la spécification de base est utilisée, sinon la procédure de complétion est appelée récursivement au niveau m. La procédure de décision pour la spécification de base est: si tout est basé sur le type booléen, la décidabilité de l'égalité inductive revient à vérifier que $vrai \neq faux$, s'il n'y a pas de relations entre les constructeurs, l'égalité inductive coïncide avec l'égalité équationnelle; autrement, si la spécification de base peut être associée à un système de réécriture convergent alors soit l'égalité inductive coïncide avec l'égalité équationnelle si la spécification est ω-complète, soit nous utilisons la méthode par réductibilité inductive.

2.2.2 Approche basée sur la réductibilité inductive

L'approche proposée par J. P. Jouannaud et E. Kounalis [31] repose sur la notion de *réductibilité inductive* d'un terme par rapport à un système de réécriture donné. Elle ne spécifie pas l'ensemble des constructeurs et l'ensemble des opérateurs définis, et elle ne suppose pas la complétude relative de la spécification. Cette approche a été proposé aprés par D. Kapur, P. Narendran et H. Zhang [36] avec une variante d'intérêt faible puis amélioré par L. Fribourg [18] et W. Küchlin [42] en utilisant la confluence close enfin étendu au cas de "*Unfailing Completion*" par L. Bachmair, N. Dershowitz et D. Plaisted [2].

Définition 2.1 *Soient R un système de réécriture et t un terme de $T(F, X)$. Le terme t est dit inductivement réductible, ou quasi-réductible, si et seulement si toute instance close de t est réductible.*

Pour prouver la validité d'une équation $t_1 = t_2$ dans $I(F, A)$, nous procédons de la manière suivante:

- transformer l'ensemble des axiomes A en un système de réécriture convergent R,
- orienter l'équation $t_1 = t_2$ en une règle de réécriture $t_1 \rightarrow t_2$ telle que t_1 soit inductivement réductible,
- si $R \cup \{t_1 \rightarrow t_2\}$ est convergent ou se complète en un système de réécriture convergent tel que tout membre gauche de toute paire critique orientée en une règle de réécriture est inductivement réductible alors $t_1 = t_2$ est valide dans $I(F, A)$.

Le problème majeur de cette méthode est de trouver un algorithme de décidabilité de la réductibilité inductive. Plusieurs procédures ont été proposées pour tester la réductibilité inductive d'un terme pour un système de réécriture linéaire à gauche [40] et [59], dans le cas non linéaire D. Kapur, P. Narendran et H. Zhang [35] ont proposé un algorithme très complexe puis amélioré par D. Plaisted. H. Comon [10] montre sa décidabilité dans le cas général en donnant un algoritheme mais qui reste toujours complexe dans le cas géneral.

La procédure 6.8 décrit la méthode proposée dans [40] pour tester la réductibilité inductive par rapport à un système de réécriture linéaire à gauche.

2.3 Preuve par consistance

2.3.1 Egalité inductive

Soient F un ensemble d'opérateurs et X un ensemble de variables. Nous partitionnons F en deux sous-ensembles C et D disjoints tels que:

- C est l'ensemble de *constructeurs*, ou opérateurs de base,
- D est l'ensemble d'*opérateurs définis*.

Les constructeurs décrivent les objets du type alors que les opérateurs définis sont des opérateurs définis dans le type abstrait de données. Ils s'éliminent dans le calcul des valeurs des opérateurs des objets du type.

Nous supposons dans la suite du problème que C n'est pas vide et que X et F sont disjoints.

Soient A un ensemble d'équations, ou d'axiomes, et $t_1 = t_2$ une autre équation dans $T(F, X)$. Nous allons rappeler la définition des deux congruences sur $T(F, X)$ engendrées par A que nous avons vues dans la première partie du chapitre 1:

- la première est l'égalité équationnelle, notée $=_A$:

Définition 2.2 *$t_1 =_A t_2$ si et seulement si $t_1 = t_2$ est valide dans $M(A)$.*

- la deuxième est l'égalité inductive, notée $=_{ind(A)}$:

Définition 2.3 *$t_1 =_{ind(A)} t_2$ si et seulement si $t_1 = t_2$ est valide dans $I(F, A)$, ce qui est équivalent à: $\forall \sigma \in \Sigma(F) \quad \sigma(t_1) =_A \sigma(t_2)$.*

Convention 2.1
Dans le premier cas, nous disons aussi que $t_1 = t_2$ est un théorème équationnel, ou conséquence équationnelle de A. Nous notons $EQU(A)$ l'ensemble des théorèmes équationnels de $M(A)$ qui représente la théorie équationnelle associée à A.
Dans le deuxième cas, nous disons que $t_1 = t_2$ est un théorème inductif, ou conséquence inductive de A. Nous notons $IND(A)$ l'ensemble des théorèmes inductifs de $I(F, A)$ qui représente la théorie inductive associée à A.

Remarquons que pour toute équation $t_1 = t_2$ tels que t_1 et t_2 sont deux termes non clos et telle que $t_1 =_A t_2$, $t_1 =_{ind(A)} t_2$. La réciproque est fausse, en effet, il existe des théorèmes dont la preuve nécessite une induction sur la structure des termes clos.

Exemple 2.2 *Reprenons l'exemple précédent. $flatten(flatten(x)) = flatten(x)$ est un théorème inductif mais pas un théorème équationnel.*

En général, pour prouver la validité d'une équation dans l'algèbre initiale $I(F, A)$ de la variété équationnelle $M(A)$, nous avons besoin explicitement d'un raisonnement par récurrence, alors qu'un raisonnement équationnel, c'est à dire par une succession de remplacement d'égaux par des égaux à partir des axiomes, est suffisant dans la variété équationnelle $M(A)$.

Dans une spécification structurelle d'un type abstrait de données, l'ensemble des équations A peut être divisé en deux sous-ensembles A_C et A_D disjoints tels que:

- A_C est l'ensemble des relations entre les constructeurs, éventuellement vide,
- A_D est la spécification des opérateurs définis.

Du point de vue de la programmation fonctionnelle, A_C définit la structure de type sur lequel nous travaillons, et A_D la définition des opérateurs c'est à dire un programme fonctionnel.

Lors de l'enrichisssement de la spécification de base (S, C, A_C), nous exigeons que l'ajout de nouveaux opérateurs et de nouveaux axiomes ne détruise pas l'ancienne spécification. De plus nous exigeons que les nouveaux termes créés s'expriment en fonction des anciens. Ce qui s'exprime respectivement par les deux propriétés fondamentales de consistance et de complétude relative de la nouvelle spécification par rapport à la spécification de base. C'est le cas d'un enrichissement dit *protégé* en OBJ [19].

Rappelons la définition de la consistance et la complétude relative d'une spécification.

Définition 2.4 *Un enrichissement (S, F, A) de (S, C, A_C) est relativement consistant par rapport à (S, C, A_C) si et seulement si la restriction de la congruence $=_A$ à $T(C)$ coïncide avec la congruence $=_{A_C}$.*
C'est à dire $\forall (t, t') \in T(C)^2 \quad t =_A t' \implies t =_{A_C} t'$.

Intuitivement, cela signifie que lors de l'enrichissement il n'y a pas d'identification d'éléments distincts de (S, C, A_C).

Définition 2.5 *Un enrichissement (S, F, A) de (S, C, A_C) est relativement complet par rapport à (S, C, A_C) si et seulement si* $\forall t \in T(F) \quad \exists t' \in T(C) \quad t =_A t'$.

Intuitivement, cela signifie que lors de l'enrichissement aucun élément nouveau n'a été ajouté à (S, C, A_C).

Ce qui veut dire que si la spécification (S, F, A) est relativement consistante et complète par rapport à (S, C, A_C) alors $T(F)/_{=_A}$ est isomorphe à $T(C)/_{=_{A_C}}$. C'est à dire que l'algèbre des classes des termes clos modulo A et l'algèbre des classes des termes clos modulo A_C coïncident.

La complétude relative est appelée *complétude suffisante* par J. V. Guttag et J. J. Horning [22] ou *principe de définition*, propriété plus restreinte, par G. Huet et J. M. Hullot [26]. Dans le chapitre 3, nous présenterons une méthode pour prouver la complétude relative d'une spécification par rapport à la spécification de base.

Dans certains cas, nous avons besoin de décider, prouver ou réfuter, un théorème inductif. Comme nous allons le voir plus loin, cela s'avère nécessaire dans la spécification de base (S, C, A_C), représentant la théorie des relations entre les constructeurs. Cela est possible si la théorie inductive coïncide avec la théorie équationnelle, puisque la théorie équationnelle est décidable si elle est associée à un système de réécriture convergent. En d'autres termes, toutes les conséquences inductives des relations entre les constructeurs sont obtenues par un simple raisonnement équationnel. Quant la théorie inductive coïncide avec la théorie équationnelle, la spécification est dite *ω-complète*. La propriété de ω-complétude [23] est appelée *complétude inductive* par E. Paul [57].

Définition 2.6 *Une spécification (S, F, A) est ω-complète si et seulement si tout théorème inductif est un théorème équationnel.*

Dans le chapitre 4, nous présenterons des exemples de spécifications ω-complètes. Ne sachant pas décider la ω-complétude sauf dans des cas restreints, nous supposons qu'elle est testée par l'utilisateur. Dans le cas où la spécification de base est linéaire à gauche, qui est le cas le plus fréquent expérimentalement, nous utilisons plutôt la méthode basée sur la réductibilité inductive, pour prouver la validité dans l'algèbre initiale $I(C, A_C)$ de la variété équationnelle $M(A_C)$. Dans ce cas, nous disposons d'une procédure de décision de la réductibilité inductive. Cette méthode est présentée dans le paragraphe 4 de ce chapitre.

2.3.2 Lemmes fondamentaux

La preuve de la validité dans l'algèbre initiale nécessite en général une induction structurelle sur les termes. Nous n'allons pas faire de la récurrence explicite mais nous allons utiliser le concept de consistance relative d'une spécification que nous avons introduit ci-dessus. La preuve de la validité dans l'algèbre initiale et la consistance relative sont reliées de la manière suivante: l'adjonction d'une équation qui n'est pas un théorème inductif dans une spécification donnée détruit la consistance de cette spécification. Ce qui se traduit par le lemme suivant.

Lemme 2.1 *Soit E une équation. E est valide dans $I(F, A)$ si et seulement si $(S, F, A \cup E)$ est relativement consistante par rapport à (S, F, A).*

Ce résultat peut être raffiné en utilisant la construction structurelle des spécifications. La spécification (S, F, A) étant un enrichissement relativement complet de la spécification (S, C, A_C), alors (S, F, A) et $(S, F, A \cup E)$ sont relativement consistantes par rapport à (S, C, A_C) en même temps lorsque E est valide dans $I(F, A)$. Ainsi, pour prouver la validité de E dans l'algèbre initiale $I(F, A)$, associée à la spécification relativement complète (S, F, A), il suffit de montrer que son adjonction préserve la consistance relative: comme le montre ce lemme qui est la base de la méthode de preuve par consistance [63], [61] et [38].

Lemme 2.2 (Rémy, Kirchner, Paul) *Soit E une équation dont nous voulons tester la validité dans l'algèbre initiale $I(F, A)$. Si la spécification (S, F, A) est relativement complète par rapport à (S, C, A_C), alors les propositions suivantes sont équivalentes:*

- *(i) $(S, F, A \cup E)$ est relativement consistante par rapport à (S, C, A_C).*
- *(ii) (S, F, A) est relativement consistante par rapport à (S, C, A_C) et E est valide dans $I(F, A)$.*

Preuve: Il nous semble utile de détailler ici cette preuve bien qu'elle soit classique.

- $(i) \Longrightarrow (ii)$
  Soient s et t deux termes de $T(C)$ tels que $s =_A t$. Nous avons $s =_A t$ ce qui implique que $s =_{A \cup E} t$ ce qui implique que $s =_{A_C} t$ par (i). donc (S, F, A) est relativement consistante par rapport à (S, C, A_C). Supposons que l'équation E, $s = t$, ne soit pas valide dans $I(F, A)$ donc il existe une substitution close σ telle que $\sigma(s) \neq_A \sigma(t)$. Or puisque $\sigma(s)$ et $\sigma(t)$ sont deux termes clos de $T(F)$ et que (S, F, A) est relativement complète par rapport à (S, C, A_C), alors il existe deux termes s_0 et t_0 de $T(C)$ tels que $\sigma(s) =_A s_0$ et $\sigma(t) =_A t_0$. $\sigma(s) \neq_A \sigma(t)$ ce qui implique que $s_0 \neq_A t_0$ ce qui implique que $s_0 \neq_{A_C} t_0$, cette dernière implication vient du fait que A_C est un sous-ensemble de A. De l'autre côté $s =_E t$ ce qui implique que $\sigma(s) =_{A \cup E} \sigma(t)$ et $\sigma(s) =_{A \cup E} \sigma(t)$ implique que $s_0 =_{A_C} t_0$ par (i). ce qui est contradictoire.

- (ii)$\Longrightarrow$ (i)
  Soient s et t deux termes de $T(C)$. $s =_{A \cup E} t$ implique que $s =_A t$ car l'équation E est valide dans $I(F, A)$. De l'autre côté $s =_A t$ implique que $s =_{A_C} t$ car (S, F, A) est relativement consistante par rapport à (S, C, A_C). Donc $(S, F, A \cup E)$ est relativement consistante par rapport à (S, C, A_C).

Intuitivement cela signifie qu'au lieu de prouver la validité d'une équation E dans l'algèbre initiale $I(F, A)$ d'une spécification (S, F, A) relativement complète par rapport à (S, C, A_C), l'équation E est ajoutée à la spécification (S, F, A) et la consistance relative de cette nouvelle spécification est testée. Si elle est relativement consistante par rapport à (S, C, A_C) alors E est valide, sinon E n'est pas valide, ou bien la spécification (S, F, A) n'est pas relativement consistante par rapport à (S, F, A_C).

La propriété de consistance relative attachée à la structure des spécifications est assurée pratiquement de la manière suivante: si les nouveaux opérateurs sont définis de façon complète par des règles de réécriture et forment un système de réécriture nœthérien, et si la procédure de complétion transforme l'ensemble des règles en un système de réécriture confluent, alors le système de réécriture résultant fournit une spécification relativement consistante et complète par rapport à la spécification de base. La consistance relative est fortement reliée à la confluence close comme nous allons le voir dans le paragraphe 5.

2.3.3 Procédure de complétion inductive

La première version de la procédure de complétion inductive et l'idée de conservation de l'ensemble des instances closes irréductibles d'un système de réécriture ont été étudiées par J. L. Rémy [63] et N. Dershowitz [13].

La procédure de complétion de Knuth et Bendix, présentée dans la troisième partie du chapitre 1, peut être utilisée pour prouver la consistance relative d'une spécification de la manière suivante:
Supposons que (S, F, A) soit une spécification dont nous voulons tester la consistance relative par rapport à une spécification de base (S, C, A_C) ω-complète. Nous appelons R le système de réécriture convergent déduit de A et R_C le système de réécriture convergent déduit de A_C. Soit $=_R$ (resp. $=_{R_C}$) la fermeture réflexive-symétrique-transitive de la relation $\longrightarrow_R$ (resp. $\longrightarrow_{R_C}$) associée à R (resp. R_C). Si (S, F, A) n'est pas relativement consistante par rapport à (S, C, A_C) alors il existe au moins deux termes clos s et t de $T(C)$ tels que $s =_A t$ et $s \neq_{A_C} t$. Nous pouvons supposer que s et t sont en forme normale par rapport à R_C car R_C est convergent. Les deux termes s et t sont différents et au moins un, soit s, n'est pas en forme normale par rapport à R. Ce qui veut dire qu'il existe une règle $g \rightarrow d$ dans R et qui n'est pas dans R_C avec $g \in T(C, X)$ et donc aussi $d \in T(C, X)$. D'un autre côté l'existence d'une telle règle créera une contradiction avec le fait que (S, C, A_C) est ω-complète. En effet puisque (S, C, A_C) est ω-complète, aucune nouvelle règle entre les constructeurs, c'est à dire qui n'est pas déjà dans R_C, ne peut être ajoutée. Donc la spécification (S, F, A) n'est pas relativement consistante. Ainsi nous avons le lemme suivant.

Lemme 2.3 *Soient (S, F, A) une spécification et C un ensemble de constructeurs. Soit R (resp. R_C) le système de réécriture convergent associé à A (resp. A_C). Si (S, C, A_C) est ω-complète alors (S, F, A) est relativement consistante par rapport à (S, C, A_C) si et seulement si aucune nouvelle règle entre les constructeurs n'est engendrée lors de la complétion de A.*

Proposition 2.1 *Soient (S, F, A) une spécification et C un ensemble de constructeurs. Soit R (resp. R_C) le système de réécriture convergent associé à A (resp. A_C). Si tout membre gauche de règle de $R \ominus R_C$ est un terme de $T(F, X) \ominus T(C, X)$ alors (S, F, A) est relativement consistante par rapport à (S, C, A_C).*

Proposition 2.2 *Soient (S, F, A) une spécification et C un ensemble de constructeurs. Soit R (resp. R_C) le système de réécriture associé à A (resp. A_C). Si R est confluent sur les termes clos et $T(C)$ est irréductible par rapport à $R \ominus R_C$ alors (S, F, A) est relativement consistante par rapport à (S, C, A_C).*

Donc si la procédure de complétion donne un système de réécriture confluent et si

l'ordre choisi pour orienter les règles est tel que tous les constructeurs sont plus petits que les autres opérateurs alors la spécification (S, F, A) est relativement consistante par rapport à (S, C, A_C).

Donc pour tester la consistance relative, nous complétons la spécification par la procédure de complétion de Knuth et Bendix et nous regardons si une nouvelle règle dont le membre gauche est un terme de $T(C, X)$ est introduite ou non.

Nous supposons que:

- A_C est transformé en un système de réécriture convergent R_C.
- $R_C \cup A_D$ est transformé aussi en un système de réécriture convergent $R_C \cup R_D$.

Soit E une équation dont nous voulons tester sa validité. Nous exécutons la procédure de complétion initialisée par l'ensemble des règles $R_C \cup R_D$ et l'équation E. Nous supposons que $\prec$ est un ordre partiel bien fondé sur les termes, compatible avec la structure des termes et stable par substitution, avec lequel nous prouvons la terminaison des ensembles successives de règles de réécriture.

La seule différence qui existe entre la procédure de complétion classique et la procédure de complétion inductive réside dans l'étape où une paire de termes, produite dans la procédure soit par une paire critique simplifiée soit par la réduction d'une règle de réécriture, est considérée pour être orientée en une nouvelle règle de réécriture. Pour tester si une nouvelle relation entre les constructeurs est introduite, cette étape est modifiée comme suivante: soit (p, q) une telle paire telle que $p \neq q$:

Procédure 2.1

début
cas $p \in T(C,X)$ *et* $q \in T(C,X)$ *alors* *signaler(invalide)*
$p \in T(C,X)$ *et* $q \notin T(C,X)$ *alors* *si* $p \prec q$
alors introduire la nouvelle règle $q \rightarrow p$
sinon signaler(échec)
fsi
$p \notin T(C,X)$ *et* $q \in T(C,X)$ *alors* *si* $q \prec p$
alors introduire la nouvelle règle $p \rightarrow q$
sinon signaler(échec)
fsi
$p \notin T(C,X)$ *et* $q \notin T(C,X)$ *alors* *cas* $p \prec q$ *alors introduire la nouvelle règle* $q \rightarrow p$
$q \prec p$ *alors introduire la nouvelle règle* $p \rightarrow q$
autre-cas *signaler(échec)*
fcas

fcas
fin.

Théorème 2.1 (Paul) *Soit (S, C, A_C) une spécification ω-complète.*

- *si la procédure s'arrête avec succès ou ne termine pas alors $(S, F, A \cup E)$ est relativement consistante par rapport à (S, C, A_C) et par suite E est un théorème inductif.*
- *si la procédure signale invalide alors $(S, F, A \cup E)$ est relativement inconsistante par rapport à (S, C, A_C) et par suite E n'est pas un théorème inductif.*
- *si la procédure signale échec alors nous ne pouvons pas conclure.*

Preuve: Soit Q le système de réécriture final construit par la procédure. Q peut être fini ou infini et il est de la forme $R_C \cup Q_D$ avec aucune règle de Q_D n'est une relation entre les constructeurs et R_C est resté inchangé durant toute l'exécution de la procédure de complétion.

- La procédure retourne un système de réécriture fini ou infini: soient s et t deux termes de $T(C)$, $s =_{A \cup E} t$ implique que $s =_Q t$ puisque $=_Q$ est équivalent à $=_{A \cup E}$ et de l'autre côté $s =_Q t$ implique que $s =_{R_C} t$ par la forme des règles de Q. Or $s =_{R_C} t$ implique que $s =_{A_C} t$ donc $(S, F, A \cup E)$ est relativement consistante par rapport à (S, C, A_C).
- La procédure retourne invalide: toutes les équations engendrées par la procédure de complétion sont des conséquences de $A \cup E$, donc la procédure s'arrête dès qu'elle engendre une paire $(s, t) \in T(C, X)^2$ telle que $s =_{A \cup E} t$ et $s \neq t$. Puisque s et t sont en formes normales dans $T(C, X)$ cela implique que $s \neq_{R_C} t$. Puisque (S, C, A_C) est ω-complète cela implique que $s \neq_{ind(R_C)} t$. Donc il existe une substitution close σ telle que $\sigma(s) \neq_{R_C} \sigma(t)$ et évidemment $\sigma(s) =_{A \cup E} \sigma(t)$ ce qui montre l'inconsistance relative de $(S, F, A \cup E)$ par rapport à (S, C, A_C).

Remarque 2.1 *Habituellement, l'ordre $\prec$ utilisé pour prouver la terminaison ne met jamais un terme de $T(F, X) \ominus T(C, X)$ inférieur à un terme de $T(C, X)$ en choisissant une précédence pour les constructeurs C, donc les deux premières signalisations d'échec ne peuvent jamais avoir lieu.*

Exemple 2.3 *Axiomatisation de la structure d'arbre binaire.*
Soient $C = \{[], @\}$, $D = \{Append\}$ et R le système de réécriture suivant:

$$R = \{ \begin{array}{lll} 1. & Append([], x) & \rightarrow x \\ 2. & Append((x@y), z) & \rightarrow x@Append(y, z) \end{array} \}.$$

- Append est relativement complet,

- R est convergent et ω-complet,

Nous voulons prouver l'associativité de Append:

E. $Append(Append(x', y'), z') = Append(x', Append(y', z'))$

- après orientation de cette équation, nous l'ajoutons au système de réécriture R puis nous le complètons:

3. $Append(Append(x', y'), z') \rightarrow Append(x', Append(y', z'))$

+ superposition de la règle 1 sur la règle 3 à l'occurrence 1 avec la substitution $\sigma_1 = \{x' \longleftarrow [], x \longleftarrow y'\}$*:*

$$\begin{array}{rl} & Append(Append([], y'), z') \\ 3\swarrow & \\ Append([], Append(y', z')) & \downarrow 1 \\ 1\searrow & \\ & Append(y', z') \end{array}$$

+ superposition de la règle 2 sur la règle 3 à l'occurrence 1 avec la substitution $\sigma_2 = \{x' \longleftarrow x''@y'', x \longleftarrow y'\}$*:*

$$\begin{array}{rlll} & Append(Append(x''@y'', y'), z') & & \\ & & \searrow 2 & \\ 3\swarrow & & & Append(x''@Append(y'', y'), z') \\ Append(x''@y'', Append(y', z')) & & & \downarrow 2 \\ & & & x''@Append(Append(y'', y'), z') \\ 2\searrow & & & \swarrow 3 \\ & x''@Append(y'', Append(y', z')) & & \end{array}$$

Ces deux paires critiques sont convergentes, ainsi que la paire critique issue de la superposition de la règle 3 sur elle même à l'occurrence 1. Le système de réécriture résultant étant convergent alors l'équation E est un théorème inductif. La superposition des deux règles 1 et 2 sur la règle 3 fait apparaître une récurrence implicite par les substitutions σ_1 et σ_2 sur $Append(Append(x', y'), z')$ pour: $x' = []$ et $x' = x''@y''$.

Exemple 2.4 *Reprenons l'exemple 2.1.*

$$\begin{array}{lcll} C & = & \{ & [], a, b, [-], @ \quad \}, \\ R_C & = & \{ & [] @ x \rightarrow x \\ & & & x @ [] \rightarrow x \\ & & & (x @ y) @ z \rightarrow x @ (y @ z) \quad \}, \\ D & = & \{ & flatten \quad \}, \\ R_D & = & \{ & flatten([]) \rightarrow [] \\ & & & flatten([x]) \rightarrow flatten(x) \\ & & & flatten(a) \rightarrow a \\ & & & flatten(b) \rightarrow b \\ & & & flatten(a@x) \rightarrow a@flatten(x) \\ & & & flatten(b@x) \rightarrow b@flatten(x) \\ & & & flatten([x]@y) \rightarrow flatten(x)@flatten(y) \quad \}. \end{array}$$

D est relativement complet comme nous allons le démontrer dans le chapitre 3.

L'équation $flatten(flatten(x)) = flatten(x)$ est un théorème inductif mais pour le prouver il faut d'abord démontrer que $flatten(x@y) = flatten(x)@flatten(y)$ est un théorème inductif sinon la procédure de complétion diverge.

L'équation $flatten(flatten(x)) = x$ n'est pas un théorème inductif. En effet si nous l'ajoutons dans R_D l'équation $[x] = x$, qui est une nouvelle relation entre les constructeurs, est engendrée par la procédure de complétion. Si nous supposons que R_C est ω-complet alors R_C est relativement inconsistant.

2.4 Preuve par réductibilité inductive

2.4.1 Réductibilité inductive

Rappelons qu'un terme t est inductivement réductible par rapport à un système de réécriture R si et seulement si toute instance close de t est réductible par rapport à R. Le concept de réductibilité inductive est très important du fait que si nous ajoutons un ensemble d'équations, orientable en un ensemble de règles de réécriture dont le membre gauche de chacune est inductivement réductible, à un système de réécriture nœthérien alors l'ensemble des formes normales closes du système de réécriture est préservé, comme le montre le lemme suivant.

Lemme 2.4 *Soient R un système de réécriture nœthérien et R' un autre système de réécriture tel que le membre gauche de chacune de ses règles soit inductivement réductible par rapport à R. Alors tout terme clos est en forme normale par rapport à R si et seulement si il est en forme normale par rapport à $R \cup R'$.*

Ce qui permet de déduire le théorème suivant [31] qui est la base de cette méthode.

Théorème 2.2 *Soient R le système de réécriture convergent associé à un ensemble d'équations A et R' le système de réécriture associé à E et tel que le membre gauche de la règle de R' est R-inductivement réductible. Si $R \cup R'$ est convergent alors E est valide dans $I(F, A)$.*

Ainsi, puisque la réductibilité inductive est décidable dans le cas d'un système de réécriture linéaire à gauche, nous avons une procédure pour prouver la validité d'une équation E dans l'algèbre initiale $I(F, A)$, dans le cas où A peut être associé à un système de réécriture linéaire à gauche. C'est cette méthode que nous utiliserons pour prouver des théorèmes inductifs dans l'algèbre initiale $I(C, A_C)$.

2.4.2 Procédure de complétion inductive

La seule différence qui existe avec la procédure de complétion de Knuth et Bendix standard apparait au moment où une nouvelle règle est ajoutée. Une règle n'est introduite que lorsque son membre gauche est inductivement réductible par rapport au système de réécriture courant, autrement la procédure s'arrête avec réfutation de l'équation dont elle teste la validité. La correction de cette procédure est étudiée dans [40].

Exemple 2.5 *Reprenons l'exemple 2.4.*

L'équation $flatten(flatten(x)) = flatten(x)$ *est un théorème inductif mais pour le prouver il faut d'abord démontrer que* $flatten(x@y) = flatten(x)@flatten(y)$ *est un théorème inductif sinon la procédure de complétion diverge.*

L'équation $flatten(flatten(x)) = x$ *n'est pas un théorème inductif. En effet si nous l'ajoutons dans* R_D *l'équation* $[x] = x$*, qui est une nouvelle relation entre les constructeurs. est engendrée par la procédure de complétion.* $[x]$ *n'est pas inductivement réductible.*

La procédure 6.2 décrit cette méthode pour tester la validitée d'une équation dans l'algèbre initiale d'une variété équationnelle.

2.5 Combinaison des deux méthodes

Le problème qui se pose dans la méthode de preuve par consistance c'est qu'il faut supposer que le système de réécriture de base est ω-complet. A l'état actuel des choses. cette propriété ne peut pas être vérifiée d'une manière automatique et efficace, comme nous allons le voir dans le chapitre 4. La méthode de preuve par réductibilité inductive pose aussi un problème de trouver une procédure efficace de test de la réductibilité inductive. La méthode que nous avons adopté est la combinaison de ces deux méthodes. Cette méthode reppose sur la première méthode, qui est très efficace, sans supposer que le système de réécriture de base est ω-complet, mais de tester si les nouvelles relations entre les constructeurs sont des théorèmes inductives, dans la spécification de base, par la deuxième méthode. Ainsi le test de la réductibilité inductive est restreint au système de réécriture de base. Une deuxième amélioration a été apporté à la deuxième méthode en ne testant la réductibilité inductive des membres gauches des règles engendrées par complétion qu'à la fin de la complétion, ce qui permet d'éviter ce test, qui est très coûteux, pour des règles éliminables par d'autres règles plus générales aprés.

Théorème 2.3 *Soient* C *un ensemble de constructeurs,* R *un système de réécriture convergent et relativement complet par rapport à* R_C*,* E *une équation dont nous voulons tester sa validité dans l'algèbre initiale associée et* R' *le système de réécriture obtenu par complétion de* $R \cup E$*.*

E est un théorème inductive si et seulement si le membre gauche de toute règle de $R'_C \ominus R_C$ est inductivement réductible par rapport à R_C.

La procédure 6.1 décrit cette méthode pour tester la validitée d'une équation dans l'algèbre initiale d'une variété équationnelle.

2.6 Confluence close

Pour prouver la validité d'une équation E dans l'algèbre initiale de la variété équationnelle associée à un système de réécriture R, nous avons supposé, en plus de la terminaison de R, que R est confluent. Dans les deux méthodes précédentes, nous ajoutons E à R puis nous complétons le système résutant pour avoir un système de réécriture confluent alors que la confluence close, c'est à dire la confluence sur les termes clos, suffit. L. Fribourg [18] propose une condition suffisante de cette propriété par une sélection linéaire des superpositions à une seule occurrence donnée:

- superpositions uniquement des règles de R sur E ou sur les paires critiques issues de E, et pas de superpositions d'une paire critique sur une autre paire critique ou sur une règle de R.
- superpositions appliquées à la seule occurrence sélectionnée.

Nous avons aussi la proposition suivante qui est la quasi-réciproque de la proposition 2.2.

Proposition 2.3 *Soient (S, F, A) une spécification et C un ensemble de constructeurs. Soit R (resp. R_C) le système de réécriture associé à A (resp. A_C). Si R_C est confluent clos et $\longrightarrow_R$ est relativement complète par rapport à (S, C, A_C) et (S, F, A) est relativement consistante par rapport à (S, C, A_C) alors R est confluent clos.*

Cette limitation du calcul des superpositions permet, en plus du gain du temps, de prouver la validité dans des cas où la procédure de complétion inductive diverge en engendrant infiniment des paires critiques comme dans le cas de l'exemple suivant.

Exemple 2.6 *Soit R le système de réécriture, spécifiant les entiers naturels, suivant:*

$$R = \{ \begin{array}{lll} 1. & 0 + x & \rightarrow x \\ 2. & s(x) + y & \rightarrow s(x + y) \end{array} \}.$$

L'ajout de la règle 3. $x + (y + z) \rightarrow (x + y) + z$ *fait diverger la procédure de complétion en engendrant infiniment des paires critiques de la forme $((x + s^n(y)) + z, x + s^n(y + z))$ par superposition de la règle* 2, *et les paires critiques issues de cette superposition, sur la règle* 3 *à l'occurrence* 2.

Convention 2.2
Soient R un système de réécriture, t un terme de $T(F,X)$ et $u \in G(t)$. nous posons:
$\Sigma(F)\downarrow_R$*: l'ensemble des substitutions closes R-irréductibles.*
$\Sigma_u(t) = \{\sigma \mid \sigma \in \Sigma(F)\downarrow_R \quad \exists g \to d \in R \quad \sigma(t/u) = \sigma(g)\}$,
$I(t)\downarrow_R$*: l'ensemble des instances closes en formes normales par rapport à R de t.*

Définition 2.7 *Une occurrence close u d'un terme t_1 est complète si et seulement si $\forall \sigma \in \Sigma(F)\downarrow_R \quad \sigma(t_1)$ est réductible à l'occurrence u.*
Dans ce cas, nous disons que l'équation $t_1 = t_2$ est complètement superposable.

Proposition 2.4 *Une occurrence u de t est complète si et seulement si*
$\forall f \in D \ \{\sigma(f(y_1,\ldots,y_n)) \mid \sigma \in \Sigma_u(t)\} \supseteq I(f(x_1,\ldots,x_n))\downarrow_R$.

L'occurrence où se fait les superpositions doit être complète. Elle correspond au terme d'induction [6] sélectionné dans les méthodes d'induction classiques [8], [6] et [1]. La méthode de [31] fait plus qu'une induction à la fois alors qu'une suffit.

Exemple 2.7 *Reprenons l'exemple 2.6. Les occurrences ε et 2 de la règle 3 sont complètes. Pour la règle 3'. $(x+y)+z \to x+(y+z)$, l'occurrence 1 est complète par contre l'occurrence ε n'est pas complète.*

Si R est relativement complet alors tout terme contenant un opérateur défini f est inductivement réductible en utilisant l'ensemble des occurrences du sous-terme dont la racine est f. Si R est relativement complet alors toute équation, qui contient un opérateur défini, est complètement superposable. Si $t_1 = t_2$ est complètement superposable alors t_1 est inductivement réductible. La réciproque de cette dernière implication est fausse comme nous pouvons le voir dans l'exemple suivant [31]:

Exemple 2.8 *Soit R le système de réécriture, spécifiant les entiers modulo 2. suivant:*
$R = \{s(s(0)) \to 0\}$.
Soient $\sigma_1 = \{x \longleftarrow 0\}$ et $\sigma_2 = \{x \longleftarrow s(0)\}$ les deux substitutions closes irréductibles. $\sigma_1(s(s(x)))$ est réductible à l'occurrence ε alors que $\sigma_2(s(s(x)))$ est réductible à l'occurrence 1. $s(s(x))$ est inductivement réductible mais $s(s(x)) = x$ n'est pas complètement superposable.

W. Küchlin [42] élimine cette limitation en étendant la notion de superposition à une seule occurrence complète à la notion de superposition à un ensemble complètement inductif d'occurrences. Par exemple dans l'exemple 2.8 $\{\varepsilon, 1\}$ est complètement inductive. Si t/u est R-inductivement réductible alors $G(t/u)$ est un ensemble inductivement complet d'occurrences mais n'est pas toujours minimal.

En plus du problème de la détermination d'une occurrence complète, ou d'un ensemble d'occurrences, il existe des cas où il y a plusieurs occurrences complètes mais seulement certaines peuvent mener à bien le processus. Nous les appelons les "bonnes" occurrences complètes, et elles correspondent aux "bons" termes d'induction [6] dans les méthodes d'induction classiques. C'est le cas de l'exemple 2.6 où ε est une bonne occurrence complète par contre 2 est une occurrence complète "mauvaise".

3
COMPLETUDE RELATIVE

Nous allons présenter dans ce chapitre une procédure qui teste la complétude relative d'une spécification par rapport à la spécification de base.

Cette procédure, basée sur l'unification, est une extension de la méthode proposée par J. J. Thiel [67] pour permettre la présence de relations entre les constructeurs, et/ou les spécifications non linéaires.

Cette méthode trouve son importance dans le fait qu'elle permet de préciser, sous la forme la plus générale, tous les termes où un opérateur n'est pas défini. De la même façon elle donne, sous la forme la plus générale, tous les termes où un opérateur est défini plus d'une fois.

3.1 Introduction

La propriété de la complétude relative d'une spécification est toujours souhaitée dans la construction structurée des spécifications. Elle permet d'assurer la cohérence d'une spécification par rapport à la spécification de base. Nous l'exigerons aussi dans la programmation fonctionnelle et dans les types abstraits de données. Elle est aussi nécessaire pour prouver des théorèmes inductifs par la méthode fondée sur la consistance, comme nous l'avons vu dans le chapitre 2.

La preuve de la complétude relative d'une spécification donnée par rapport à la spécification de base est indécidable en général. T. Nipkow et G. Weikum [56] montrent sa décidabilité dans le cas des systèmes de réécriture linéaires à gauche. Plusieurs méthodes, basées en général sur la réécriture, ont été proposées, en donnant des conditions suffisantes, pour la tester. Toutes ces méthodes imposent des restrictions plus ou moins pertinentes sur la spécification donnée et sur la spécification de base, de plus la plupart d'entre elles sont inefficaces. Citons celles de G. Huet et J. M. Hullot [26], M. Bidoit [4], N. Dershowitz [13], J. J. Thiel [67], E. Kounalis [40], H. Comon [9] [10] et D. Kapur, P. Narendran et H. Zhang [35].

- Le moyen le plus intuitif et le plus simple de satisfaire la propriété de complétu-

de relative d'une spécification consiste à définir chaque opérateur de l'ensemble des opérateurs définis D par *récurrence structurelle* sur les arguments déjà construits à partir des opérateurs de l'ensemble des constructeurs C. Une telle définition est appelée *définition complète* de D. C'est la méthode proposée par G. Huet et J. M. Hullot puis par M. Bidoit pour tester la complétude relative en se restreignant à des spécifications dont le membre gauche est linéaire, sa racine est l'unique opérateur défini et les arguments sont des termes de $T(C, X)$.

- La méthode proposée par N. Dershowitz est basée sur la notion *d'ensemble de motifs* qui est l'ensemble des termes dont la racine est l'opérateur dont nous testons la complétude de définition et les arguments sont des termes de $T(C, X)$ dont la profondeur est strictement inférieure à la profondeur maximum des membres gauches des règles de la spécification. Cette méthode nécessite une expression exhaustive de l'ensemble des motifs.

  Exemple 3.1 *Soient C l'ensemble des constructeurs $C = \{0, s\}$ et R le système de réécriture suivant qui défini l'opérateur $+$:*
  $$R = \{ \begin{array}{ll} 0+x & \rightarrow 0 \\ s(x)+y & \rightarrow s(x+y) \end{array} \}.$$
  L'ensemble des motifs de $+$ est $\{0+0, 0+s(y), s(x)+0, s(x)+s(y)\}$.

- E. Kounalis améliore la méthode de N. Dershowitz en se basant sur la notion *d'arbre de motifs* au lieu d'ensemble de motifs, ce qui rend cette méthode plus efficace que celle de N. Dershowitz en éliminant certains motifs inutiles.

  Exemple 3.2 *Reprenons l'exemple précédent.*
  L'arbre de motifs de $+$ est $\{0+y, s(x)+y\}$.

- La méthode proposée par J. J. Thiel est basée sur la notion de *recouvrement.* Un opérateur f est complètement défini si et seulement si l'ensemble des membres gauches des règles de la spécification dont la racine est f couvre $\{f(x_1, \ldots, x_n)\}$, c'est à dire il contient toutes les instances closes de $\{f(x_1, \ldots, x_n)\}$. Le test de recouvrement est basé sur l'unification, ce qui rend cette méthode très efficace. C'est cette méthode que nous allons exposer dans la suite de ce chapitre en l'étendant dans le cas des systèmes de réécriture non linéaires à gauche et dans le cas où il y a des relations entre les constructeurs.

Exemple 3.3 *Soit $C = \{0, s\}$. L'opérateur $+$, sur les entiers, sera défini si nous connaissons la valeur de $0+x$ et $s(x)+y$. Une définition complète de $+$ sera donnée donc par deux règles ayant respectivement pour membre gauche, chacun de ces deux termes.*

La plupart de ces méthodes citées ci-dessus sont basées sur la propriété suivante, équivalente à la complétude relative: tout terme dont la racine est un opérateur de D et les arguments appartiennent à $T(C)$ est R-réductible. Ce qui est équivalent aussi à: tout terme de $T(F,X) \ominus T(C,X)$ est R-inductivement réductible, comme nous allons le voir dans le paragraphe suivant.

3.2 Convertibilité relative

Soit (S,F,A) une spécification dont nous voulons tester la complétude relative par rapport à la spécification de base (S,C,A_C). Nous partitionnons l'ensemble des opérateurs F en un ensemble d'opérateurs de base, ou de constructeurs C et un ensemble d'opérateurs définis D, puis nous partitionnons l'ensemble des axiomes A en A_C et A_D qui sont respectivement l'ensemble des axiomes dont la racine du membre gauche est un constructeur et l'ensemble des axiomes dont la racine du membre gauche est un opérateur défini.

Nous supposons dans la suite que tout axiome dont la racine est un constructeur a ses arguments dans $T(C,X)$. Nous supposons aussi dans toute la suite que l'ensemble des axiomes A d'une spécification (S,F,A) est associé à un système de réécriture convergent R obtenu à partir de A par la procédure de complétion. Nous utilisons la notation (S,F,R) au lieu de (S,F,A). Nous partitionnons l'ensemble des règles R en R_C et R_D. de la même façon que A, et nous supposons que R est nœthérien et R_C est convergent.

Convention 3.1
Soit f un opérateur défini et X un ensemble dénombrable de variables. nous posons:
$T_f(C,X)$: l'ensemble des termes dont la racine est f et les arguments sont dans $T(C,X)$,
$T_f(C)$: l'ensemble des termes dont la racine est f et les arguments sont dans $T(C)$,
$MG(f,R)$: l'ensemble des membres gauches des règles de R dont la racine est f.
Soient t un terme de $T_f(C,X)$ et T un sous-ensemble de $T_f(C,X)$, nous posons:
$I(t)$: l'ensemble des instances closes dans $T(C)$ de t,
$I(T) = \cup_{t \in T} I(t)$,
$x_1, \ldots, x_n$ est une suite de variables distinctes de X.
$\ominus$ et $\oplus$ sont respectivement la différence ensembliste et la réunion disjointe.

Rappelons la définition de la complétude relative d'une spécification (S,F,R) par rapport à sa spécification de base (S,C,R_C).

Définition 3.1 *(S,F,R) est relativement complète par rapport à (S,C,R_C) si et seulement si $\forall t \in T(F) \quad \exists t' \in T(C) \quad t =_R t'$.*

Evidemment (S,F,R) est relativement complète par rapport à (S,C,R_C) si et seulement si la forme normale par rapport à R de tout terme clos de $T(F)$ appartient à $T(C)$.

Cela peut être testé en utilisant le nouveau concept de *convertibilité relative* suivant.

Définition 3.2 *Soient f un opérateur défini de D et (S, F, R) une spécification. f est relativerment convertible par rapport à C si et seulement si tout terme de $T_f(C)$ est R-réductible.*

Autrement dit, f est relativement convertible par rapport à C si et seulement si pour toute n-uplet de formes normales $(s_1, \ldots, s_n)$ dans $T(C)$ où n est l'arité de f. $f(s_1, \ldots, s_n)$ est R-réductible, ou toute instance close de $f(x_1, \ldots, x_n)$, $I(f(x_1, \ldots, x_n))$, est R-réductible, c'est à dire que f est complètement défini sur l'ensemble des constructeurs.

La preuve de la complétude relative d'une spécification se ramène donc à la preuve de la convertibilité relative de tous les opérateurs définis, c'est ce qui exprime le lemme suivant:

Lemme 3.1 *La spécification (S, F, R) est relativement complète par rapport à (S, C, R_C) si et seulement si tout opérateur défini est relativement convertible par rapport à C.*

Preuve:

- Implication directe: évidente.
- Réciproque: si (S, F, R) n'est pas relativement complète alors il existe un terme t de $T(F)$ tel que sa forme normale par rapport à R $t\downarrow$ n'appartient pas à $T(C)$. Donc il existe un sous-terme t' de $t\downarrow$ tel que t' appartient à $T_f(C)$ et t' n'est pas R-réductible ce qui est contradictoire avec la convertibilité relative de f par rapport à C.

Théorème 3.1 *f est relativement convertible par rapport à C si et seulement si tout terme de $I(f(x_1, \ldots, x_n)) \ominus I(MG(f, R))$ est R_C-réductible.*

Preuve:

- Implication directe: évidente.
- Réciproque: $I(f(x_1, \ldots, x_n))$ étant l'ensemble des termes clos dont la racine est f, si tous les termes qui ne sont pas dans $I(MG(f, R))$, c'est à dire non R-réductible par une règle de racine f, ou non R_D-réductible, sont R_C-réductible alors tous les termes clos de racine f sont R-réductibles et par suite f est relativement convertible par rapport à C.

Exemple 3.4 *Soient $C = \{0, o, s\}$ où o est l'opérateur opposé sur les entiers. $D = \{+\}$ et R le système de réécriture suivant:*

$$R = \{ \begin{array}{ll} o(0) & \rightarrow 0 \\ o(o(x)) & \rightarrow x \\ s(o(s(x))) & \rightarrow o(x) \\ 0+x & \rightarrow x \\ s(x)+y & \rightarrow s(x+y) \\ o(s(x))+y & \rightarrow o(s(x+o(y))) \end{array} \}.$$

$+$ est relativement convertible par rapport à C, en effet:

$I(x_1+x_2) \ominus I(MG(+,R)) = \{t \mid t \in I(o(0)+y) \vee t \in I(o(o(x))+y)\}$ ne contient que les termes R_C-réductibles.

Ainsi, pour prouver la convertibilité relative d'un opérateur défini f par rapport à un ensemble de constructeurs C, il faut prouver que tout terme de $I(f(x_1,\ldots,x_n))$ est soit une instance du membre gauche d'une règle de R, dont la racine du membre gauche est f, soit R_C-réductible.

Pour utiliser ce théorème en pratique, nous introduisons le concept de *recouvrement.* Ce concept est introduit par J. J. Thiel [67] puis présenté indépendemment par J. L. Lassez et K. Mariott dans le contexte de la programmation logique et des automates d'apprentissage [44].

Remarque 3.1 *Soient R un système de réécriture de termes et E un ensemble d'équations. Nous notons $R \cup E$ le système de réécriture équationnel associé. Soit f un opérateur de D, f est relativement convertible par rapport à C si et seulement si tout terme de $I(f(x_1,\ldots,x_n)) \ominus I(MG(f,R))$ est $R_C \cup E$-réductible, c'est à dire R_C-réductible modulo E.*

3.3 Recouvrement

3.3.1 Définition

Définition 3.3 *Soient T et S deux sous-ensembles de $T(F,X)$. Nous disons que T couvre S si et seulement si $I(S)$ est un sous-ensemble de $I(T)$, c'est à dire que toute instance close d'un terme de S est une instance close d'un terme de T.*

Exemple 3.5 *Soient $C = \{0,s,p\}$ et $D = \{+\}$.*
$\{s(x)+y, p(x)+y\}$ ne couvre pas $\{x+y\}$ alors que $\{0+x, s(x)+y, p(x)+y\}$ couvre $\{x+y\}$.

Le but du concept de recouvrement est d'appliquer le théorème 3.1 pour trouver la partie K de $I(f(x_1,\ldots,x_n))$ qui n'est pas couverte par $I(MG(f,R))$ et pour vérifier ensuite, que les termes de K sont R_C-réductibles.

En pratique, nous ne pouvons pas utiliser directement le concept de recouvrement car il introduit la manipulation des ensembles infinis de termes, cependant nous allons introduire un autre concept qui est le *complément d'un terme.*

3.3.2 Complément d'un terme

Définition 3.4 *Soit t un terme de $T(C,X)$. Nous appelons complément de t, tout ensemble fini $A(t,C)$ de termes de $T(C)$ tel que $T(C) = I(t) \oplus I(A(t,C))$.*

En particulier, si t est une variable de X, $A(t,C)$ est vide puisque $I(t) = T(C)$.

Nous allons présenter maintenant une proposition qui donne une définition constructive du concept de complément d'un terme linéaire de $T(C,X)$, c'est à dire un terme qui a au plus une occurrence de chaque variable.

Proposition 3.1 *Soient t un terme linéaire de $T(C,X)$ tel que $t = c_j(t_1,\ldots,t_{n_j})$, $C = \{c_1,\ldots,c_m\}$ où n_h est l'arité de c_h et*

$$\begin{array}{rcl} A(t,C) & = & \{c_i(x_1,\ldots,x_{n_i}) \mid x_1,\ldots,x_{n_i} \in X \wedge 1 \leq i \leq m \wedge i \neq j\} \\ & \cup_{1\leq k\leq n_j} & \{c_j(t_1,\ldots,t_{k-1},t',x_{k+1},\ldots,x_{n_j}) \mid x_{k+1},\ldots,x_{n_j} \in X \wedge t' \in A(t_k,C)\} \end{array}$$

avec toutes les variables de $t_1,\ldots,t_{k-1},t',x_{k+1},\ldots,x_{n_j}$ sont distinctes,

alors $A(t,C)$ est un complément de t.

Preuve:

- $I(t) \cap I(A(t,C)) = \emptyset$ est évident,
- $I(t) \cup I(A(t,C)) = T(C)$ en effet: soit s un terme de $T(C)$,

si la racine de s est c_i avec $i \neq j$
alors $s \in I(A(t,C))$ car s est une instance de $c_i(x_1,\ldots,x_{n_i})$
sinon soit $s = c_j(s_1,\ldots,s_{n_j})$
si il existe σ telle que $\sigma(t) = s$
alors $s \in I(t)$
sinon nous construirons σ de la façon suivante:
soit $k \in [1,n_j]$ tel que
$\forall h < k \quad \exists \sigma_h \in \Sigma(C,X) \quad \sigma_h(t_h) = s_h$
et $\forall \rho \in \Sigma(C,X) \quad \rho(i_k) \neq s_k$
Donc $\exists \sigma_k \in \Sigma(C,X) \quad \exists t' \in A(t_k,C) \quad \sigma_k(t') = s_k$
or puisque t est linéaire, σ est définie comme suite:
cas $x \in V(t_h)$ avec $h < k$ alors $\sigma(x) = \sigma_h(x)$
$x \in V(t')$ alors $\sigma(x) = \sigma_k(x)$
$x = x_h$ avec $h > k$ alors $\sigma(x) = s_h$
et nous avons: $\sigma(c_j(t_1,\ldots,t_{k-1},t',x_{k+1},\ldots,x_{n_j})) = s$

et par suite $s \in I(A(t, C))$.

Exemple 3.6 *Soit* $C = \{0, s, p\}$,
$A(0, C) = \{s(x), p(x)\}$, $A(s(x), C) = \{0, s(0), s(p(x)), p(x)\}$.

Nous allons maintenant présenter un nouveau concept de *complémentarité d'une substitution*, qui permet de calculer le complémentaire de l'ensemble des instances closes d'une instance d'un terme.

3.3.3 Complément d'une substitution

Définition 3.5 *Une substitution* σ *est linéaire si et seulement si*

- $\forall x \in D(\sigma)$ $\sigma(x)$ *est linéaire,*
  et
- $\forall (x, y) \in D(\sigma)^2$ $x \neq y \Longrightarrow V(\sigma(x)) \cap V(\sigma(y)) = \emptyset$.

Autrement dit, σ transforme tout terme linéaire en un terme linéaire.

Définition 3.6 *Soit* σ *une substitution linéaire de* $\Sigma(C, X)$. *Nous appelons complément de* σ, *l'ensemble* $B(\sigma, C)$ *de toutes les substitutions* ρ *telles que:*

- $\rho \neq \sigma$,
- $D(\rho) = D(\sigma)$,
- $\forall x \in D(\rho)$ $\rho(x) \in A(\sigma(x), C) \vee \rho(x) = \sigma(x)$.

Exemple 3.7 *Soient* $C = \{0, s, p\}$ *et* $\sigma = \{x \longleftarrow 0, y \longleftarrow s(v), z \longleftarrow u\}$.

$$
\begin{array}{rcl}
B(\sigma, C) = \{ & \{ & x \longleftarrow s(x') \ , \ y \longleftarrow s(v) \ , \ z \longleftarrow u \ \}: \\
& \{ & x \longleftarrow p(x') \ , \ y \longleftarrow s(v) \ , \ z \longleftarrow u \ \}: \\
& \{ & x \longleftarrow 0 \ , \ y \longleftarrow 0 \ , \ z \longleftarrow u \ \}: \\
& \{ & x \longleftarrow s(x') \ , \ y \longleftarrow 0 \ , \ z \longleftarrow u \ \}; \\
& \{ & x \longleftarrow p(x') \ , \ y \longleftarrow 0 \ , \ z \longleftarrow u \ \}; \\
& \{ & x \longleftarrow 0 \ , \ y \longleftarrow p(y') \ , \ z \longleftarrow u \ \}: \\
& \{ & x \longleftarrow s(x') \ , \ y \longleftarrow p(y') \ , \ z \longleftarrow u \ \}; \\
& \{ & x \longleftarrow p(x') \ , \ y \longleftarrow p(y') \ , \ z \longleftarrow u \ \} \ \}.
\end{array}
$$

Le concept du complément d'une substitution permet de calculer facilement $I(t) \ominus I(\sigma(t))$ pour $t \in T(C, X)$.

La procédure 6.6 (resp. la procédure 6.7) décrit cette méthode pour calculer le complémentaire d'une substitution (resp. d'un terme).

Proposition 3.2 *Soient t un terme de $T_f(C,X)$ et σ une substitution linéaire.*

- $I(t) = \cup_{\rho \in B(\sigma,C)} I(\rho(t)) \cup I(\sigma(t))$,
  et
- $\forall \rho \in B(\sigma,C) \quad I(\rho(t)) \cap I(\sigma(t)) = \emptyset$.

Preuve: Par définition du complément d'une substitution.

Remarque 3.2 *La restriction posée concerne la linéarité de la substitution σ et non pas la linéarité du terme t ou du terme $\sigma(t)$. En effet si par exemple $C = \{0, s\}$ et $\sigma = \{y \longleftarrow s(x)\}$ alors $I(y+y) \ominus I(\sigma(y+y)) = I(y+y) \ominus I(s(x)+s(x)) = I(0+0)$. Cependant nous pouvons étendre cette méthode aux cas où le système de réécriture n'est pas linéaire à gauche.*

Le théorème suivant donne une condition suffisante pour tester le concept de recouvrement.

Théorème 3.2 *Soient T et S deux sous-ensembles de $T(F,X)$. T couvre S si l'une des deux conditions suivantes est satisfaite:*

1. *S est vide,*
2. *il existe deux termes $t \in T$ et $s \in S$ unifiables par une substitution linéaire σ et tels que: $(T \ominus \{t\}) \cup \{\rho(t) \mid \rho \in B(\sigma,C)\}$ couvre $(S \ominus \{s\}) \cup \{\rho(s) \mid \rho \in B(\sigma,C)\}$.*

Preuve:

1. Si S est vide, c'est évident.
2. Soit s un terme de S. Puisque T couvre S il existe un terme t de T tel qu'il existe une instance de s qui est égale à une instance de t. Donc il existe une substitution σ telle que t et s sont unifiables par σ. Etant donné tous les termes de S sont linéaires, en particulier s, donc la substitution σ est linéaire.
   Soient $T' = (T \ominus \{t\}) \cup \{\rho(t) \mid \rho \in B(\sigma,C)\}$ et $S' = (S \ominus \{s\}) \cup \{\rho(s) \mid \rho \in B(\sigma,C)\}$,
   par la proposition précédente nous avons:
   $I(T) = I(T \ominus \{t\}) \cup I(t) = I(T \ominus \{t\}) \cup (\cup_{\rho \in B(\sigma,C)} I(\rho(t)) \cup I(\sigma(t))) = I(T') \cup I(\sigma(t))$,
   de la même façon nous avons:
   $I(S) = I(S \ominus \{s\}) \cup I(s) = I(S \ominus \{s\}) \cup (\cup_{\rho \in B(\sigma,C)} I(\rho(s)) \cup I(\sigma(s))) =$

$I(S') \cup I(\sigma(s))$.
Donc puisque $\sigma(t) = \sigma(s)$, $I(T)$ contient $I(S)$ si et seulement si $I(T')$ contient $I(S')$.

Terminaison: A chaque étape de l'itération, nous supprimons un terme s de S. Dans le cas où le terme s est unifiable avec un terme t de T par une substitution linéaire σ, nous ajoutons à S les instances de s correspondantes aux substitutions complémentaires de σ. Ces termes que nous ajoutons sont moins généraux que s et, puisque nous prenons l'unificateur principal σ, alors quelle que soit ρ une substitution de $B(\sigma, C)$ il n'existe pas de substitution ρ' de $B(\sigma, C)$ telle que $\rho(s)$ et $\rho'(t)$ soient unifiables par une substitution linéaire. Donc à l'étape suivante, soit $\rho(s)$ n'est pas couvert par T soit il est couvert par un terme autre que les termes que nous avons ajouté à l'étape précédente; or T est fini, donc le processus termine.

Nous allons maintenant présenter le théorème qui permet de tester la convertibilité relative d'un opérateur défini f par rapport à un ensemble de constructeurs C. Le test se fait de la manière suivante:
Nous partons avec $T_o = MG(f, R)$ et $S_o = \{f(x_1, \ldots, x_n)\}$, puis nous allons répéter le processus, décrit dans le théorème précédent, jusqu'à ce que S soit vide ou qu'il n'existe plus de termes $t \in T$ et $s \in S$ qui peuvent être unifiés par une substitution linéaire. Ceci arrivera certainement puisque nous pouvons prouver que les termes de la forme $f(t_1, \ldots, t_n)$ produits, par unification ou par calcul de complément, n'ont aucun argument t_i de taille supérieure à la taille des mêmes termes dans T_o et S_o. Soient T_{fin} et S_{fin} les résultats finaux de ce processus.

3.4 Test de convertibilité

Dans ce paragraphe nous allons proposer un théorème de test du concept de convertibilité relative d'un opérateur défini par rapport à un ensemble de constructeurs.

3.4.1 Théorème de convertibilité

Théorème 3.3

- *Si T_{fin} et S_{fin} sont vides alors f est relativement convertible par rapport à C sans ambiguïté.*
- *Si T_{fin} n'est pas vide et S_{fin} est vide alors f est relativement convertible par rapport à C avec ambiguïté, c'est à dire que tous les termes de T_{fin} sont définis plus d'une fois.*

- *Si S_{fin} n'est pas vide alors f est relativement convertible par rapport à C si et seulement si tout terme de S_{fin} est R_C-inductivement réductible.*

Preuve: évidente.

S'il y a des relations entre les constructeurs, c'est à dire que R_C n'est pas vide ou C n'est pas libre, le test de la R_C-réductiblité-inductive est nécessaire pour avoir une condition nécessaire et suffisante alors que la R_C-réductibilté est suffisante pour avoir une condition suffisante mais pas nécessaire, pour tester la convertibilité relative d'un opérateur défini comme le montre l'exemple suivant.

Exemple 3.8 *Soit $C = \{0, s\}$, $D = \{+\}$ et supposons que l'ensemble des axiomes, spécifiant les entiers modulo 2, est donné par le système de réécriture suivant:*

$$R = \{ \begin{array}{lcl} s(s(0)) & \rightarrow & 0 \\ 0+x & \rightarrow & x \\ s(0)+x & \rightarrow & s(x) \end{array} \}.$$

$+$ est relativement convertible par rapport à C car $s(s(x))+y$ est R_C-inductivement réductible même s'il n'est pas R_C-réductible.

La troisième sorte de résultats retournés par l'algorithme, S_{fin}, est très intéressante puisqu'elle donne la forme de tous les termes où l'opérateur f n'est pas défini, dans le cas où ils ne sont pas R_C-inductivement réductibles. Cette propriété rend cet algorithme très utile dans un environnement de programmation fonctionnelle ou dans les types abstraits de données, comme nous allons le voir dans les exemples ci-dessous. Puisque ces termes sont obtenus par unification, alors ils sont, dans un certain sens, les plus généraux possible car nous prenons toujours l'unificateur principal. Ce qui permet de dire que cette méthode est meilleure que la méthode fondée sur les arbres de motifs qui est présentée par E. Kounalis [40], ainsi que celles de D. Plaisted [59] et D. Kapur, P. Narendran et H. Zhang [35].

Lors de l'unification, il est indispensable de prendre l'unificateur principal pour que la procédure termine comme le montre l'exemple suivant. Dans [67], J. J. Thiel pense qu'utiliser l'unificateur principal n'est pas indispensable, mais qu'il s'agit simplement d'une artifice visant à accélérer l'algorithme, l'exemple suivant contredit cette affirmation et montre que la procédure peut diverger lorsque nous ne prenons pas l'unificateur principal.

Exemple 3.9 *Reprenons l'exemple 3.1.*

$$\begin{array}{llll} \{0+y_1\} & \textit{couvre} & \{0+x_1\} & \textit{avec} \quad \sigma_1 = \{x_1 \longleftarrow 0, y_1 \longleftarrow 0\}, \\ \{0+s(y_2)\} & \textit{couvre} & \{0+s(x_2)\} & \textit{avec} \quad \sigma_2 = \{x_2 \longleftarrow 0, y_2 \longleftarrow 0\}, \\ \{0+s(s(y_3))\} & \textit{couvre} & \{0+s(s(x_3))\} & \textit{avec} \quad \sigma_3 = \{x_3 \longleftarrow 0, y_3 \longleftarrow 0\}, \\ & \vdots & & \vdots \\ \{0+s^n(y)\} & \textit{couvre} & \{0+s^n(x)\} & \textit{avec} \quad \sigma_n = \{x \longleftarrow 0, y \longleftarrow 0\}. \end{array}$$

La procédure 6.4 décrit cette méthode pour tester la convertibilité d'un opérateur par rapport à un système de réécriture et un ensemble de constructeurs.

Remarque 3.3 *Cette méthode reste valable pour les systèmes de réécriture équationnels R∪E en prenant la E-unification, c'est à dire l'unification modulo E. Elle n'est utile donc que pour les théories E pour lesquelles nous connaissons un algorithme de E-unification et de E-filtrage.*

Si T ne contient que des termes linéaires, cette procédure est une procédure de décision pour la convertibilité relative mais cette méthode peut ne pas fonctionner dans certains cas où le système de réécriture n'est pas linéaire à gauche. En général, nous prenons $S_o = \{f(x_1, \ldots, x_n)\}$, mais l'algorithme peut échouer si toutes les substitutions unifiant les termes sont non linéaires. Cependant un autre ensemble S_o, relativement convertible par rapport à C et non ambigu, peut ramener à bien le processus. Suivant le degré, le nombre d'occurrences d'une même variable, de la non linéarité du système de réécriture.

$$\begin{array}{lll} S_o = \bigcup_{c_i \in C} \{ & f(x_1, \ldots, x_{k-1}, & \\ & c_i(y_1, \ldots, y_{n_i}), & \\ & x_{k+1}, \ldots, x_n) & \}. \end{array}$$

C'est le cas de l'exemple suivant dû à E. Kounalis [40].

3.4.2 Exemples

Exemple 3.10 *Soit f la fonction qui calcule le report d'un additionneur binaire, qui est aussi la fonction "majorité" dans un système insensible aux défaillances [68].*
Soient $C = \{1, 0\}$, $D = \{f\}$ et R le système de réécriture suivant:

$$\begin{array}{lllll} R & = & \{ & f(x,x,y) & \rightarrow & x \\ & & & f(x,y,x) & \rightarrow & x \\ & & & f(y,x,x) & \rightarrow & x \quad \}. \end{array}$$

Nous avons: $f(x,y,z) = x \diamond y \diamond z$ avec la désignation booléenne suivante: $0 = vrai$. $1 = faux$ et $\diamond = ou\text{-}exclusif$.

Pour tester la complétude relative de cette spécification, ou la convertibilité relative de f par rapport à C, nous pouvons prendre $S_o = \{f(u,v,1), f(u,v,0)\}$. qui est relativement convertible par rapport à C et non ambigu, en effet:
$\{f(x,x,y), f(x,y,x), f(y,x,x)\}$ couvre $\{f(u,v,1), f(u,v,0)\}$ car $f(x,x,y)$ et $f(u,v,1)$ sont unifiables par $\sigma = \{u \longleftarrow x, v \longleftarrow x, y \longleftarrow 1\}$ qui n'est pas linéaire. Cependant
$f(x,y,x)$ et $f(u,v,1)$ sont unifiables par $\sigma = \{x \longleftarrow 1, y \longleftarrow v, u \longleftarrow 1\}$ qui est linéaire.

$$\begin{array}{lllll} B(\sigma, C) & = & \{ & \{x \longleftarrow 0, y \longleftarrow v, u \longleftarrow 1\}; & \\ & & & \{x \longleftarrow 1, y \longleftarrow v, u \longleftarrow 0\}; & \\ & & & \{x \longleftarrow 0, y \longleftarrow v, u \longleftarrow 0\} & \}. \end{array}$$

$\{f(x,x,y), f(y,x,x), f(0,v,0)\}$ couvre $\{f(u,v,0), f(0,v,1)\}$ car
$f(y,x,x)$ et $f(u,v,0)$ sont unifiables par $\sigma = \{x \longleftarrow 0, y \longleftarrow u, v \longleftarrow 0\}$ qui est linéaire.

$$\begin{array}{lllll} B(\sigma, C) & = & \{ & \{x \longleftarrow 1, y \longleftarrow u, v \longleftarrow 0\}; & \\ & & & \{x \longleftarrow 0, y \longleftarrow u, v \longleftarrow 1\}; & \\ & & & \{x \longleftarrow 1, y \longleftarrow u, v \longleftarrow 1\} & \}. \end{array}$$

$\{f(x,x,y), f(0,v,0). f(u,1,1)\}$ *couvre* $\{f(0,v,1). f(u,1,0)\}$ *car*

$f(x,x,y)$ *et* $f(0,v,1)$ *sont unifiables par* $\sigma = \{x \longleftarrow 0, y \longleftarrow 1, v \longleftarrow 0\}$ *qui est linéaire.*

$B(\sigma, C) = \{ \{x \longleftarrow 1, y \longleftarrow 0, v \longleftarrow 1\};$
$\{x \longleftarrow 1, y \longleftarrow 1, v \longleftarrow 0\};$
$\{x \longleftarrow 0, y \longleftarrow 0, v \longleftarrow 1\};$
$\{x \longleftarrow 1, y \longleftarrow 1, v \longleftarrow 1\};$
$\{x \longleftarrow 1, y \longleftarrow 0, v \longleftarrow 0\};$
$\{x \longleftarrow 0, y \longleftarrow 1, v \longleftarrow 1\};$
$\{x \longleftarrow 0, y \longleftarrow 0, v \longleftarrow 0\} \}.$

$\{f(0,v,0), f(u,1,1), f(1,1,1), f(1,1,0). f(0,0,0)\}$ *couvre* $\{f(u,1,0), f(0,1,1)\}$ *car*

$f(0,v,0)$ *et* $f(u,1,0)$ *sont unifiables par* $\sigma = \{v \longleftarrow 1, u \longleftarrow 0\}$ *qui est linéaire.*

$B(\sigma, C) = \{ \{v \longleftarrow 0, u \longleftarrow 0\};$
$\{v \longleftarrow 0, u \longleftarrow 1\};$
$\{v \longleftarrow 1, u \longleftarrow 1\} \}.$

$\{f(u,1,1), f(1,1,1), f(1,1,0), f(0,0,0)\}$ *couvre* $\{f(1,1,0), f(0,1,1)\}$ *car*

$f(u,1,1)$ *et* $f(0,1,1)$ *sont unifiables par* $\sigma = \{u \longleftarrow 0\}$ *qui est linéaire.*

$B(\sigma, C) = \{\{u \longleftarrow 1\}\}$.

$\{f(1,1,1), f(1,1,0), f(0,0,0)\}$ *couvre* $\{f(1,1,0)\}$ *car*

$f(1,1,0)$ *et* $f(1,1,0)$ *sont unifiables par la substitution identité qui est linéaire.*

$\{f(1,1,1), f(0,0,0)\}$ *couvre* $\emptyset$.

Donc f *est relativement convertible par rapport à* C *et les deux termes* $f(1.1,1)$ *et* $f(0,0,0)$ *sont définis plus d'une fois.*

Exemple 3.11 *Reprenons l'exemple LISTE présenté dans le chapitre 2.*

Soient $C = \{[], a, b, [-], @\}$, $D = \{flatten\}$ *et* R *le système de réécriture suivant:*

$R = \{ \; []@x \to x$
$x@[] \to x$
$(x@y)@z \to x@(y@z)$
$flatten([]) \to []$
$flatten([x]) \to flatten(x)$
$flatten(a) \to a$
$flatten(b) \to b$
$flatten(a@x) \to a@flatten(x)$
$flatten(b@x) \to b@flatten(x)$
$flatten([x]@y) \to flatten(x)@flatten(y) \; \}.$

Pour tester la complétude relative de cette spécification, ou la convertibilité relative de flatten par rapport à C, *nous prenons:*

$T_o = \{ \; flatten([]); flatten([x]); flatten(a); flatten(b); flatten(a@x); flatten(b@x);$
$flatten([x]@y) \; \}$ *et* $S_o = \{ \; flatten(u) \; \}$.

Après le déroulement du processus de recouvrement sur T_o *et* S_o *nous obtenons:*

$T_{fin} = \emptyset$ *et* $S_{fin} = \{ \; flatten([]@x_1); flatten((x_2@x_3)@x_4) \; \}$.

Or tous les termes de S_{fin} *sont réductibles donc* $flatten$ *est relativement convertible par rapport à* C *sans ambiguïté et par suite* R *est relativement complet.*

La procédure 6.3 décrit cette méthode pour tester la complétude relative d'un système de réécriture par rapport à un ensemble de constructeurs.

4
ω-COMPLETUDE

Dans le présent chapitre nous allons traiter des problèmes qui se sont posés lors de l'étude de la propriété de la ω-complétude d'une spécification algébrique. Nous avons pu résoudre certaines questions, mais le problème général reste ouvert. Pour cela nous avons étudié de nombreux exemples que nous présentons dans ce chapitre.

4.1 Introduction

La première question qui se pose lors d'une étude d'une propriété est sa décidabilité.

Question 4.1 *Est ce que la propriété de la ω-complétude d'une spécification est décidable ou non ?*

A notre connaissance, la vérification de la propriété de la ω-complétude n'a jamais été étudiée pour elle-même mais à l'occasion d'exemples bien particuliers. Cette propriété est reconnu par la communauté comme indécidable.

- A. Tarski [65] est le premier à évoquer le problème de la ω-complétude lors d'une étude systématique du *théorème de complétude* pour la logique équationnelle. Il pose en particulier le fameux 'problème de lycée' "high sckool problem".
- G. D. Plotkin [60] a étudié la non ω-complétude du λ-*calcul*.
- L. Henkin [24] a étudié la ω-complétude de la spécification des entiers naturels dans le cadre de la logique équationnelle.
- W. Taylor [66] a donné une liste d'exemples de théories spécifiables par un ensemble fini d'équations, *finitely based*, et une liste d'exemples non spécifiables par un ensemble fini d'équations dans le cadre d'une étude bibliographique sur l'algèbre universelle.

- E. Paul [57] a étudié la ω-complétude, sous le nom de la *complétude-inductive*, d'un certain nombre d'exemples de spécifications, dans le contexte du test de la validité d'une équation dans l'algèbre initiale associée à une spécification, par complétion inductive. C'est dans ce même cadre que nous sommes amenés à étudier cette propriété.
- J. Heering [23] a étudié la ω-complétude d'un certain nombre d'exemples de spécifications, dans le cadre de l'évaluation partielle des programmes.

4.2 Définition de la ω-complétude

Avant de donner la définition de la ω-complétude, nous allons rappeler la notion de *validité* qui sera utilisée plus loin.

Définition 4.1 *Soient A un ensemble d'équations et $t_1 = t_2$ une autre équation.*
$t_1 = t_2$ est valide dans la variété équationnelle $M(A)$, ou est un théorème équationnel, ce qui est noté $t_1 =_A t_2$, si et seulement si $t_1 = t_2$ est une conséquence de A par une suite de remplacement d'égaux par des égaux. $t_1 = t_2$ est valide dans l'algèbre initiale $I(F, A)$, ou est un théorème inductif, ce qui est noté $t_1 =_{ind(A)} t_2$, si et seulement si pour toute substitution σ de $\Sigma(F)$ $\sigma(t_1) =_A \sigma(t_2)$.

A cause de la complétude de la logique équationnelle, la preuve de la validité équationelle se fait par un simple raisonnement équationnel, c'est à dire par des remplacements d'égaux par des égaux. Dans le cas de la validité inductive, les équations closes restent prouvables par un simple raisonnement équationnel par contre les équations avec variables nécessitent un raisonnement par récurrence en général, comme nous l'avons vu dans le chapitre 2.

Remarquons que tout théorème équationnel est un théorème inductif. La réciproque est fausse en général.

Définition 4.2 *Soit (S, F, A) une spécification. (S, F, A) est dite ω-complète si et seulement si tout théorème inductif est un théorème équationnel.*

Autrement dit, la validité dans $I(F, A)$ devient équationnelle et par suite toute conséquence inductive de A est obtenue par un pur raisonnement équationnel.

Dans une spécification ω-complète, la théorie inductive et la théorie équationnelle coïncident.

Exemple 4.1 *Soit la spécification (S, F, A) des entiers naturels muni des opérations d'addition et de multiplication tel que:*

$$
\begin{array}{lll}
S & = & ENT\_NAT. \\
F & = & \{\ 0, s, +, \star \quad \}. \\
A & = & \{\ x+0 \quad == \quad x \\
 & & \quad x+s(y) \quad == \quad s(x+y) \\
 & & \quad x\star 0 \quad == \quad 0 \\
 & & \quad x\star s(y) \quad == \quad x+(x\star y) \quad \}.
\end{array}
$$

Cette spécification n'est pas ω-complète. Elle devient ω-complète [23] [24] si nous lui ajoutons l'ensemble A' d'équations suivant:

$$
\begin{array}{lll}
A' & = & \{\ x+y \quad == \quad y+x \\
 & & \quad x+(y+z) \quad == \quad (x+y)+z \\
 & & \quad x\star y \quad == \quad y\star x \\
 & & \quad x\star(y\star z) \quad == \quad (x\star y)\star z \\
 & & \quad x\star(y+z) \quad == \quad (x\star y)+(x\star z) \quad \}.
\end{array}
$$

qui expriment la commutativité et l'associativite de $+$ et $\star$ et la distributivité de $+$ par rapport à $\star$. Si nous ajoutons l'opérateur @ défini par l'ensemble A'' d'équations suivant:

$$
\begin{array}{lll}
A'' & = & \{\ x@0 \quad == \quad x \\
 & & \quad 0@x \quad == \quad 0 \\
 & & \quad s(x)@s(y) \quad == \quad s(x@y) \quad \},
\end{array}
$$

alors la nouvelle spécification n'est plus ω-complète et nous ne pouvons pas la rendre ω-complète par l'adjonction d'un ensemble fini d'équations [12]. Nous pouvons noter les théorèmes suivants:

$$
\begin{array}{lll}
A''' & = & \{\ x@x \quad == \quad x \\
 & & \quad (x+y)@x \quad == \quad x+y \\
 & & \quad x@(x+y) \quad == \quad 0 \\
 & & \quad \ddots \qquad\qquad \}.
\end{array}
$$

4.3 Etude de la ω-complétude

Dans toute la suite, nous allons parler de la ω-complétude d'un système de réécriture de termes au lieu de la ω-complétude d'une spécification.

Les deux techniques principales utilisées dans la littérature pour tester la ω-complétude d'un système de réécriture de termes donné sont les suivantes.

4.3.1 Méthode de discrimination

Cette technique est basée sur le théorème suivant:

Théorème 4.1 *Soit R un système de réécriture convergent.*
R est ω-complet si et seulement si

$$\forall (u,v) \in T(F,X)^2 \quad u\downarrow \neq v\downarrow \implies \exists \sigma \in \Sigma(F)\ \sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow .$$

Preuve: Par définition de la ω-complétude, R est ω-complet si et seulement si

$$\begin{array}{lccc} \forall (u,v) \in T(F,X)^2 & u =_{ind(A)} v & \Longrightarrow & u =_A v \\ & \Updownarrow & & \Updownarrow \\ & u =_A u\!\downarrow =_{ind(A)} v\!\downarrow =_A v & & u\!\downarrow = v\!\downarrow \\ & \Updownarrow & & \\ & \forall \sigma \in \Sigma(F) \sigma(u\!\downarrow) =_A \sigma(v\!\downarrow) & & \\ & \Updownarrow & & \\ & \sigma(u\!\downarrow)\!\downarrow = \sigma(v\!\downarrow)\!\downarrow & & \end{array}$$

Donc R est ω-complet si et seulement si

$\forall (u,v) \in T(F,X)^2 \quad \forall \sigma \in \Sigma(F)\ \sigma(u\!\downarrow)\!\downarrow = \sigma(v\!\downarrow)\!\downarrow \implies u\!\downarrow = v\!\downarrow$

et par contraposition nous avons: R est ω-complet si et seulement si

$\forall (u,v) \in T(F,X)^2 \quad u\!\downarrow \neq v\!\downarrow \implies \exists \sigma \in \Sigma(F)\ \sigma(u\!\downarrow)\!\downarrow \neq \sigma(v\!\downarrow)\!\downarrow$.

Cette technique a été utilisée pour vérifier la ω-complétude des exemples cités dans [45] et [46].

Exemple 4.2 *Soit le système de réécriture spécifiant un monoïde libre a deux éléments suivant:*

$$\begin{array}{lcll} F & = & \{ & e, a, b, . \ \}, \\ R & = & \{ & x.e \ \rightarrow \ x \\ & & & e.x \ \rightarrow \ x \\ & & & (x.y).z \ \rightarrow \ x.(y.z) \ \}. \end{array}$$

L'algèbre initiale $T(F)\!\downarrow_R$ est le monoïde libre $(\{a,b\},.)^\star$ et l'algèbre générique $T(F,X)\!\downarrow_R$ est équivalente à $(\{a,b\} \cup X,.)^\star$.

Soient $u\!\downarrow$ et $v\!\downarrow$ deux termes de $T(F,X)\!\downarrow_R$ tels que $u\!\downarrow \neq v\!\downarrow$.

$\exists (c,d) \in (\{a,b\} \cup X)^2$ tel que $c \neq d$, $u\!\downarrow = t.c.t'$ et $v\!\downarrow = t.d.t''$.

cas $c \in \{a,b\}$ *et* $d \in \{a,b\}$ *alors* $\forall \sigma \in \Sigma\ \ \sigma(u\!\downarrow)\!\downarrow \neq \sigma(v\!\downarrow)\!\downarrow$

- $c \in a,b$ *et* $d = x$ — *alors si* $c = a$ *(resp.* $c = b$*)*
  - *alors soit* $\sigma = \{x \longleftarrow b$ *(resp.* $x \longleftarrow a)\}$
  - *donc* $\sigma(u\!\downarrow)\!\downarrow \neq \sigma(v\!\downarrow)\!\downarrow$
- $c = x$ *et* $d = y$ — *alors soit* $\sigma = \{x \longleftarrow a, y \longleftarrow b\}$
  - *donc* $\sigma(u\!\downarrow)\!\downarrow \neq \sigma(v\!\downarrow)\!\downarrow$.

Donc ce système de réécriture est ω-complet.

Exemple 4.3 *Soit le système de réécriture spécifiant les entiers relatifs suivant:*

$$\begin{array}{lcll} F & = & \{ & 0, o, s \qquad\qquad\quad \}, \\ R & = & \{ & o(0) \ \rightarrow \ 0 \\ & & & o(o(x)) \ \rightarrow \ x \\ & & & s(o(s(x))) \ \rightarrow \ o(x) \ \} \end{array}$$

avec 0 est zero, o est l'opposé et s le successeur.

La forme normale des termes clos est sous l'une des formes suivantes:

0
$s^n(0)$ *pour* $n > 0$ *(où* $s^n(0) = \underbrace{s(s(...(s}_{n}(0))...))$*)*
$o(s^n(0))$ *pour* $n > 0$.

La forme normale des termes de l'algèbre générique $T(F, \{x\})\downarrow_R$ *est sous l'une des formes suivantes:*

$s^n(x)$ *pour* $n \geq 0$
$s^n(o(x))$ *pour* $n \geq 0$
$o(s^n(x))$ *pour* $n \geq 0$ *(où* $s^0(o(x)) = o(x)$*)*
$o(s^n(o(x)))$ *pour* $n > 0$.

Soient $u\downarrow$ *et* $v\downarrow$ *deux termes de* $T(F, \{x\})\downarrow_R$ *tels que* $u\downarrow \neq v\downarrow$.

cas $u\downarrow = s^n(x)$ *alors*
  cas $v\downarrow = s^m(o(x))$ *alors*
    cas $n = m$ *alors soit* $\sigma = \{x \longleftarrow s(0)\}$
      $\sigma(u\downarrow)\downarrow = s^{n+1}(0)$ *et* $\sigma(v\downarrow)\downarrow = s^{n-1}(0)$
      donc $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$
    $n \neq m$ *alors soit* $\sigma = \{x \longleftarrow 0\}$
      $\sigma(u\downarrow)\downarrow = s^n(0)$ *et* $\sigma(v\downarrow)\downarrow = s^m(0)$
      donc $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$
  $v\downarrow = o(s^m(x))$ *alors soit* $\sigma = \{x \longleftarrow 0\}$
    $\sigma(u\downarrow)\downarrow = s^n(0)$ *et* $\sigma(v\downarrow)\downarrow = o(s^m(0))$
    donc $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$
  $v\downarrow = o(s^m(o(x)))$ *alors soit* $\sigma = \{x \longleftarrow 0\}$
    $\sigma(u\downarrow)\downarrow = s^n(0)$ *et* $\sigma(v\downarrow)\downarrow = o(s^m(0))$
    donc $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$
$u\downarrow = s^n(o(x))$ *alors*
  cas $v\downarrow = o(s^m(x))$ *alors soit* $\sigma = \{x \longleftarrow 0\}$
    $\sigma(u\downarrow)\downarrow = s^n(0)$ *et* $\sigma(v\downarrow)\downarrow = o(s^m(0))$
    donc $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$
  $v\downarrow = o(s^m(o(x)))$ *alors soit* $\sigma = \{x \longleftarrow 0\}$
    $\sigma(u\downarrow)\downarrow = s^n(0)$ *et* $\sigma(v\downarrow)\downarrow = o(s^m(0))$
    donc $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$
$u\downarrow = o(s^n(x))$ *alors* $v\downarrow = o(s^m(o(x)))$
  cas $n = m$ *alors soit* $\sigma = \{x \longleftarrow s(0)\}$
    $\sigma(u\downarrow)\downarrow = o(s^{n+1}(0))$ *et* $\sigma(v\downarrow)\downarrow = o(s^{n-1}(0))$
    donc $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$
  $n \neq m$ *alors soit* $\sigma = \{x \longleftarrow 0\}$
    $\sigma(u\downarrow)\downarrow = o(s^n(0))$ *et* $\sigma(v\downarrow)\downarrow = o(s^m(0))$
    donc $\sigma(u\downarrow)\downarrow \neq \sigma(v\downarrow)\downarrow$.

Donc ce système de réécriture est ω*-complet.*

4.3.2 Méthode polynômiale

Cette technique est basée sur le théorème suivant: Soient G une grammaire de termes engendrant $T(F,X)\downarrow_R$ et $L(G)$ le langage reconnu par G, autrement dit, les termes de $T(F,X)\downarrow_R$ qui sont sous une forme polynômiale canonique nous avons:

$$\forall t \in T(F,X) \quad \exists! s \in L(G) \;\; t =_R s.$$

Théorème 4.2 *Soit R un système de réécriture convergent. R est ω-complet si et seulement si*

$$\forall (t_1,t_2) \in T(F,X)^2 \quad t_1 =_{ind(R)} t_2 \Longleftrightarrow rep(t_1) \equiv rep(t_2).$$

où $rep(t)$ est la représentation pôlynomiale d'un terme t par une grammaire des formes normales d'un système de réécriture et $\equiv$ est l'équivalence syntaxique.

Preuve: évidente.

Cette technique a été utilisée pour vérifier la ω-complétude des exemples cités dans [23] et [57].

Exemple 4.4 *Soit le système de réécriture spécifiant les entiers naturels construit à partir de 0, 1 et + comme suite:*

$$\begin{array}{lll} F = \{ & 0,1,+ & \}, \\ R = \{ & x+0 & \rightarrow \; x \\ & x+y & \rightarrow \; y+x \\ & x+(y+z) & \rightarrow \; (x+y)+z \quad \}. \end{array}$$

Ce système de réécriture est ω-complet. En effet, soient t_1 et t_2 deux termes de $T(F,X)$ tels que $t_1 =_{ind(R)} t_2$. En utilisant le système de réécriture R puis en regroupant les sous-termes identiques, t_1 et t_2 peuvent se mettre respectivement sous les formes suivantes:

$$rep(t_1) = (\underbrace{1+\cdots+1}_{n_0}) + (\underbrace{x_1+\cdots+x_1}_{n_1}) + \cdots + (\underbrace{x_p+\cdots+x_p}_{n_p})$$

et

$$rep(t_2) = (\underbrace{1+\cdots+1}_{m_0}) + (\underbrace{x_1+\cdots+x_1}_{m_1}) + \cdots + (\underbrace{x_p+\cdots+x_p}_{m_p})$$

Les x_i étant des variables distinctes apparaissant dans t_1 ou t_2. Les n_i ou m_i sont éventuellement nuls, auquel cas la variable x_i, ou la constante 1 pour $i=0$, n'apparait pas dans t_1 ou t_2. Soit $\sigma_0 = \{x_j \longleftarrow 0 \mid j \in [1,p]\}$ $\sigma_0(rep(t_1)) =_R \sigma_0(rep(t_2))$ implique que $\underbrace{1+\cdots+1}_{n_0} =_R \underbrace{1+\cdots+1}_{m_0}$. D'où d'après la théorie des polynômes nous avons $n_0 = m_0$. Soit $\sigma_i = \{x_i \longleftarrow 1, x_j \longleftarrow 0 \mid j \in [1,p] \wedge j \neq i\}$ $\sigma_i(rep(t_1)) =_R \sigma_i(rep(t_2))$ implique que $\underbrace{1+\cdots+1}_{n_i} =_R \underbrace{1+\cdots+1}_{m_i}$. D'où d'après la théorie des polynômes nous avons $n_i = m_i$ pour tout $i \in [1,p]$. Donc $rep(t_1)$ et $rep(t_2)$ sont identiques.

4.3.3 Enrichissement ω-complet

En pratique, nous ne pouvons pas utiliser directement ces deux approches car elles introduisent la manipulation d'ensembles infinis. Nous allons exposer une autre méthode plus générale. Nous allons d'abord poser une question assez naturelle.

Question 4.2 *Soit $P = (S, F, A_P)$ une spécification, existe t-il une spécification $Q = (S, F, A_Q)$ telle que:*

- *$A_Q \supset A_P$ et A_Q est fini,*
- *$I(F, A_Q) = I(F, A_P)$,*
- *Q est ω-complète.*

Autrement dit, étant donnée une spécification, admet elle toujours un enrichissement protégé, fini et ω-complet ?

La réponse est en général "non", car sinon l'induction serait axiomatisable au premier ordre ce qui n'est pas vrai, et comme le montre l'exemple suivant:

Exemple 4.5 *Soit la spécification (S, F, A) des entiers relatifs telle que:*

$$
\begin{array}{lcll}
S & = & ENT\_REL, & \\
F & = & \{\ 0, s, p, + & \}, \\
A & = & \{\ x+0 & == \ x \\
 & & x+s(y) & == \ s(x+y) \\
 & & p(0) & == \ 0 \\
 & & p(s(x)) & == \ x \qquad \}, \\
A' & = & \{\ x+y & == \ y+x \\
 & & x+(y+z) & == \ (x+y)+z \quad \} \\
 & \cup & \{\ p(x+x) & == \ p(x)+x \\
 & & p(x+x+x) & == \ p(x)+x+x \\
 & & \ddots & \qquad \}.
\end{array}
$$

Nous ne pouvons pas obtenir un enrichissement fini et ω-complet. Il y a une infinité de théorèmes inductifs indépendants dans A' qu'il faut ajouter à A. Remarquons que $p(x+y) = p(x) + y$ n'est pas un théorème inductif. En effet $p(0+s(0)) = p(s(0+0)) = p(s(0)) = 0$ alors que $p(0) + s(0) = 0 + s(0) = s(0+0) = s(0)$.

Dans cet exemple le domaine de l'algèbre initiale est infini, nous pouvons penser que dans le cas contraire ce serait possible. L'exemple suivant montre que se n'est pas le cas.

Exemple 4.6 *Soit la spécification suivante dûe à R. C. Lyndon [50]:*

$$
\begin{array}{lcl}
F & = & \{\ 0, e, b_1, b_2, c, d_1, d_2, \star\ \}, \\
A & = & \{\ \star \text{ est partout égal à } 0 \text{ sauf en:}
\end{array}
$$

$$
\begin{array}{lllll}
 & c\star e & == & c & \\
 & c\star b_i & == & d_i & \text{où } i,j=1,2.\\
 & d_i\star e & == & d_i & \\
 & d_i\star b_j & == & d_i & \},\\
A' = \{ & 0\star x & == & 0 & \\
 & x\star 0 & == & 0 & \\
 & x\star(y\star z) & == & 0 & \\
 & ((\cdots((x_1\star x_2)\star x_3)\cdots)\star x_n)\star x_1 & == & 0 & \\
 & ((\cdots((x_1\star x_2)\star x_3)\cdots)\star x_n)\star x_2 & == & (\cdots((x_1\star x_2)\star x_3)\cdots)\star x_n & \}.
\end{array}
$$

où $n=1,2,\ldots$.

Dans cet exemple le domaine de l'algèbre initiale de la variété équationnelle est fini mais nous ne pouvons pas obtenir une spécification finie et ω-complète de l'opérateur $\star$. Il y a une infinité de théorèmes inductifs indépendants dans A' qu'il faut ajouter à A. Cette spécification est ω-complète.

Problème 4.1 *Comment et sous quelles conditions une spécification admet elle un enrichissement ω-complet fini ?*

Autrement dit, existe t-il une axiomatisation finie dont tous les théorèmes inductifs peuvent être démontrés équationnellement ?

La réponse est en général "non", car axiomatiser la ω-complétude est la même chose qu'axiomatiser l'induction.

Cela nous amène à chercher quand une équation $t_1 = t_2$ peut être un théorème inductif ?

Les questions suivantes se posent:

- t_1 et/ou t_2 doivent-ils être en formes normales ?
- t_1 et/ou t_2 doivent-ils être inductivement réductibles ?
- $\cdots$?

Les réponses de ces deux premières questions sont:

- t_1 et t_2 doivent être en formes normales car nous cherchons l'existence de théorèmes inductifs qui ne sont pas équationnels.
- t_1 et t_2 ne sont pas obligatoirement inductivement réductibles en général comme le montrent les exemples suivants:

Exemple 4.7 *Soit la spécification d'un groupe engendré par un seul élément suivante:*

$$
\begin{array}{lllll}
F = \{ & e,a,i,\star & & & \},\\
A = \{ & x\star e & == & x & \\
 & x\star(y\star z) & == & (x\star y)\star z & \\
 & x\star i(x) & == & e & \}.
\end{array}
$$

$x \star y = y \star x$, commutativité d'un groupe engendré par un seul élément. est un théorème inductif alors que $x \star y$ n'est pas inductivement réductible en effet $a \star a$. qui est une instance close de $x \star y$, n'est pas réductible par rapport au système de réécriture associé à A en orientant les équations de gauche à droite.

Exemple 4.8 *Soit la spécification suivante:*

$F = \{ a, b, f.g \}.$

$A = \{ f(a) == g(a)$

$\quad\quad g(b) == f(b) \}.$

$f(x) = g(x)$ est un théorème inductif alors que ni $f(x)$ ni $g(x)$ n'est inductivement réductible par rapport au système de réécriture associé à A en orientant les équations de gauche à droite.

4.3.4 ω-complétude relative

Nous allons nous restreindre aux théorèmes orientables en règles de réécriture et aux systèmes de réécriture convergents dont la réductibilité inductive est décidable.

Définition 4.3 *Soient R un système de réécriture convergent et $\prec$ un ordre de réduction qui assure la terminaison de R. R est relativement ω-complet par rapport à $\prec$ si et seulement si tout théorème inductif et orientable par $\prec$ est un théorème équationnel.*

Lemme 4.1 *Soit R un système de réécriture de termes orienté par un ordre $\prec$. Si R est ω-complet alors R est relativement ω-complet par rapport à $\prec$.*

La réciproque est fausse, car il peut exister des théorèmes inductifs non orientables en règles de réécriture, comme nous l'avons vu dans l'exemple 4.1 où la commutativité est un théorème inductif orientable par aucun ordre.

La ω-complétude relative peut servir entre autre pour tester la non ω-complétude d'une spécification.

Dans le cas de la ω-complétude relative, au moins un des deux termes de l'équation doit être inductivement réductible. C'est en particulier le membre gauche, après orientation de l'équation.

Théorème 4.3 *Si tout terme en forme normale a au moins une instance close en forme normale alors R est relativement ω-complet par rapport à $\prec$.*

Preuve: Par l'absurde. Supposons que R n'est pas ω-complet par rapport à $\prec$. Donc il existe une équation en forme normale $t_1 = t_2$ orientable par rapport à $\prec$ telle que $t_1 =_{ind(R)} t_2$ et $t_1 \neq_R t_2$. Supposons que $t_1 = t_2$ s'oriente

par rapport à $\prec$ en $t_1 \rightarrow t_2$. Soit σ une substitution close. Nous avons $\sigma(t_1)\downarrow = \sigma(t_2)\downarrow$ et $\sigma(t_2)\downarrow \prec \sigma(t_2) \prec \sigma(t_1)$. Donc $\sigma(t_1) \neq \sigma(t_1)\downarrow$. Donc t_1, qui est en forme normale, n'a aucune de ses instances closes en forme normales.

Définition 4.4 *Soit t un terme de $T(F, X)$.*
La profondeur de t est définie par:

$$prof(t) = \begin{cases} 1 & t \in X \vee t \in F_0, \\ max(\{prof(t_1), \ldots, prof(t_n)\}) + 1 & t = f(t_1, \ldots, t_n) \wedge f \in F_n \wedge n > 0. \end{cases}$$

La profondeur d'une règle est égale à la profondeur du membre gauche de la règle.

Pour que le théorème précédent soit opérationel nous pouvons essayer de résoudre le problème suivant:

Problème 4.2 *Existe t-il un entier naturel α tel que: si tout terme en forme normale de profondeur inférieure ou égale à α a une instance close en forme normale alors tout terme en forme normale a une instance close en forme normale.*

La détermination de α n'est pas simple, comme le montre l'exemple suivant:

Exemple 4.9 *Soit n un entier naturel.*

$$\begin{array}{lcl} F & = & \{+\} \cup_{i \in [0,n]} G_i \quad où \\ G_i & = & \{a_i, f_i\} \quad où \quad arité(a_i) = 0 \quad et \quad arité(f_i) = 1, \\ R & = & \cup_{i \in [0,n]} R_i \quad où \\ R_i & = & \{ \quad f_i(a_i) \rightarrow a_i \\ & & \quad f_i(f_i(x)) \rightarrow x \\ & & \quad f_i(x + y) \rightarrow f_i(x) + f_i(y) \quad \}. \end{array}$$

Tout terme irréductible de profondeur inférieure ou égale à $E(\log_2 n)+1$ où E est la partie entière et $\log_2$ est le logarithme à base 2, est non inductivement réductible. Alors qu'il existe des termes inductivement réductible, de profondeur supérieure, comme par exemple: $f_1(x) + (f_2(x) + (\cdots (f_{n-1}(x) + f_n(x)) \cdots))$.

Nous pouvons tout de même espérer résoudre les deux problèmes suivants:

Problème 4.3 *Existe t-il un entier naturel β tel que: si tout théorème inductif de profondeur inférieure ou égale à β est équationnel alors il n'existe plus de théorème inductif et par suite R est ω-complet par rapport à $\prec$.*

Problème 4.4 *Existe t-il deux entiers naturels γ et θ tels que: s'il existe un théorème inductif de profondeur comprise entre γ et θ alors il existe une infinité de théorèmes inductifs indépendants, c'est à dire non équationnels, et par suite R n'admet pas d'enrichissement fini ω-complet par rapport à $\prec$.*

Si ce dernier problème est résolu, nous pouvons nous passer de la restriction sur la finitude de l'enrichissement, en utilisant le concept de *méta-théorème* qui sera une extension du concept de *méta-règle* [KIR 87].

4.4 Décidabilité de la ω-complétude

Nous allons nous restreindre dans ce paragraphe au cas des spécifications où le domaine de l'algèbre initiale de la variété équationnelle associée est fini pour décider la propriété de ω-complétude. Même dans ce cas, il existe des spécifications algébriques qui n'admettent pas un enrichissement fini ω-complet comme le montre l'exemple 4.6 dû à R. C. Lyndon [50].

Nous allons étudier maintenant un cas particulier où la ω-complétude est décidable. L'idée de base est de se restreindre au cas où le domaine de l'algèbre générique de la variété équationnelle à un renommage des variables près est fini.

Théorème 4.4 *S'il n'y a qu'un nombre fini de termes à un renommage des variables près et à une égalité près alors la propriété de la ω-complétude est décidable.*

En effet, il suffit de prendre tous les couples de termes et de tester chaque fois si c'est un théorème inductif ou pas en le vérifiant cas par cas, ce qui peut être fait exhaustivement puisque le domaine de l'algèbre initiale de la variété équationnelle, ou l'ensemble des termes clos modulo la variété équationnelle, est fini s'il n'y a qu'un nombre fini de termes à un renommage des variables près et à une égalité près. Le nombre de couples, à un renommage des variables près, étant fini, la ω-complétude est décidable.

Exemple 4.10 *Soit la spécification suivante:*

$$\begin{array}{lcll} F & = & \{ & e, a, * \qquad\qquad\qquad\qquad\quad \}, \\ A & = & \{ & e * x == x * e == x \\ & & & x * (y * z) == (x * y) * z == e \quad \}. \end{array}$$

Le domaine de l'algèbre initiale de $M(A)$ est $\{e, a, a*a\}$ et le domaine de l'algèbre générique sur X de $M(A)$ à un renommage des variables près est $\{e, a.x, a*a, x*x, x*y\}$. Cette spécification n'est pas ω-complète car $x*y = y*x$ est un théorème inductif.

Dans le chapitre 5 nous allons présenter une méthode, fondée sur la réécriture, les automates et les grammaires d'arbres, qui permet de tester la finitude d'un langage d'arbres. Ça sera, en particulier, le cas pour le domaine de l'algèbre initiale et pour celui de l'algèbre générique d'une variété équationnelle. Cette méthode permet aussi d'engendrer le domaine de l'algèbre initiale et celui de l'algèbre générique d'une variété équationnelle.

Soient X un ensemble infini de variables, supposé toujours dénombrable, et R un système de réécriture convergent dont le domaine de l'algèbre initiale de la variété équationnelle associée est fini.

La notion de renommage des variables peut être formalisée par la relation suivante.

Définition 4.5 *Soit $\sim$ une relation sur $T(F,X)\downarrow_R^2$ définie par:*
$(t_1,t_2) \sim (t'_1,t'_2) \Longleftrightarrow \exists \sigma : V(t_1) \cup V(t_2) \longrightarrow X$ *injective telle que:* $\sigma(t_1) = t'_1$ *et* $\sigma(t_2) = t'_2$.

La relation $\sim$ est une relation d'équivalence. L'ensemble des couples de termes à un renommage des variables près et à une égalité près est $T(F,X)\downarrow_R^2 /_\sim$.

Soit $profc((t_1,t_2)) = sup(\{prof(t_1), prof(t_2)\})$.

Lemme 4.2 *Soit n un entier naturel.*
$\{(t_1,t_2) \mid (t_1,t_2) \in T(F,X)\downarrow_R^2 /_\sim \wedge profc(t_1,t_2) = n\}$ *est fini.*

Preuve: Soit $\alpha = sup(\{arité(f) \mid f \in F\})$. Un terme de profondeur n a au maximum α^n variables distinctes. Donc il suffit de se restreindre à un ensemble X' fini de variables de cardinal $2 * \alpha^n$. $K = \{(t_1,t_2) \mid (t_1,t_2) \in T(F,X')\downarrow_R^2 \wedge profc(t_1,t_2) = n\}$ est fini. $\forall (t_1,t_2) \notin K \quad \exists (t'_1,t'_2) \in K \quad \exists \sigma :$ $V(t_1) \cup V(t_2) \longrightarrow X'$ injective telle que: $\sigma(t_1) = t'_1$ et $\sigma(t_2) = t'_2$.

Lemme 4.3 $T(F,X)\downarrow_R^2 /_\sim$ *est fini si et seulement si* $sup(\{prof(t) \mid t \in T(F,X)\downarrow_R\})$ *est fini.*

Preuve: évidente.

Exemple 4.11 *Le type abstrait BOOLEEN peut être construit à partir des constantes vrai et faux, et de l'opérateur unaire de négation $\neg$.*

$$
\begin{array}{lcll}
F & = & \{ & vrai, faux, \neg \quad \}, \\
R & = & \{ & \neg vrai \quad \rightarrow \quad faux \\
 & & & \neg faux \quad \rightarrow \quad vrai \\
 & & & \neg\neg x \quad \rightarrow \quad x \qquad \}, \\
T(F)\downarrow_R & = & \{ & vrai, faux \quad \}, \\
T(F,X)\downarrow_R /_\sim & = & \{ & vrai, faux, \quad x, \neg x \quad \}.
\end{array}
$$

Il n'y a plus de théorèmes inductifs formés de couples de termes de $T(F,X)\downarrow_R /_\sim$, donc cette spécification est ω-complète.

Nous présentons dans ce qui suit un cas particulier de systèmes de réécriture où le domaine de l'algèbre générique de la variété équationnelle associée à un renommage des variables près est fini.

Soient X un ensemble infini de variables et R un système de réécriture convergent dont le domaine de l'algèbre initiale de la variété équationnelle associée est fini.

Nous allons nous situer sous les 3 hypothèses suivantes: $\forall g \rightarrow d \in R$

H1. $prof(g) \leq prof(d)$, où $prof(t)$ est la profondeur d'un terme t.

H2. $V(g) = V(d)$, où $V(t)$ est l'ensemble des variables d'un terme t.

H3. $\forall x \in V(g) \quad profv(x, g) = profv(x, d)$, où $profv(x, t)$ est le maximum des profondeurs de tous les sous-termes contenant x d'un terme t.

Lemme 4.4 *Sous les 3 hypothèses H1, H2 et H3,*
$\forall g \to d \in R \quad \forall \sigma \in \Sigma(F, X) \quad prof(\sigma(g)) \leq prof(\sigma(d))$.

Preuve: évidente.

Lemme 4.5 *Sous les 3 hypothèses H1, H2 et H3,*
$\forall (t, t') \in T(F, X)^2 \quad t \xrightarrow{\star} t' \Longrightarrow prof(t) \leq prof(t')$.

Preuve: Il suffit de prouver que:

$$\forall (t_1, t_2) \in T(F, X)^2 \quad t_1 \longrightarrow t_2 \Longrightarrow prof(t_1) \leq prof(t_2).$$

Soient t_1 et t_2 deux termes de $T(F, X)^2$ tels que: $t_1 \longrightarrow t_2$.
Or $t_1 \longrightarrow t_2$ implique $\exists u \in G(t_2) \quad \exists g \to d \in R \quad \exists \sigma \in \Sigma(F, X)$ tels que:
$t_2/u = \sigma(g)$ et $t_1 = t_2[u \longleftarrow \sigma(d)]$.
Or $prof(t_1) \leq prof(t_2)$ est équivalent à $prof(\sigma(g)) \leq prof(\sigma(d))$.

Théorème 4.5 *Sous les 3 hypothèses H1, H2 et H3, si le domaine de l'algèbre initiale est fini alors le domaine de l'algèbre générique à un renommage des variables près est fini.*

Preuve: Par l'absurde. Soient $\alpha = sup(\{prof(t) \mid t \in T(F) \downarrow_R\})$ et t un terme de $T(F, X) \downarrow_R$ tel que $prof(t) \geq \alpha$. Soit t' une instance close de t, $prof(t') \geq prof(t)$ donc t' est réductible vers un terme t'' et $prof(t'') \geq prof(t')$. Nous recommençons le même processus pour t'' et ainsi de suite indéfiniment. Ceci contredit le fait que R est nœthérien.

Dans ce cas nous sommes capable de tester la propriété d'ω-complétude.

Dans tout ce chapitre nous avons eu besoin du langage des formes normales du système de réécriture, d'où l'intérêt de sa caractérisation. C'est l'objet du chapitre suivant.

5

RECONNAISSANCE ET GENERATION DES FORMES NORMALES CLOSES

Le but de ce chapitre est de présenter une méthode réalisée pour engendrer le langage régulier d'arbres des termes clos en forme normale par rapport à un système de réécriture de termes linéaire à gauche donné. Ce langage est isomorphe au domaine de l'algèbre initiale associée au système de réécriture donné. Cette méthode permet aussi de décider la finitude et la vacuité de ce langage.

La méthode proposée consiste à trouver tout d'abord un automate d'arbres, déterministe fini, qui reconnaît le langage d'arbres des termes clos en forme normale puis à construire la grammaire d'arbres associée. Le langage des termes clos en forme normale, par rapport à un système de réécriture de termes, étant le complémentaire du langage des termes clos réductibles par rapport au langage des termes clos. Nous commençons donc par construire l'automate, déterministe fini, qui reconnaît le langage des termes réductibles puis nous cherchons son complémentaire.

5.1 Introduction

En théorie des langages, le moyen le plus classique pour reconnaître un langage régulier est l'utilisation des machines formelles appelées *automates*. Un langage est un ensemble de mots et un mot est une suite finie d'éléments d'un vocabulaire donné. Un vocabulaire est un ensemble fini de symboles appelés lettres.

Pour connaître si un mot est un élément d'un langage, l'automate de mot fonctionne de la manière suivante: Il part d'un état dit initial puis il change d'état en lisant le mot de gauche à droite, ou l'inverse, jusqu'à ce qu'il arrive à la fin du mot. S'il s'arrête avec un état dit de satisfaction alors le mot appartient au langage sinon il n'appartient pas au

langage.

Dans le cas des langages d'arbres, un mot n'est plus une suite d'éléments d'un vocabulaire mais un arbre étiqueté par un vocabulaire dont les symboles, appelés des opérateurs, sont gradués par une fonction arité qui à chaque opérateur associe un entier naturel. Donc nous aurons des *automates d'arbres*.

Les automates sont des cas particuliers des automates d'arbres et les résultats dans la théorie des automates de mots restent en général valables dans une forme généralisée appropriée [52]. De la même façon nous introduisons les *grammaires d'arbres* pour la génération des langages d'arbres.

H. Comon et J. L. Rémy [11] ont proposé une méthode, basée sur la notion de *disunification* [10], pour caractériser le langage des formes normales closes.

5.2 Automate et grammaire d'arbres

Nous allons tout d'abord rappeler certaines notations utiles pour la suite.

Convention 5.1
X: ensemble de variables,
F: ensemble fini d'opérateurs,
F_i: ensemble des opérateurs d'arités i où $i \geq 0$,
$T(F,X)$: ensemble des termes,
$T(F)$: ensemble des termes clos.

5.2.1 Automate d'arbres

Dans ce paragraphe, nous allons présenter les différentes notions de base du concept d'automate d'arbres, nécessaires pour les paragraphes qui suivent.

Définition 5.1 *Un automate d'arbres A sur F est un triplet (E,S,P) tel que:*

- *E: ensemble des états de A,*
- *S: ensemble des états de satisfaction, ou final, de A, où $S \subseteq E$,*
- *P: ensemble des règles de transition dont la forme est:*
  $f(e_1,\ldots,e_n) \rightarrow e_0$ appelée transition de réduction,
  $e \rightarrow e'$ appelée Λ-transition
  où $e,e',e_0,e_1,\ldots,e_n \in E$; $f \in F_n$ et Λ est l'arbre vide.

Remarque 5.1

- *Un automate d'arbres est en fait un quadruplet (F, E, S, P). F étant le vocabulaire des arbres, il reste inchangé durant toute la suite.*
- *Il n'y a pas d'état initial, comme nous trouvons habituellement dans la littérature des automates de mots. En effet, pour $f \in F_0$, constante, nous avons $f() \rightarrow e_f$ comme point de départ. En fait nous pouvons prendre pour états initiaux l'ensemble $I = \{\ e_c \mid c \in F_0 \wedge c() \rightarrow e_c\ \}$.*
- *P est un système de réécriture de termes clos dans $T(F \cup E)$ où les éléments de E sont des constantes.*

Dans toute la suite, nous parlerons d'automate au lieu d'automate d'arbres sur F, s'il n'y a pas d'ambiguïté sur F.

Convention 5.2
Nous notons $\longrightarrow_P$ la relation de réduction associée au système de réécriture P et $\xrightarrow{*}_P$ la fermeture réflexive-transitive de $\longrightarrow_P$. $\longrightarrow_P$ et $\xrightarrow{*}_P$ seront notées respectivement $\longrightarrow$ et $\xrightarrow{*}$ en abréviation s'il n'y a pas d'ambiguïté sur P.

Définition 5.2 *Soit $A = (E, S, P)$ un automate. Un état e de E est dit accessible si et seulement si il existe un terme t de $T(F)$ tel que $t \xrightarrow{*}_P e$.*

Définition 5.3

- *Un automate est dit fini si et seulement si son ensemble d'états est fini.*
- *Un automate est dit déterministe si et seulement si il n'y a aucun terme $f(e_1, \ldots, e_n)$ tel que $f(e_1, \ldots, e_n) \xrightarrow{*} e$ et $f(e_1, \ldots, e_n) \xrightarrow{*} é$ où $e \neq é$ et si de plus pour tout terme $f(e_1, \ldots, e_n)$ il existe une transition. En terme de système de réécriture cela signifie que le système est "sans superposition" ou "non ambigu" dans la terminologie d'Huet et Lévy.*
- *Un automate est dit sans Λ si et seulement si c'est un automate avec aucune Λ-transition.*

Remarque 5.2 *Un automate déterministe fini peut être considéré comme une $F \cup E$-algèbre finie dont le domaine est l'ensemble E.*

Définition 5.4

- *Un automate déterministe est dit réduit si et seulement si en tant que $F \cup E$-algèbre il ne contient aucune sous-algèbre autre que lui-même.*

- *Un automate est dit minimal si et seulement si il est réduit et en tant que $F \cup E$-algèbre il ne peut pas avoir d'algèbre quotient autre qu'elle-même, c'est à dire qu'il n'y a aucune congruence non triviale. Ce qui revient à dire que son ensemble d'états est minimal.*

Dans toute la suite, nous ne considérons que les automates finis.

Définition 5.5 *Soit $A = (E, S, P)$ un automate sur F.*

- *Un terme t de $T(F)$ est reconnu par A, si et seulement si il existe un état e de S tel que $t \xrightarrow{*}_P e$.*
- *Le sous-ensemble de $T(F)$ des termes reconnus par A, noté $L(A)$, est appelé langage d'arbres sur F reconnu par A.*

Dans toute la suite, nous parlerons de langage au lieu de langage d'arbres sur F, s'il n'y a pas d'ambiguïté. Comme pour la théorie des automates de mots, nous avons:

Lemme 5.1 *Un langage sur F, sous-ensemble de $T(F)$, est reconnaissable par un automate indéterministe si et seulement si il est reconnaissable par un automate déterministe.*

Nous allons considérer que les automates déterministes durant le reste de ce paragraphe. Nous allons voir comment enlever l'indéterminisme dans le paragraphe suivant.

Pour l'application de la caractérisation du langage des formes normales closes, nous avons formalisé certains résultats, connus sur les automates de mots, dans le cas des automates d'arbres. L'utilisation des techniques de la réécriture facilite énormément leurs démonstrations.

Théorème 5.1 *Soit L un langage sur F reconnu par un automate $A = (E, S. P)$. Le langage L', complémentaire de L par rapport à $T(F)$, est reconnu par l'automate complémentaire $A' = (E', S', P')$ tel que $E' = E$, $S' = E \ominus S$ et $P' = P$.*

Dans la suite de ce paragraphe, nous allons étudier la décidabilité de l'équivalence de deux automates différents, c'est à dire l'égalité des deux langages reconnus par ces deux automates, ainsi que la finitude et la vacuité du langage reconnu par un automate.

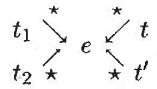
Convention 5.3

- *La relation sous-terme (resp. sous-terme strict) sera noté par $t \succeq t'$ (resp. $t \succ t'$) où t' sous-terme de t.*
- *Le remplacement dans un terme t d'un sous-terme t_1 de t par un terme t_2 sera noté par $t[t_1 \longleftarrow t_2]$.*

Le lemme suivant est relativement intuitif mais fondamental pour la preuve des théorèmes qui suivent.

Lemme 5.2 *Soit $A = (E, S, P)$ un automate. Soient t, t', t_1 et t_2 quatre termes tels que $t \succeq t_1$ et $t' = t[t_1 \longleftarrow t_2]$ et e, e' deux états de E.*
$t_1 \xrightarrow{\star} e \wedge t_2 \xrightarrow{\star} e \wedge t \xrightarrow{\star} e' \implies t' \xrightarrow{\star} e'$.

Preuve: Par induction structurelle sur le terme t.

- $t = \Lambda$: nous avons $t_1 = t$ et $t' = t_1$ et par suite  ,
- $t = f(s_1, \ldots, s_n)$: si le remplacement se fait dans un sous-terme s_j de t alors par hypothèse de récurrence t et t' ont le même état.
  Si $t_1 = s_j$ alors $t' = f(s_1, \ldots, s_{j-1}, t_2, s_{j+1}, \ldots, s_n)$ et par suite t et t' ont le même état puisque t_1 et t_2 ont le même état.

Le théorème suivant donne une condition nécessaire et suffisante pour qu'un langage soit vide.

Théorème 5.2 *Soit A un automate dont le nombre d'états est α. $L(A)$ n'est pas vide si et seulement si A reconnaît au moins un terme de profondeur inférieure ou égale à α.*

Preuve: La preuve de ce théorème se réduit à la preuve de la condition nécessaire. Soit t un terme de profondeur β reconnu par A, $t \xrightarrow{\star} e_0$ où e_0 est un état de satisfaction de A. Si $\beta \leq \alpha$ alors le théorème est vérifié. Sinon nous allons démontrer qu'il existe un terme t' de profondeur inférieure ou égale à α reconnu par A. La construction du terme t', en fonction de t, se fait de la manière suivante: t étant de profondeur β alors il existe au moins une suite de sous-termes $(t_0, \ldots, t_\beta)$ de t telle que $t = t_0 \succ t_1 \succ \cdots \succ t_\beta \in F_0$. Soit $(e_0, \ldots, e_\beta)$ la suite des états de A telle que $\forall i \in [0, \beta]$ $t_i \xrightarrow{\star} e_i$. Or $\beta > \alpha$ donc il existe des états de cette suite qui sont identiques. Soient e_l et e_k deux états de cette suite qui sont égaux avec $l < k$. Soit $t' = t[t_l \longleftarrow t_k]$. D'après le lemme 5.2 $t' \xrightarrow{\star} e_0$ et la profondeur de t' est inférieure à la profondeur de t. Si la profondeur de t' est inférieure ou égale à α alors le théorème est vérifié. Sinon il suffit de recommencer le même processus sur t' jusqu'à ce qu'un sous-terme de profondeur inférieure ou égale à α soit obtenu.

Le lemme suivant donne une condition suffisante pour qu'un langage soit infini.

Lemme 5.3 *Soit A un automate dont le nombre d'états est α. Si A reconnaît un terme de profondeur supérieure ou égale à $\alpha + 1$ alors $L(A)$ est infini.*

Preuve: La preuve de ce théorème repose sur le même principe que la preuve précédente. Soit t un terme reconnu par A de profondeur β avec $\beta > \alpha$. Nous allons démontrer qu'il existe un terme t' de profondeur supérieure à t reconnu par A. La construction du terme t', en fonction de t, se fait de la manière suivante: t étant de profondeur β alors il existe au moins une suite de sous-termes $(t_0, \ldots, t_\beta)$ de t telle que $t = t_0 \succ t_1 \succ \cdots \succ t_\beta \in F_0$. Soit $(e_0, \ldots, e_\beta)$ la suite des états de A telle que $\forall i \in [0.\beta]$ $t_i \xrightarrow{*} e_i$. Or $\beta > \alpha$ donc il existe des états de cette suite qui sont identiques. Soient e_l et e_k deux états de cette suite qui sont égaux avec $l < k$. Soit $t' = t[t_k \longleftarrow t_l]$. D'après le lemme 5.2 $t' \xrightarrow{*} e_0$ et la profondeur de t' est supérieure à la profondeur de t. Si nous itérons le même processus sur t', nous obtenons une infinité de termes reconnus par A.

Le théorème suivant donne une condition nécessaire et suffisante pour qu'un langage soit infini.

Théorème 5.3 *Soit A un automate dont le nombre d'états est α. $L(A)$ est infini si et seulement si A reconnaît au moins un terme de profondeur β avec $\alpha < \beta \leq 2 * \alpha$.*

Preuve: La preuve de ce théorème repose aussi sur le même principe que la preuve du théorème de la vacuité. Il suffit de prouver l'implication directe car sa réciproque est donnée par le lemme précédent. Soit t un terme de profondeur β reconnu par A, $t \xrightarrow{*} e_0$ où e_0 un état de satisfaction de A et $\alpha < \beta$. Si $\beta \leq 2 * \alpha$ alors le théorème est vérifié. Sinon nous allons démontrer qu'il existe un terme t' de profondeur inférieure ou égale à $2 * \alpha$ et supérieure à α reconnu par A. La construction du terme t', en fonction de t, se fait de la manière suivante: t étant de profondeur β alors il existe au moins une suite de sous-termes $(t_0, \ldots, t_\alpha, \ldots, t_{2*\alpha}, \ldots, t_\beta)$ de t telle que $t = t_0 \succ t_1 \succ \cdots \succ t_\beta \in F_0$. Soit $(e_0, \ldots, e_\beta)$ la suite des états de A telle que $\forall i \in [0, \beta]$ $t_i \xrightarrow{*} e_i$. Il existe deux états e_l et e_k de cette suite qui sont égaux avec $\alpha < l < k < 2 * \alpha$. Soit $t' = t[t_l \longleftarrow t_k]$. D'après le lemme 5.2 $t' \xrightarrow{*} e_0$ et la profondeur de t' est inférieure à la profondeur de t. Si la profondeur de t' est inférieure ou égale à $2 * \alpha$ alors le théorème est vérifié. Sinon il suffit de recommencer le même processus sur t' jusqu'à ce qu'un sous-terme de profondeur inférieure ou égale à $2 * \alpha$ soit obtenu.

H. Trad [69] a démontré que le langage reconnu par un automate d'arbres, dont le nombre d'états est α, est infini si et seulement si l'automate reconnaît au moins un terme de profondeur comprise entre α et $3 * \alpha$. Notre résultat est plus efficace et en plus la démonstration est plus simple.

Définition 5.6 *Soient A et B deux automates. $A \times B$ est l'automate produit de A et B. Si $A = (E_A, S_A, P_A)$ et $B = (E_B, S_B, P_B)$ alors $A \times B = (E_A \otimes E_B, S_A \otimes S_B, P)$ où les règles de P sont de la forme:*
- Si $f(e_1, \ldots, e_n) \rightarrow e_0 \in P_A$ et $f(e_1', \ldots, e_n') \rightarrow e_0' \in P_B$ alors $f((e_1, e_1'), \ldots, (e_n, e_n')) \rightarrow (f(e_1, \ldots, e_n), f(e_1', \ldots, e_n')) \in P$.
- Si $e_1 \rightarrow e_2 \in P_A$ et $e_1' \rightarrow e_2' \in P_B$ alors $(e_1, e_1') \rightarrow (e_2, e_2') \in P$.
$\otimes$ est le produit ensembliste.

Lemme 5.4 *Soient A et B deux automates. $L(A) \cap L(B) = L(A \times B)$.*

Le théorème suivant donne une condition nécessaire et suffisante pour que deux automates soient équivalents, ou que leurs langages associés soient égaux.

Théorème 5.4 *Soient A et A' deux automates déterministes dont le nombre d'états est respectivement α et α'. $L(A)$ et $L(A')$ ne sont pas égaux si et seulement si il existe un terme de profondeur inférieure ou égale au produit de α par α' qui est reconnu par l'un et non par l'autre.*

Preuve: La preuve de ce théorème se réduit à la preuve de la condition nécessaire. Soient B et B' les deux automates complémentaires de A et A' respectivement. Il suffit de démonter que l'un des deux langages $L(A) \cap L(B')$ et $L(A') \cap L(B)$ n'est pas vide. Notons que le nombre d'états du complémentaire d'un automate déterministe est égal au nombre d'états de l'automate. Ainsi que le nombre d'états du produit de deux automates est égal au produit du nombre d'états des deux automates.

5.2.2 Grammaire d'arbres

Dans ce paragraphe, nous allons présenter les différentes notions de base du concept de grammaire d'arbres, nécessaire pour les paragraphes qui suivent.

Définition 5.7 *Une grammaire d'arbres G sur F est un triplet (Γ, ξ, Q) tel que:*

- *Γ: ensemble des vocabulaires auxiliaires de G,*
- *ξ: est un élément de Γ appelé axiome de G,*
- *Q: ensemble des règles de dérivation, ou de production, dont la forme est: $\eta_0 \rightarrow f(\eta_1, \ldots, \eta_n)$ où $\eta_0 \in \Gamma$, $\eta_1, \ldots, \eta_n \in T(F, \Gamma)$ et $f \in F_n$.*

Remarque 5.3 *Q peut être considéré comme un système de réécriture de termes dans $T(F \cup \Gamma)$ où les éléments de Γ considérés comme des constantes.*

Dans toute la suite, nous parlerons de grammaire au lieu de grammaire d'arbres sur F, s'il n'y a pas d'ambiguïté.

Convention 5.4
Nous notons $\longrightarrow_Q$ la relation de réduction associée au système de réécriture Q et $\xrightarrow{*}_Q$ la fermeture réflexive-transitive de $\longrightarrow_Q$. $\longrightarrow_Q$ et $\xrightarrow{*}_Q$ seront notées respectivement $\longrightarrow$ et $\xrightarrow{*}$ en abréviation s'il n'y a pas d'ambiguïté sur Q.

Définition 5.8 *Soient t et t' deux termes de $T(F \cup \Gamma)$. $t \longrightarrow_Q t'$ si et seulement si $\exists u \in O(t)$ $\exists \eta \in \Gamma$ tels que $\exists \eta \rightarrow s \in Q$ où $t/u = s$ et $t' = t[u \longleftarrow \eta]$.*

Définition 5.9 *Soit $G = (\Gamma, \xi, Q)$ une grammaire sur F.*

- *Un terme t de $T(F)$ est engendré par G si et seulement si il existe une dérivation de ξ telle que $\xi \xrightarrow{*}_Q t$.*
- *Le sous-ensemble de $T(F)$ des termes engendés par G, noté $L(G)$, est appelé langage d'arbres sur F engendré par G. C'est exactement l'ensemble des formes irréductibles par rapport à Q de ξ et appartenant à $T(F)$.*

Dans toute la suite, nous parlerons de langage au lieu de langage d'arbres sur F, s'il n'y a pas d'ambiguïté.

Comme pour les automates et les grammaires de mots, nous avons:

Lemme 5.5 *Pour tout automate il existe une grammaire associée équivalente, c'est à dire qui engendre exactement le langage reconnu par l'automate.*

La construction de cette grammaire sera donnée dans le quatrième paragraphe. Il est important de noter la grande similitude entre les grammaires et les automates. Les premières insistent sur la génération, les seconds sur la reconnaissance. Les non terminaux des grammaires correspondent aux états des automates et les flèches sont renversées.

5.3 Reconnaissance des formes normales closes

Dans ce paragraphe, nous allons construire un automate, déterministe, qui reconnaît le langage des termes clos en forme normale par rapport à un système de réécriture de termes linéaires à gauche donné. Pour cela, nous allons tout d'abord construire un automate A qui reconnaît le langage des termes clos réductibles. L'automate reconnaissant le langage des termes clos en forme normale est le complémentaire de A.

Soit R un système de réécriture de termes linéaire à gauche quelconque, pour toute la suite.

Remarque 5.4 *R n'est pas supposé convergent, c'est à dire confluent et nœthérien, car nous ne nous intéressons qu'aux membres gauches des règles de R. Nous supposons uniquement que R est linéaire à gauche.*

Rappelons les notations suivantes:

Convention 5.5
$T(F)\downarrow_R$: ensemble des termes clos de $T(F)$ en forme normale par rapport à R.
$T(F)\uparrow^R$: ensemble des termes clos de $T(F)$ réductibles par rapport à R.
$MG(R)$: ensemble des membres gauches des règles de R.
$\Sigma(F)$: ensemble des substitutions closes dans $T(F)$.

5.3.1 Reconnaissance des termes clos réductibles

Dans ce paragraphe, nous allons construire un automate indéterministe avec Λ qui reconnaît le langage des termes clos réductibles. Dans le paragraphe suivant, nous allons réduire l'indéterminisme, s'il y a lieu.

Construction de l'automate indéterministe

Soit A un automate reconnaissant le langage $T(F)\uparrow^R$ des termes clos réductibles par rapport à R. La construction de l'automate $A = (E, S, P)$ se fait en 3 étapes successives:

- La première étape consiste à reconnaître les membres gauches des règles du système de réécriture R. A chacun de ces termes est associé un état de satisfaction. L'association d'un terme à un état ce fait de la manière suivante:

  - à chaque constante c nous associons un nouvel état e_c par une règle de transition $c \rightarrow e_c$.
  - toutes les variables sont transformées en un unique état particulier d'attente noté $e_?$.
  - si $t = f(s_1, \ldots, s_n)$ et si $e_{s_1}, \ldots, e_{s_n}$ sont respectivement les états associés à $s_1, \ldots, s_n$ alors t est associé à un nouvel état e_t par la règle de transition $f(e_{s_1}, \ldots, e_{s_n}) \rightarrow e_t$.
  - nous posons ok comme unique état de satisfaction. Chaque état e_t, avec t le membre gauche d'une règle de R, est relié à l'état ok par une Λ-transition.

  Il n'y a qu'un nombre fini d'états e_t car il n'y a qu'un nombre fini de sous-termes dans les membres gauches.

- La deuxième étape consiste à reconnaître les instances des membres gauches des règles de R. Chacun des états, créé à l'étape précédente, est relié à l'état $e_?$ par une Λ-transition, sauf pour l'état $e_?$ lui même et l'état ok. Puis pour chaque opérateur f de F nous ajoutons la règle de transition $f(e_?, \ldots, e_?) \rightarrow e_?$, qui transmets l'incertitude à l'étage supérieur.

- La troisième étape consiste à reconnaître les termes contenant un sous-terme reconnu par les étapes précédentes. Pour chaque opérateur f non constant, nous ajoutons les règles de transitions $f(e_?, \ldots, e_?, ok, e_?, \ldots, e_?) \rightarrow ok$, qui transmettent le fait d'avoir reconnu un membre gauche de règle à l'étage supérieur.

Ces trois étapes peuvent être formalisées de la manière suivante:

(1) reconnaissance des termes de $MG(R)$:

$$\begin{aligned}
E_1 &= \{\ e_s \mid s = t/u \wedge u \in G(t) \wedge t \in MG(R)\ \} \\
&\oplus \{\ e_?.ok\ \}, \\
S_1 &= \{\ ok\ \}, \\
P_1 &= \{\ x \rightarrow e_? \mid x \in X\ \} \\
&\oplus \{\ f(e_{s_1}, \ldots, e_{s_n}) \rightarrow e_{f(s_1,\ldots,s_n)} \mid n = \text{arité}(f) \wedge \\
&\qquad \forall i \in [1,n] s_i \longrightarrow e_{s_i} \wedge f(s_1, \ldots, s_n) = t/u \wedge u \in G(t) \wedge t \in MG(R)\ \} \\
&\oplus \{\ e_t \rightarrow ok \mid t \in MG(R)\ \}.
\end{aligned}$$

Autrement dit, pour toute variable de tout terme de $MG(R)$ nous lui associons l'unique état $e_?$. Pour chaque terme t de $MG(R)$, pour chaque sous-terme s de t, non réduit à une variable, nous engendrons un nouvel état et une nouvelle règle. L'état ok est le seul état de satisfaction et les états associés aux termes de $MG(R)$ sont reliés à ok par des Λ-transitions.

(2) reconnaissance des instances des termes de $MG(R)$ et des termes qui ne sont pas présents dans $MG(R)$:

$$\begin{aligned}
E_2 &= E_1, \\
S_2 &= S_1, \\
P_2 &= P_1 \\
&\oplus \{\ e \rightarrow e_? \mid e \in E_1 \ominus \{e_?, ok\}\ \} \\
&\oplus \{\ f(\underbrace{e_?, \ldots, e_?}_{\text{arité}(f)}) \rightarrow e_? \mid f \in F\ \}.
\end{aligned}$$

Autrement dit, chaque état peut être vu comme égal à l'état $e_?$, ceci pour reconnaître toutes les instances des termes de $MG(R)$. Un terme est réduit à l'état $e_?$ s'il ne peut pas être réduit autrement, ceci pour reconnaître tous les termes qui ne sont pas présents dans $MG(R)$.

(3) reconnaissance des termes qui contiennent un sous-terme reconnu dans la première étape:

$$\begin{array}{lcl} E_3 & = & E_2, \\ S_3 & = & S_2, \\ P_3 & = & P_2 \\ & \oplus & \{\ f(\underbrace{e_?, \ldots, e_?}_{\text{arité}(f)}, ok, e_?, \ldots, e_?) \rightarrow ok \mid f \in F \ominus F_0\ \}. \end{array}$$

Autrement dit, dès qu'un sous-terme s d'un terme t est reconnu, le terme t tout entier est reconnu.

Cet automate est fini, il n'y a qu'un nombre fini d'états, en effet nous ne considérons que les sous-termes des membres gauches, des règles de R, qui sont en nombre fini.

Remarquons que cet automate est très fortement non déterministe et contient beaucoup de Λ-transition, les $e_t \rightarrow ok$ introduits à la première étape et les $e \rightarrow e_?$ introduits à la deuxième étape.

Remarque 5.5 *Puisque nous considérons habituellement des systèmes de réécritures convergent, nous pouvons réduire le nombre d'états de l'automate A de la manière suivante: transformer tous les états e_t, où t appartient à $MG(R)$, par l'état ok dans toutes les règles de P; puis éliminer ces états de E.*

Remarque 5.6 *Cet automate peut reconnaître aussi les termes R-réductible non clos, en effet, pour reconnaître un terme t non clos il suffit de reconnaître le terme s obtenu par transformation de toutes les variables de t par l'état $e_?$.*

Complexité

Soient R un système de réécriture, F l'ensemble des opérateurs et $A = (E, S, P)$ un automate associé à R, construit par la méthode décrite ci-dessous. La complexité de A peut être donnée par le nombre des états C de E dans le pire des cas. Soient m la profondeur de R, n l'arité maximale de F, p le nombre de règles de R et q le nombre de constantes de F. Au pire des cas, tous les membres gauches de R sont de profondeur m et tous ses sous-termes stricts ont comme racine un opérateur d'arité n, c'est à dire que les constantes et les variables n'apparaissent qu'au niveau n.

$$C = p * \frac{n^m - 1}{n - 1} + q + 1.$$

Exemple

Soit R le système de réécriture suivant:

$$\begin{array}{lcl} R & = & \{\ +(0,y) \rightarrow y \\ & & \quad +(s(0),y) \rightarrow s(y) \\ & & \quad +(s(s(x)),y) \rightarrow s(s(+(x,y)))\ \}, \\ F & = & \{\ 0, s, +\ \}. \\ MG(R) & = & \{\ +(0,y), +(s(0),y), +(s(s(x)),y)\ \}. \end{array}$$

Nous allons construire l'automate $A = (E, S, P)$ reconnaissant le langage $T(F)\uparrow^R$ des termes clos R-réductibles. par la méthode que nous venons de décrire:

(1)

$$
\begin{array}{lcl}
P_1 = \{ & x & \to e_?, \\
& 0 & \to e_0, \\
& +(e_0, e_?) & \to e_1, \\
& s(e_0) & \to e_{s_0}, \\
& +(e_{s_0}, e_?) & \to e_2, \\
& s(e_?) & \to e_s, \\
& s(e_s) & \to e_{ss}, \\
& +(e_{ss}, e_?) & \to e_3, \\
& e_1 & \to ok, \\
& e_2 & \to ok, \\
& e_3 & \to ok \quad \}.
\end{array}
\qquad
\begin{array}{lcl}
E_1 = \{ & e_?, \\
& e_0, \\
& e_1, \\
& e_{s_0}, \\
& e_2, \\
& e_s, \\
& e_{ss}, \\
& e_3, \\
& ok \quad \}
\end{array}
$$

$$S_1 = \{ \ ok \ \}.$$

(2)

$$
\begin{array}{lcl}
E_2 = & E_1, & \\
S_2 = & S_1, & \\
P_2 = & P_1 & \\
\oplus \{ & e_0 & \to e_?, \\
& e_{s_0} & \to e_?, \\
& e_s & \to e_?, \\
& e_{ss} & \to e_?, \\
& e_1 & \to e_?, \\
& e_2 & \to e_?, \\
& e_3 & \to e_?, \\
& 0 & \to e_?, \\
& s(e_?) & \to e_?, \\
& +(e_?, e_?) & \to e_? \quad \}.
\end{array}
$$

(3)

$$
\begin{array}{lcl}
E_3 = & E_2, & \\
S_3 = & S_2, & \\
P_3 = & P_2 & \\
\oplus \{ & s(ok) & \to ok, \\
& +(ok, e_?) & \to ok, \\
& +(e_?, ok) & \to ok \quad \}.
\end{array}
$$

Soit donc l'automate indéterministe avec Λ $A = (E, S, P)$ où:

$$E = \{ \ ok, e_?, e_0, e_{s_0}, e_s, e_{ss}, e_1, e_2, e_3 \ \}.$$
$$S = \{ \ ok \ \}.$$

$$
\begin{array}{rlll}
P = \{ & x & \rightarrow & e_?. \\
& 0 & \rightarrow & e_0. \\
& s(e_0) & \rightarrow & e_{s_0}. \\
& +(e_0, e_?) & \rightarrow & e_1. \\
& s(e_?) & \rightarrow & e_s. \\
& +(e_{s_0}, e_?) & \rightarrow & e_2. \\
& s(e_s) & \rightarrow & e_{ss}. \\
& +(e_{ss}, e_?) & \rightarrow & e_3. \\
& e_1 & \rightarrow & ok, \\
& e_2 & \rightarrow & ok, \\
& e_3 & \rightarrow & ok, \\
& e_0 & \rightarrow & e_?. \\
& e_{s_0} & \rightarrow & e_?, \\
& e_s & \rightarrow & e_?, \\
& e_{ss} & \rightarrow & e_?, \\
& e_1 & \rightarrow & e_?, \\
& e_2 & \rightarrow & e_?, \\
& e_3 & \rightarrow & e_?, \\
& 0 & \rightarrow & e_?, \\
& s(e_?) & \rightarrow & e_?, \\
& +(e_?, e_?) & \rightarrow & e_?, \\
& s(ok) & \rightarrow & ok, \\
& +(ok, e_?) & \rightarrow & ok, \\
& +(e_?, ok) & \rightarrow & ok \quad \}.
\end{array}
$$

Remarque 5.7

- *A est indéterministe car:*
  $s(0) \longrightarrow s(e_0) \longrightarrow e_{s_0}$ *et* $s(0) \longrightarrow s(e_0) \longrightarrow s(e_?) \longrightarrow e_s$ *et* $e_{s_0} \neq e_s$.
- *A est indéterministe avec* Λ *car:* $e_0 \rightarrow e_? \in P$.

La procédure 6.10 décrit cette méthode pour construire un automate reconnaissant le langage des termes clos réductibles par rapport à un système de réécriture linéaire à gauche.

Correction de l'automate indéterministe

La construction est telle que l'automate reconnait les termes qui contiennent une instance d'un membre gauche de règle et eux seuls. Nous allons cependant le démontrer formellement. Avant cela, nous allons présenter trois lemmes utiles pour cette démonstration.

Lemme 5.6 $\forall t \in T(F) \quad t \xrightarrow{\star} e_?$.

Autrement dit, tout terme clos se transforme, par l'automate A, à l'état $e_?$.

Preuve: Par induction structurelle sur les termes.

- induction structurelle de base:
  Soit $t = c$, où $c \in F_0$. $c \longrightarrow e_?$ d'après l'étape (2).
- une étape d'induction structurelle:
  Soient $n \in N$ et $t_1, \ldots, t_n \in T(F)$ tels que $\forall i \in [1, n] t_i \xrightarrow{*} e_?$.
  Soit $f \in F_n$ $f(t_1, \ldots, t_n) \xrightarrow{*} f(e_?, \ldots, e_?)$ d'après (1).
  $f(e_?, \ldots, e_?) \longrightarrow e_?$ d'après l'étape (2).

Lemme 5.7 $\forall t \in T(F)\ \exists u \in O(t) \quad t/u \xrightarrow{*} ok \implies t \xrightarrow{*} ok.$

Autrement dit, si un sous-terme d'un terme clos se transforme, par l'automate A, en l'état de satisfaction ok alors le terme tout entier peut se transformer en l'état ok.

Preuve: Par induction structurelle sur les termes.

- induction structurelle de base:
  évidente.
- une étape d'induction structurelle:
  Soient $n \in N$ et $t_1, \ldots, t_n \in T(F)$ tels que:
  $\forall i \in [1, n] \exists u \in O(t_i) \quad t_i/u \xrightarrow{*} ok \implies t_i \xrightarrow{*} ok.$
  Soient $f \in F_n$ et $u \in O(f(t_1, \ldots t_n))$ tel que: $f(t_1, \ldots, t_n)/u \xrightarrow{*} ok$
  - $u = \varepsilon$: évidente.
  - $u = iv$ où $i \in [1, n]$ et $v \in O(t_i)$: donc $t_i/v \xrightarrow{*} ok$ et par hypothèse $t_i \xrightarrow{*} ok$. D'après le lemme 5.6 nous avons: $\forall i \in [1, n]\ t_i \xrightarrow{*} e_?$.
    Donc $f(t_1, \ldots, t_n) \xrightarrow{*} f(\underbrace{e_?, \ldots, e_?}_{i-1}, ok, e_?, \ldots, e_?)$ d'après l'étape (1)
    et $f(\underbrace{e_?, \ldots, e_?}_{i-1}, ok, e_?, \ldots, e_?) \longrightarrow ok$ d'après l'étape (3).

Lemme 5.8 $\forall t \in T(F)\ \forall e \in E$

$t \xrightarrow{*} e \implies \exists \sigma \in \Sigma(F)\ \exists s \in T(F, X)\ \exists l \in MG(R)\ \exists u \in G(l) \quad t = \sigma(s) \wedge e = e_s \wedge s = l/u.$

Autrement dit, tout terme clos se transforme, par l'automate A, en l'état correspondant à une instance d'un sous-terme du membre gauche d'une règle du système de réécriture.

Preuve: Par induction structurelle sur les termes.

- induction structurelle de base:
  évidente.
- une étape d'induction structurelle:
  Soient $n \in N$ et $t_1, \ldots, t_n \in T(F)$ tels que: $\forall i \in [1, n]$

$t_i \xrightarrow{\star} e_i \Longrightarrow \exists \sigma_i \in \Sigma(F)\ \exists s_i \in T(F,X)\ \exists l_i \in MG(R)\ \exists u_i \in G(l_i)\quad t_i = \sigma_i(s_i)$ et $e_i = e_{s_i}$ et $s_i = l_i/u_i$.
Soient $f \in F_n$ et $e, e_1, \ldots e_n \in E$ tels que:
$f(t_1, \ldots, t_n) \xrightarrow{\star} e$ et $\forall i \in [1,n]\ t_i \xrightarrow{\star} e_i$. Donc $f(t_1 \ldots . t_n) \xrightarrow{\star} f(e_1, \ldots, e_n) \longrightarrow e$. Soit $\sigma = \cup_{i \in [1,n]} \sigma_i$ puisque nous avons supposé que le système de réécriture R est linéaire à gauche. Donc $f(t_1 \ldots . t_n) = f(\sigma_1(s_1) \ldots . \sigma_n(s_n)) = \sigma(f(s_1, \ldots, s_n))$ et
$f(s_1, \ldots, s_n) \xrightarrow{\star} f(e_{s_1}, \ldots, e_{s_n}) \longrightarrow e_{f(s_1, \ldots . s_n)}$, $e = e_{f(s_1 \ldots . s_n)}$ et il existe l membre gauche de R, u occurrence de l tels que $f(s_1 \ldots . s_n) = l/u$ d'après l'étape (1).

Théorème 5.5 $L(A) = T(F)\uparrow^R$.

Preuve:

- $T(F)\uparrow^R \subseteq L(A)$ en effet:
  Soit t un terme clos R-réductible: $\exists l \in MG(R)\ \ \exists u \in G(t)\ \ \exists \sigma \in \Sigma(F)$ | $t/u = \sigma(l)$. Soit $l = \Delta(x_1, \ldots, x_n)$ où $x_1, \ldots, x_n$ la suite des variables présent dans le terme l et Δ son "contexte". Or par construction nous avons:
  $\forall i \in [1,n]\ \ x_i \longrightarrow e_?$ et $l = \Delta(x_1, \ldots, x_n) \xrightarrow{\star} \Delta(e_?, \ldots . e_?) \xrightarrow{\star} ok$. Nous posons $\sigma(x_1) = s_1, \ldots, \sigma(x_n) = s_n$. Or d'après le lemme 5.6 nous avons: $\forall i \in [1,n]\ \ s_i \xrightarrow{\star} e_?$.
  Donc $\sigma(l) = \Delta(s_1, \ldots, s_n) \xrightarrow{\star} \Delta(e_?, \ldots, e_?) \xrightarrow{\star} ok$. Donc $t/u \xrightarrow{\star} ok$ et d'après le lemme 5.7 $t \xrightarrow{\star} ok$ donc t est reconnu par A.
- $L(A) \subseteq T(F)\uparrow^R$ en effet:
  Soit t un terme reconnu par l'automate A: $t \xrightarrow{\star} ok$. Soit v la plus grande occurrence de t telle que $t/v \xrightarrow{\star} ok$. Donc d'après le lemme 5.8 $\exists \sigma \in \Sigma(F)\ \exists s \in T(F,X)\ \exists l \in MG(R)\ \exists u \in G(l)\quad t/v = \sigma(s)$ et $ok = e_s$ et $s = l/u$. Donc $s \longrightarrow ok$ et $u = \varepsilon$ d'où $s = l$ et par suite $t/v = \sigma(l)$. Donc t/v est R-réductible et par suite t est R-réductible.

5.3.2 Réduction de l'indéterminisme

Construction de l'automate déterminisme

Soit $A = (E, S, P)$ un automate indéterministe avec Λ, nous allons chercher l'automate $A' = (E', S', P')$ déterministe qui reconnaît le même langage que A. La réduction de l'indéterminisme consiste à regrouper les états "équivalents", ainsi les états de A' sont des ensembles d'états de A. Nous allons étendre la notion de termes sur des ensembles

ainsi que les règles lors de la construction de A'. Un automate étant une $F \cup E$-algèbre, l'interprétation de l'ensemble des symboles de fonctions F dans A et A' sont différents. Pour alléger les notations, nous allons prendre les mêmes symboles de fonctions pour A et A' s'il n'y a pas d'ambiguïté.

La construction la plus simple de l'automate déterministe $A_d = (E_d, S_d, P_d)$, à partir de l'automate $A = (E, S, P)$, se fait de la manière suivante:

$$\begin{array}{lcll} E_d & = & \mathcal{P}(E), & \\ P_d & = & \{ \ f(q_1,\ldots,q_n) \rightarrow q_0 \mid & f \in F \ \wedge \ n = arité(f) \ \wedge \\ & & & \exists (e_1,\ldots,e_n) \in q_1 \otimes \cdots \otimes q_n \ \exists e_0 \in q_0 \ tq \\ & & & f(e_1,\ldots,e_n) \xrightarrow{\star}_P e_0 \ \}. \\ S_d & = & \{ \ q \mid \exists e \in S \ tq \ e \in q \ \}. & \end{array}$$

Les états de A_d sont des ensembles qui contiennent des états de A. $\mathcal{P}(E)$ est l'ensemble des parties de E.

Cette méthode engendre beaucoup d'états qui ne seront jamais accessibles par l'automate. Nous allons décrire une méthode qui tend à minimiser l'ensemble des états en ne prenant que les états accessibles. La construction de l'automate déterministe réduit $A' = (E', S', P')$ se fait par itération. Les automates construits sont des "restrictions" de l'automate A_d.

- L'automate initial A'_0 est $(\emptyset, \emptyset, \emptyset)$ c'est à dire, qu'il n'y a pas d'état et donc pas de transition.

- L'itération consiste à ajouter de nouveaux états accessibles à partir des états de l'automate précédent ainsi que les transitions associées.
  $q_0 \in E'_{i+1}$ si et seulement si il existe $(q_1,\ldots,q_n) \in E'^n_i$ et f d'arité n tel que $f(q_1,\ldots,q_n) \xrightarrow{\star}_{P_d} q_0$.
  L'itération s'arrête si $(E'_{i+1}, P'_{i+1}) = (E'_i, P'_i)$.

- A la fin nous posons comme ensemble des états de satisfaction, l'ensemble des états qui contiennent au moins un état de satisfaction de A.

Nous allons maintenant "oublier" l'automate A_d et construire directement A' à partir de A. La construction de A' se fait formellement en 3 étapes successives.

[1] initialisation de l'automate A':

$$\begin{array}{lcl} E'_0 & = & \emptyset, \\ P'_0 & = & \emptyset. \end{array}$$

[2] itération de regroupement des états de A:

$$\begin{array}{lcll} P'_{i+1} & = & P'_i & \\ & \oplus & \{ \ f(q_1,\ldots,q_n) \rightarrow q_0 \mid f \in F \wedge n = arité(f) \ \wedge \ q_1,\ldots,q_n \in E'_i \ \wedge \\ & & q_0 = \{ \ e_0 \mid \exists \ \ e_1 \in q_1,\ldots,e_n \in q_n, e'_0 \in E, e'_1 \in E,\ldots,e'_n \in E \ tq \\ & & e_1 \xrightarrow{\star}_P e'_1 \wedge \cdots \wedge e_n \xrightarrow{\star}_P e'_n \wedge f(e'_1,\ldots,e'_n) \longrightarrow_P e'_0 \wedge e'_0 \xrightarrow{\star}_P e_0 \ \} \ \}. \\ E'_{i+1} & = & E'_i & \\ & \cup & \{ \ q_0 \mid f(q_1,\ldots,q_n) \rightarrow q_0 \in P'_{i+1} \ \}. & \end{array}$$

L'itération s'arrête quand aucun nouvel état n'est créé.

A la fin de cette itération, nous posons:

$E' = E'_i$.

$P' = P'_i$.

[3] états de satisfaction:

$S' = \{\ q \in E' \mid ok \in q\ \}$.

Autrement dit, tout ensemble d'états contenant ok, seul état de satisfaction de A, est un état de satisfaction pour A'.

Remarque 5.8 *A la première étape de l'itération, seule les constantes interviennent dans* P'_1.

$$P'_1 = \{\ c \to q_0 \mid c \in F_0 \ \wedge\ q_0 = \{e_0 \mid c \xrightarrow{*}_P e_0\}\ \}.$$

Dans les autres étapes de l'itération, les constantes n'interviennent plus.

Remarque 5.9 *Le processus termine car* E'_i *croit et* $card(E'_i)$ *est borné par* $2^{card(E)}$.

Remarque 5.10 *Réduire l'indéterminisme d'un automate* $A = (E, S, P)$ *revient à chercher un automate* $A' = (E', S', P')$ *qui reconnaît le même langage que* A *et dont le système de réécriture* P' *est "sans superposition", ce qui est plus fort que la confluence.*

Exemple

Reprenons l'exemple précédent. Nous allons construire l'automate déterministe réduit $A' = (E', S', P')$ reconnaissant le même langage $T(F)\uparrow^R$ que A, par la méthode que nous venons de décrire:

[1]

$E'_0 = \emptyset$,

$P'_0 = \emptyset$.

[2]

$$\begin{array}{lllll} P'_1 = & P'_0 & & E'_1 = & E'_0 \\ \oplus\ \{ & 0 \to \{e_?, e_0\}\ \} & & \oplus\ \{ & \{e_?, e_0\}\ \}. \end{array}$$

$$\begin{array}{lllll} P'_2 = & P'_1 & & E'_2 = & E'_1 \\ \oplus\ \{ & s(\{e_?, e_0\}) & \to \{e_?, e_{s_0}, e_s\}. & \oplus\ \{ & \{e_?. e_{s_0}. e_s\}, \\ & +(\{e_?, e_0\}, \{e_?. e_0\}) & \to \{ok, e_?. e_1\}\ \} & & \{ok. e_?. e_1\}\ \}. \end{array}$$

Pour réduire la complexité, q marque un état quelconque de E'_{i-1}.

$$
\begin{array}{lcl}
P'_3 & = & P'_2 \\
 & \oplus & \{\ s(\{e_?, e_{s_0}, e_s\}) \rightarrow \{e_?, e_s, e_{ss}\}, \\
 & & \ \ s(\{ok, e_?, e_1\}) \rightarrow \{ok, e_?, e_s\}, \\
 & & \ \ +(\{e_?, e_{s_0}, e_s\}, q) \rightarrow \{ok, e_?, e_2\}, \\
 & & \ \ +(\{e_?, e_0\}, \{e_?, e_{s_0}, e_s\}) \rightarrow \{ok, e_?, e_1\}, \\
 & & \ \ +(\{ok, e_?, e_1\}, q) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{e_?, e_0\}, \{ok, e_?, e_1\}) \rightarrow \{ok, e_?, e_1\}\ \}.
\end{array}
\qquad
\begin{array}{lcl}
E'_3 & = & E'_2 \\
 & \oplus & \{\ \{e_?, e_s, e_{ss}\}, \\
 & & \ \ \{ok, e_?, e_s\}, \\
 & & \ \ \{ok, e_?, e_2\}, \\
 & & \ \ \{ok, e_?\}\ \}
\end{array}
$$

$$
\begin{array}{lcl}
P'_4 & = & P'_3 \\
 & \oplus & \{\ s(\{e_?, e_s, e_{ss}\}) \rightarrow \{e_?, e_s, e_{ss}\}, \\
 & & \ \ s(\{ok, e_?, e_s\}) \rightarrow \{ok, e_?, e_s, e_{ss}\}, \\
 & & \ \ s(\{ok, e_?, e_2\}) \rightarrow \{ok, e_?, e_s\}, \\
 & & \ \ s(\{ok, e_?\}) \rightarrow \{ok, e_?, e_s\}, \\
 & & \ \ +(\{e_?, e_s, e_{ss}\}, q) \rightarrow \{ok, e_?, e_3\}, \\
 & & \ \ +(\{e_?, e_0\}, \{e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?, e_1\}, \\
 & & \ \ +(\{e_?, e_{s_0}, e_s\}, \{e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?, e_2\}, \\
 & & \ \ +(\{ok, e_?, e_1\}, \{e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{ok, e_?, e_s\}, q) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{e_?, e_0\}, \{ok, e_?, e_s\}) \rightarrow \{ok, e_?, e_1\}, \\
 & & \ \ +(\{e_?, e_{s_0}, e_s\}, \{ok, e_?, e_s\}) \rightarrow \{ok, e_?, e_2\}, \\
 & & \ \ +(\{ok, e_?, e_1\}, \{ok, e_?, e_s\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{ok, e_?, e_2\}, q) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{e_?, e_0\}, \{ok, e_?, e_2\}) \rightarrow \{ok, e_?, e_1\}, \\
 & & \ \ +(\{e_?, e_{s_0}, e_s\}, \{ok, e_?, e_2\}) \rightarrow \{ok, e_?, e_2\}, \\
 & & \ \ +(\{ok, e_?, e_1\}, \{ok, e_?, e_2\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{ok, e_?\}, q) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{e_?, e_0\}, \{ok, e_?\}) \rightarrow \{ok, e_?, e_1\}, \\
 & & \ \ +(\{e_?, e_{s_0}, e_s\}, \{ok, e_?\}) \rightarrow \{ok, e_?, e_2\}, \\
 & & \ \ +(\{ok, e_?, e_1\}, \{ok, e_?\}) \rightarrow \{ok, e_?\}\ \}.
\end{array}
\qquad
\begin{array}{lcl}
E'_4 & = & E'_3 \\
 & \oplus & \{\ \\
 & & \ \ \{ok, e_?, e_s, e_{ss}\}, \\
 & & \ \ \{ok, e_?, e_3\}\ \}
\end{array}
$$

$$
\begin{array}{lcl}
P'_5 & = & P'_4 \\
 & \oplus & \{\ s(\{ok, e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?, e_s, e_{ss}\}, \\
 & & \ \ s(\{ok, e_?, e_3\}) \rightarrow \{ok, e_?, e_s\}, \\
 & & \ \ +(\{ok, e_?, e_s, e_{ss}\}, q) \rightarrow \{ok, e_?, e_3\}, \\
 & & \ \ +(\{e_?, e_0\}, \{ok, e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?, e_1\}, \\
 & & \ \ +(\{e_?, e_{s_0}, e_s\}, \{ok, e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?, e_2\}, \\
 & & \ \ +(\{ok, e_?, e_1\}, \{ok, e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{e_?, e_s, e_{ss}\}, \{ok, e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?, e_3\}, \\
 & & \ \ +(\{ok, e_?, e_s\}, \{ok, e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{ok, e_?, e_2\}, \{ok, e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{ok, e_?\}, \{ok, e_?, e_s, e_{ss}\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{ok, e_?, e_3\}, q) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{e_?, e_0\}, \{ok, e_?, e_3\}) \rightarrow \{ok, e_?, e_1\}, \\
 & & \ \ +(\{e_?, e_{s_0}, e_s\}, \{ok, e_?, e_3\}) \rightarrow \{ok, e_?, e_2\}, \\
 & & \ \ +(\{ok, e_?, e_1\}, \{ok, e_?, e_3\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{e_?, e_s, e_{ss}\}, \{ok, e_?, e_3\}) \rightarrow \{ok, e_?, e_3\}, \\
 & & \ \ +(\{ok, e_?, e_s\}, \{ok, e_?, e_3\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{ok, e_?, e_2\}, \{ok, e_?, e_3\}) \rightarrow \{ok, e_?\}, \\
 & & \ \ +(\{ok, e_?\}, \{ok, e_?, e_3\}) \rightarrow \{ok, e_?\}\ \}.
\end{array}
\qquad
\begin{array}{lcl}
E'_5 & = & E'_4
\end{array}
$$

$E'_6 = E'_5$.
$P'_6 = P'_5$.
Donc
$E' = E'_5$,
$P' = P'_5$.

[3]

$$S' = \{ \{ok, e_?\}, \{ok, e_?, e_s\}, \{ok, e_?, e_s, e_{ss}\}, \{ok, e_?, e_1\}, \{ok, e_?, e_2\}, \{ok, e_?, e_3\} \}.$$

Soit donc l'automate déterministe $A' = (E', S', P')$ où:

$$\begin{array}{rcl}
E' & = & \{ \{e_?, e_0\}, \{e_?, e_{s_0}, e_s\}, \{e_?, e_s, e_{ss}\}, \\
 & & \{ok, e_?\}, \{ok, e_?, e_s\}, \{ok, e_?, e_s, e_{ss}\}, \{ok, e_?, e_1\}, \{ok, e_?, e_2\}, \{ok, e_?, e_3\} \}, \\
S' & = & \{ \{ok, e_?\}, \{ok, e_?, e_s\}, \{ok, e_?, e_s, e_{ss}\}, \{ok, e_?, e_1\}, \{ok, e_?, e_2\}, \{ok, e_?, e_3\} \}, \\
P' & = & \{ \begin{array}{lcll}
0 & \rightarrow & \{e_?, e_0\}, & \\
s(\{e_?, e_0\}) & \rightarrow & \{e_?, e_{s_0}, e_s\}, & \\
s(\{e_?, e_{s_0}, e_s\}) & \rightarrow & \{e_?, e_s, e_{ss}\}, & \\
s(\{e_?, e_s, e_{ss}\}) & \rightarrow & \{e_?, e_s, e_{ss}\}, & \\
s(\{ok, e_?\}) & \rightarrow & \{ok, e_?, e_s\}, & \\
s(\{ok, e_?, e_s\}) & \rightarrow & \{ok, e_?, e_s, e_{ss}\}, & \\
s(\{ok, e_?, e_s, e_{ss}\}) & \rightarrow & \{ok, e_?, e_s, e_{ss}\}, & \\
s(\{ok, e_?, e_1\}) & \rightarrow & \{ok, e_?, e_s\}, & \\
s(\{ok, e_?, e_2\}) & \rightarrow & \{ok, e_?, e_s\}, & \\
s(\{ok, e_?, e_3\}) & \rightarrow & \{ok, e_?, e_s\}, & \underline{\forall q \in E'.} \\
+(\{e_?, e_0\}, q) & \rightarrow & \{ok, e_?, e_1\}, & \\
+(\{e_?, e_{s_0}, e_s\}, q) & \rightarrow & \{ok, e_?, e_2\}, & \\
+(\{e_?, e_s, e_{ss}\}, q) & \rightarrow & \{ok, e_?, e_3\}, & \\
+(\{ok, e_?\}, q) & \rightarrow & \{ok, e_?\}, & \\
+(\{ok, e_?, e_s\}, q) & \rightarrow & \{ok, e_?\}, & \\
+(\{ok, e_?, e_s, e_{ss}\}, q) & \rightarrow & \{ok, e_?, e_3\}, & \\
+(\{ok, e_?, e_1\}, q) & \rightarrow & \{ok, e_?\}, & \\
+(\{ok, e_?, e_2\}, q) & \rightarrow & \{ok, e_?\}, & \\
+(\{ok, e_?, e_3\}, q) & \rightarrow & \{ok, e_?\} & \}.
\end{array}
\end{array}$$

La procédure 6.12 décrit cette méthode pour calculer l'automate déterministe réduit.

Correction de l'automate déterministe

Avant de démontrer la correction de l'automate déterministe A', nous allons présenter un lemme utile pour cette démonstration.

Lemme 5.9

$$\forall t \in T(F)\ \forall q \in E' \quad t \xrightarrow{\star}_{P'} q \iff q = \{ e \mid e \in E \wedge t \xrightarrow{\star}_{P} e \}.$$

Autrement dit, tout terme clos se transforme, par l'automate A', vers l'ensemble qui contient tous les états sources du terme par l'automate A et seulement ces états.

Preuve: Par induction structurelle sur les termes.

- induction structurelle de base:
  soit $t = c$ où $c \in F_0$
  $c \xrightarrow{*}_{P'} q$ où, par définition de l'automate A',
  $q = \{\ e \mid e \in E \wedge c \xrightarrow{*}_P e\ \}$.
- une étape d'induction structurelle:
  Soient $n \in N$, $t_1, \ldots, t_n \in T(F)$ et $q_1, \ldots, q_n \in E'$ tels que:
  $\forall i \in [1,n] t_i \xrightarrow{*}_{P'} q_i \iff q_i = \{\ e \mid e \in E \wedge t_i \xrightarrow{*}_P e\ \}$.

- $t \xrightarrow{*}_{P'} q \implies q = \{\ e \mid e \in E \wedge t \xrightarrow{*}_P e\ \}$:
  + $\{\ e \mid e \in E \wedge t \xrightarrow{*}_P e\ \} \subseteq q$:
    Soient $f \in F_n$ et $q_0 \in E'$ tels que:
    $f(t_1, \ldots, t_n) \xrightarrow{*}_{P'} f(q_1, \ldots, q_n) \longrightarrow'_P q_0$.
    Soient $e_1, \ldots, e_n \in E$ tels que: $\forall i \in [1,n] t_i \xrightarrow{*}_P e_i$.
    Or par hypothèse de récurrence $\forall i \in [1,n] e_i \in q_i$.
    Soit $e_0 \in E$ tel que:
    $f(t_1, \ldots, t_n) \xrightarrow{*}_P f(e_1, \ldots, e_n) \longrightarrow_P e_0$.
    Donc $e_0 \in q_0$ par construction de l'automate A'.
  + $q \subseteq \{\ e \mid e \in E \wedge t \xrightarrow{*}_P e\ \}$:
    Soient $f \in F_n$ et $q_0 \in E'$ tels que:
    $f(t_1, \ldots, t_n) \xrightarrow{*}_{P'} f(q_1, \ldots, q_n) \longrightarrow'_P q_0$.
    Soit $e_0 \in q_0$, par construction de l'automate A':
    $\exists e_1 \in q_1, \ldots, e_n \in q_n$ tels que: $\exists e'_1, \ldots, e'_n \in E$ tels que:
    $e_1 \xrightarrow{*}_P e'_1, \ldots, e_n \xrightarrow{*}_P e'_n, f(e'_1, \ldots, e'_n) \longrightarrow_P e'_0$ *et*
    $e'_0 \xrightarrow{*}_P e_0$ donc $f(e_1, \ldots, e_n) \xrightarrow{*}_P e_0$.
    Or $\forall i \in [1,n] t_i \xrightarrow{*}_P e_i$ car $e_i \in q_i$ et
    $q_i \subseteq \{\ e \mid e \in E \wedge t_i \xrightarrow{*}_P e_i\ \}$.
    En somme $f(t_1, \ldots, t_n) \xrightarrow{*}_P e_0$.
- $q = \{\ e \mid e \in E \wedge t \xrightarrow{*}_P e\ \} \implies t \xrightarrow{*}_{P'} q$:
  Soient $f \in F_n$ et $q = \{\ e \mid e \in E \wedge f(t_1, \ldots, t_n) \xrightarrow{*}_P e\ \}$. Soit $q' \in E'$ tel que: $f(t_1, \ldots, t_n) \xrightarrow{*}_{P'} q'$.

  + $q \subseteq q'$:
    Soit $e \in q$ donc $f(t_1, \ldots, t_n) \xrightarrow{*}_P e$ et par suite $\exists e_1, \ldots, e_n$ tels que $\forall i \in [1,n] t_n \xrightarrow{*}_P e_i$ et $f(e_1, \ldots, e_n) \xrightarrow{*}_P e$. Or $\forall i \in [1,n] e_i \in q_i$ donc $e \in q'$.
  + $q' \subseteq q$:
    Soit $e \in q'$ $f(t_1, \ldots, t_n) \xrightarrow{*}_{P'} f(q_1, \ldots, q_n) \longrightarrow'_P q'$.
    Par construction de l'automate A'. $\exists e_1 \in q_1 \ldots e_n \in q_n$

tels que: $\exists e'_1, \ldots, e'_n \in E$ tels que:
$e_1 \xrightarrow{*}_P e'_1, \ldots, e_n \xrightarrow{*}_P e'_n, f(e'_1, \ldots, e'_n) \longrightarrow_P e'$ et $e' \xrightarrow{*}_P e$
donc $f(e_1, \ldots, e_n) \xrightarrow{*}_P e$.
Or $\forall i \in [1, n] e_i \in q_i$ donc $t_i \xrightarrow{*}_P e_i$.
Donc $f(t_1, \ldots, t_n) \xrightarrow{*}_P f(e_1, \ldots, e_n) \xrightarrow{*}_P e$.
Donc $f(t_1, \ldots, t_n) \xrightarrow{*}_P e$ et par suite $e \in q$.

Corollaire 5.1 *A' est déterministe.*

Preuve: Soient $t \in T(F)$ et $q, q' \in E'$ tels que $t \xrightarrow{*}_{P'} q$ et $t \xrightarrow{*}_{P'} q'$. D'après le lemme précédent, nous avons: $q = \{\ e \mid e \in E \wedge t \xrightarrow{*}_P e\ \}$ et $q' = \{\ e \mid e \in E \wedge t \xrightarrow{*}_P e\ \}$ donc $q = q'$.

Corollaire 5.2 $L(A') = L(A)$.

Preuve:

- $L(A') \subseteq L(A)$ en effet:
  Soit t un terme reconnu par A': $t \xrightarrow{*}_{P'} q$ où $q \in S'$. D'après le lemme précédent, nous avons: $q = \{\ e \mid e \in E \wedge t \xrightarrow{*}_P e\ \}$. Or $ok \in q$ car $q \in S'$ donc $t \xrightarrow{*}_P ok$ et par suite t est reconnu par A.
- $L(A) \subseteq L(A')$ en effet:
  Soit t un terme reconnu par A: $t \xrightarrow{*}_P ok$. Soit $q \in E'$ tel que: $t \xrightarrow{*}_{P'} q$. D'après le lemme précédent, nous avons:
  $q = \{\ e \mid e \in E \wedge t \xrightarrow{*}_P e\ \}$. Or $t \xrightarrow{*}_P ok$ donc $ok \in q$ ce qui implique que $q \in S'$ et par suite t est reconnu par A'.

5.4 Génération des formes normales closes

Dans ce paragraphe, nous allons construire une grammaire d'arbres qui engendre le langage d'arbres des termes clos en forme normale par rapport à un système de réécriture linéaire à gauche donné, à partir de l'automate qui le reconnaît. Cette construction est simple d'après l'isomorphisme entre automates et grammaires d'arbres définies auparavant. Le lecteur est invité à le garder à l'esprit dans ce qui va suivre.

5.4.1 Construction de la grammaire

Considérons, pour ce qui suit, R un système de réécriture de terme linéaire à gauche. Soient $T(F)\downarrow_R$ le langage régulier des termes clos en forme normale par rapport à

R et $A = (E, S, P)$ l'automate déterministe reconnaissant $T(F)\downarrow_R$, comme nous l'avons construit dans le paragraphe précédent.

Soit $T(F)$ le langage algébrique des termes clos sur F. Une grammaire $G = (\Gamma, \xi, Q)$ engendrant $T(F)$ peut être défini de la façon suivante:

$\Gamma = \{ \xi \}$,

$Q = \{ \xi \rightarrow f(\underbrace{\xi, \ldots, \xi}_{i}) \mid f \in F \text{ où } i = \text{ arité } (f) \}$.

La grammaire $G = (\Gamma, \xi, Q)$ engendrant le langage $T(F)\downarrow_R$ peut être construite de la manière suivante:

$\Gamma = \{ \xi \} \oplus E$,

$Q = \{ \xi \rightarrow e \mid e \in S \}$

$\oplus \{ e \rightarrow f(e_1, \ldots, e_n) \mid f(e_1, \ldots, e_n) \rightarrow e \in P \}$.

Avant de démontrer la correction de la grammaire G, nous allons donner deux exemples de construction de cette grammaire.

5.4.2 Exemples

Reprenons l'exemple précédent. Nous posons:

$OK = \{ \{ok, e_?\}, \{ok, e_?, e_s\}, \{ok, e_?, e_s, e_{ss}\}, \{ok, e_?, e_1\}, \{ok, e_?, e_2\}, \{ok, e_?, e_3\} \}$,

$ZERO = \{e_?, e_0\}$,

$UN = \{e_?, e_{s_0}, e_s\}$,

$SS = \{e_?, e_s, e_{ss}\}$.

Exemple 5.1 *Nous allons construire la grammaire $G' = (\Gamma', \xi', Q')$ engendrant le langage $T(F)\uparrow^R$ qui est reconnu par l'automate $A' = (E', S', P')$.*

$$
\begin{array}{lllll}
\Gamma' = & \{ \xi' \} & \oplus & \{ OK, ZERO, UN, SS \}, & \\
Q' = \{ & \xi' & \rightarrow & OK, & \\
 & OK & \rightarrow & +(OK, ZERO)/+(ZERO, OK)/ & \\
 & & & +(OK, UN)/+(UN, OK)/ & \\
 & & & +(OK, SS)/+(SS, OK)/ & \\
 & & & +(ZERO, ZERO)/+(ZERO, UN)/ & \\
 & & & +(UN, ZERO)/+(ZERO, SS)/ & \\
 & & & +(SS, ZERO)/+(UN, UN)/ & \\
 & & & +(UN, SS)/+(SS, UN)/ & \\
 & & & +(SS, SS)/+(OK, OK)/s(OK), & \\
 & ZERO & \rightarrow & 0, & \\
 & UN & \rightarrow & s(ZERO), & \\
 & SS & \rightarrow & s(UN)/s(SS) \}. &
\end{array}
$$

Exemple 5.2 *Nous allons construire la grammaire $G'' = (\Gamma'', \xi'', Q'')$ engendrant le langage $T(F)\downarrow_R$ qui est reconnu par l'automate $A'' = (E'', S'', P'')$.*

L'automate $A'' = (E'', S'', P'')$ *est le complémentaire de* $A' = (E', S', P')$.

$$\begin{array}{lcll} \Gamma'' & = & \{\ \xi''\ \} & \oplus \ \{\ OK, ZERO, UN, SS\ \}. \\ Q'' & = & \{\ \xi'' & \rightarrow\ ZERO/UN/SS, \\ & & ZERO & \rightarrow\ 0, \\ & & UN & \rightarrow\ s(ZERO). \\ & & SS & \rightarrow\ s(UN)/s(SS)\ \}. \end{array}$$

La procédure 6.13 décrit cette méthode pour construire la grammaire engendrant le langage reconnaissable par un automate d'arbres donné.

5.4.3 Correction de la grammaire

Soient $A = (E, S, P)$ un automate déterministe reconnaissant $T(F)\downarrow_R$ et $G = (\Gamma, \xi, Q)$ la grammaire construite comme précédemment.

Avant de démontrer la correction de la grammaire G, nous allons présenter un lemme utile pour cette démonstration.

Lemme 5.10

$\forall t \in T(F)\ \forall e \in E \quad t \xrightarrow{\star}_P e \iff e \xrightarrow{\star}_Q t.$

Preuve: Par induction structurelle sur les termes.

- induction structurelle de base:
  Soit $t = c$ où $c \in F_0$. Par construction de la grammaire G:
  $c \longrightarrow_P e \iff e \longrightarrow_Q c$ car l'automate A est déterministe.
- une étape d'induction structurelle:
  Soient $n \in N$ et $t_1, \ldots, t_n \in T(F), e_1, \ldots, e_n \in E$ tels que:
  $\forall i \in [1, n] t_i \xrightarrow{\star}_P e_i \iff e_i \xrightarrow{\star}_Q t_i$.

  - $\forall f \in F_n\ \forall e \in E$
    $f(t_1, \ldots, t_n) \xrightarrow{\star}_P e \implies e \xrightarrow{\star}_Q f(t_1, \ldots, t_n)$.
    Par construction de l'automate A:
    $f(t_1, \ldots, t_n) \xrightarrow{\star}_P f(e_1, \ldots, e_n)$ et $f(e_1, \ldots, e_n) \longrightarrow_P e$
    donc $e \longrightarrow_Q f(e_1, \ldots, e_n)$
    et $e \xrightarrow{\star}_Q f(t_1, \ldots, t_n)$
    par construction de la grammaire G.
  - $\forall f \in F_n\ \forall e \in E$
    $e \xrightarrow{\star}_Q f(t_1, \ldots, t_n) \implies f(t_1, \ldots, t_n) \xrightarrow{\star}_P e$.
    Par construction de l'automate A:
    $f(t_1, \ldots, t_n) \xrightarrow{\star}_P f(e_1, \ldots, e_n)$
    et $e \longrightarrow_Q f(e_1, \ldots, e_n)$.

Avec les notations précédentes, nous avons:

Théorème 5.6 $L(G) = L(A)$.

Autrement dit, $\forall t \in T(F)$

$$\exists e \in S\ \xi \xrightarrow{*}_Q e \xrightarrow{*}_Q t \iff \exists e \in S\ \xi \xrightarrow{*}_Q t \xrightarrow{*}_P e.$$

Preuve:

- $L(G) \subseteq L(A)$.
  Soit t un terme engendré par la grammaire G: $\xi \xrightarrow{*}_Q t$.
  Par construction de la grammaire G, il existe $e \in S$ tel que:
  $\xi \longrightarrow_Q e \xrightarrow{*}_Q t$.
  D'après le lemme précédent, nous avons: $t \xrightarrow{*}_P e$. Or $e \in S$ donc t est reconnu par l'automate A.
- $L(A) \subseteq L(G)$.
  Soit t un terme reconnu par l'automate A:
  $\exists e \in S$ tel que $t \xrightarrow{*}_P e$.
  D'après le lemme précédent, nous avons: $e \xrightarrow{*}_Q t$. Or $e \in S$ donc $\xi \longrightarrow_Q e$ par construction de la grammaire G, et par suite t est engendré par la grammaire G.

6
REVE

Dans le présent chapitre nous allons présenter la partie réalisation de notre travail. La première partie de ce chapitre est consacrée à une présentation général du système REVE. Dans la deuxième partie nous décrivons les différentes procédures des principaux résultats de notre étude. A la fin nous donnons une session complète de cette réalisation dans le logiciel REVE.

6.1 Introduction

Le système REVE est développé en collaboration entre l'équipe EURECA du CRIN et le groupe CSL de MIT. Les différentes versions de ce système sont:

REVE1 est le prototype du système. Il contient principalement la procédure de complétion de Knuth et Bendix et est écrit par P. Lescanne à MIT.

REVE2 est la version courante la plus distribuée. L'originalité de cette version est l'implantation d'une nouvelle version de la procédure de complétion avec une preuve de terminaison automatique. Il offre un interfaçage très agréable. Les axiomes autorisés sont des égalités entre des termes éventuellement avec des variables implicitement quantifiées universellement.

REVEUR3 batit à partir de REVE2, permet la complétion de systèmes de réécriture équationnels. L'ensemble des axiomes est divisé en un ensemble d'axiomes orientables en règles de réécriture et un ensemble d'axiomes utilisés comme des équations. Ce qui permet de prendre en compte des théories qui ont par exemple la propriété de commutativité, axiome qui ne peut pas être orienté sans perdre la propriété de terminaison du système de réécriture associé.

REVEUR4 construit à partir de REVE2, permet la complétion de systèmes de réécriture conditionnels. Dans cette version, les axiomes sont toujours des égalités mais certains peuvent être conditionnés par des conditions mais qui doivent être eux aussi des égalités.

Notre réalisation est restreinte aux systèmes de réécriture de termes. Nous allons présenter dans la suite le système REVE2 où nous avons monté notre implantation.

REVE, [47] [17], est un laboratoire qui manipule et produit des systèmes de réécriture à partir d'un ensemble d'axiomes. Il lit les axiomes d'une spécification, entrée par l'utilisateur, et produit les règles du système de réécriture de termes associé. Il utilise la procédure de complétion de Knuth et Bendix pour assuré la propriété de confluence. La terminaison est vérifiée automatiquement par l'une des 3 méthodes suivantes: l'ordre fondé sur les interprétations polynomiales, l'orde récursif de décomposition avec statut et l'ordre récursif sur les chemins avec statut.

REVE permet de faire des preuves dans la théorie équationnelle en utilisant les techniques de réécriture dans les environements de spécification algébrique. Nous l'avons étondu aux preuves dans la théorie inductive. Il permet également le test de la complétude relative d'une spécification algébrique. En fin il permet de construire l'automate et la grammaire du domaine de l'algèbre initiale d'une vartiété équationnelle.

L'écriture de REVE dans le langage CLU [49], langage modulaire des types abstraits de données, lui a conféré une très grande souplesse et portabilité. La modularité du système facilite sa compréhension et l'intégration des nouvelles fonctionnalités. Durant toute notre implantation, nous avons essayé de maintenir ces qualités pour qu'il reste un laboratoire de réécriture évolutive.

REVE est un grand logiciel contenant près de 30000 lignes d'instructions écrites en langage CLU. Notre contribution s'élève à plus d'un tièrs. Il est opérationnel sur l'ordinateur VAX et SUN sous le système d'exploitation UNIX.

6.2 Mise en œuvre

Nous allons présenter dans cette partie les principales procédures de notre réalisation. Ces procédures sont écrites dans un langage proche de CLU.

6.2.1 Preuve de théorèmes inductifs

Nous allons décrire dans ce paragraphe la réalisation de la méthode utilisée pour prouver la validité d'une équation dans l'algèbre initiale d'une variété équationnelle.

Nous allons présenter la spécification des différentes procédures utilisées pour ce but.

Soient R un système de réécriture de termes convergent et C un ensemble de constructeurs. Pour tester si une équation E est un théorème inductif, nous commençons par tester si c'est un théorème équationnel. Si la forme normale des deux termes de l'équation sont égaux alors E est un théorème équationnel et par suite un théorème inductif. Sinon et si R est relativement complet par rapport à C alors nous ajoutons E à R puis nous déroulons la procédure de complétion de Knuth et Bendix. Si cette procédure diverge alors nous ne pouvons rien conclure sinon et si elle n'engendre pas de nouvelles relations entre les constructeurs alors E est un théorème inductif. Sinon nous avons deux manières au choix pour conclure. Soit nous procédons par la méthode de preuve par réductibilité inductive à condition que le système de réécriture R_C est linéaire à gauche. Soit nous procédons par la méthode de preuve par consistance. Dans la première manière nous testons si les nouvelles relations entre les constructeurs sont des théorèmes inductifs de R_C, c'est à dire des lemmes de E. Dans ce cas, E est un théorème inductif si et seulement si elles sont toutes des théorèmes inductifs. Dans la deuxième manière nous supposons que R est ω-complet et nous concluons que E n'est pas un théorème inductif.

Procédure 6.1 *[preuve_consistance] Cette procédure teste, par consistance, la validité d'une équation E dans l'algèbre initiale d'une variété équationnelle engendrée par un système de réécriture de termes R convergent. C est l'ensemble des constructeurs.*

Procédure preuve_consistance(R: système_de_réécriture; C: ensemble_de_opérateurs; E: équation)

début
$e_1 = e_2$ *est l'équation E*
si $e_1 \downarrow_R = e_2 \downarrow_R$
alors E est un théorème équationnel donc un théorème inductif
sinon R_C*: système_de_réécriture:= règle_racine(R. C)*
R_C*:= complétion(*R_C*)*
si R est relativement complet par rapport à C
alors R'*: système_de_réécriture:= complétion(ajouter(R. E))*
R'_C*: système_de_réécriture:= règle_racine(*R'*, C)*
supprimer(R'_C*.* R_C*)*
si $R'_C = \emptyset$
alors E est théorème inductif
sinon cas
% Preuve par réductibilité inductive %
si linéaire_gauche(R_C*)*
alors pour-chaque E'*: équation de* R'_C *faire*
si non preuve_réd_ind(R_C*.* E'*, C)*
alors E non théorème inductif
fsi
fpour
E est théorème inductif
fsi
% Preuve par consistance %
supposer R_C *ω-complet*
E non théorème inductif
fcas
fsi
fsi
fsi
fin.

Soit R un système de réécriture de termes linéaire à gauche convergent. Pour tester si une équation E est un théorème inductif, par réductibilité inductive, nous l'ajoutons à R puis nous déroulons la procédure de complétion de Knuth et Bendix. Si cette procédure diverge alors nous ne pouvons rien conclure sinon E est un théorème inductif si et seulement si le membre gauche de toutes les nouvelles règles, n'appartenant pas à R, est inductivement réductible par rapport à R.

Procédure 6.2 *[preuve_réd_ind] Cette procédure teste, par la réductibilité inductive, la validité d'une équation E dans l'algèbre initiale d'une variété équationnelle engendrée par un système de réécriture de termes R linéaire à gauche et convergent. Soit C l'ensemble des constructeurs.*

Procédure preuve_réd_ind (R: système_de_réécriture; E: équation; C: ensemble_de_opérateurs)

début
  R': système_de_réécriture:= complétion(ajouter(R, E))
  pour-chaque r: règle de supprimer(R', R) faire
    si non ind_réductible(membre_gauche(r), R, C)
      alors E non théorème inductif
    fsi
  fpour
  E est théorème inductif
fin.

6.2.2 Complétude relative

Nous allons décrire dans ce paragraphe la réalisation de la méthode utilisée pour tester la complétude relative d'une spécification donnée par rapport à la spécification de base.

Nous allons présenter la spécification des différentes procédures utilisées pour ce but.

Soient R un système de réécriture de termes et C un ensemble de constructeurs. Pour tester la complétude relative de R par rapport C nous testons la convertibilité de tous les opérateurs définit, n'appartenant pas à C, par rapport à C.

Procédure 6.3 *[rel_complet] Cette procédure teste, par convertibilité relative des opérateurs définis, la complétude relative d'un système de réécriture R par rapport à un ensemble de constructeurs C. Elle retourne l'ensemble des opérateurs non complètement définis, l'ensemble des termes non définis et l'ensemble des termes définis plus d'une fois.*

Procédure rel_complet(R: système_de_réécriture; C: ensemble_de_opérateurs)
Retourne(ensemble_de_opérateurs; ensemble_de_termes: ensemble_de_termes)

```
début
    opér_nonsufdéf: ensemble_de_opérateurs:= ∅
    terme_nondéf: ensemble_de_termes:= ∅
    terme_ambigu: ensemble_de_termes:= ∅
    pour-chaque f: opérateur de supprimer(opérateurs(R). C) faire
                f_nondéf, f_ambigu: ensemble_de_termes:= convertible(f. R. C)
                si f_nondéf ≠ ∅
                   alors ajouter(opér_nonsufdéf, f)
                         ajouter(terme_nondéf. f_nondéf)
                fsi
                si f_ambigu ≠ ∅
                   alors ajouter(terme_ambigu, f_ambigu)
                fsi
    fpour
    retourner(opér_nonsufdéf, terme_nondéf, terme_ambigu)
fin.
```

Procédure 6.4 *[convertible] Cette procédure teste la convertibilité d'un opérateur défini f, par recouvrement, dans un système de réécriture R par rapport à un ensemble de constructeurs C. Elle retourne l'ensemble des termes non définis et l'ensemble des termes définis plus d'une fois.*

Procédure convertible(f: opérateur; R: système_de_réécriture; C: ensemble_de_opérateurs)
Retourne(ensemble_de_termes: ensemble_de_termes)

début
R_C: système_de_réécriture:= règle_racine(R, C)
T: ensemble_de_termes:= membre_gauche(R, f)
$x_1, \ldots, x_{arité(f)}$: variable:= nouvelle_variable
S: ensemble_de_termes:= $\{f(x_1, \ldots, x_{arité(f)})\}$
f_nondéf, f_ambigu: ensemble_de_termes:= recouvrement(S, T, C)
pour-chaque s: terme de f_nondéf faire
si réductible(s, R_C)
alors supprimer(f_nondéf, s)
sinon si linéaire_gauche(R_C)
alors si ind_réductible(s, R_C, C)
alors supprimer(f_nondéf, s)
fsi
fsi
fsi
fpour
si f_nondéf $\neq \emptyset$ et non linéaire_gauche(R)
alors changer S, intéractivement, par un autre ensemble complet et non ambigu
recommencer le processus
fsi
retourner(f_nondéf, f_ambigu)
fin.

Procédure 6.5 *[recouvrement] Cette procédure teste le recouvrement d'un ensemble de termes S par un ensemble de termes T par rapport à un ensemble de constructeurs C. Elle retourne l'ensemble des termes non recouverts et l'ensemble des termes qui restent dans T.*

```
Procédure recouvrement(S, T: ensemble_de_termes; C: ensemble_de_opérateurs)
Retourne(ensemble_de_termes: ensemble_de_termes)

début
    nonrec: ensemble_de_termes:= ∅
    tant-que S ≠ ∅ faire
        s: terme:= un_élément(S)
        σ: substitution
        linéaire: booléen:= faux
        pour-chaque t: terme de T faire
            σ:= unification(s, t)
                excepté quand non-unifiable: continuer
            si linéaire(σ(s))
              alors linéaire:= vrai
                   "EXIT"
            fsi
        fpour
        si non linéaire
          alors ajouter(nonrec, s)
          sinon pour-chaque ρ: substitution de complément_sub(σ, C) faire
                        ajouter(S, ρ(s))
                        ajouter(T, ρ(t))
                fpour
                supprimer(T, t)
        fsi
        supprimer(S, s)
    ftq
    retourner(nonrec, T)
fin.
```

Procédure 6.6 *[complément_sub] Cette procédure calcule l'ensemble des substitutions complémentaires d'une substitution σ par rapport à un ensemble de constructeurs C.*

Procédure complément_sub(σ: substitution; C: ensemble_de_opérateurs)
Retourne(ensemble_de_substitutions)

```
début
    comp_σ: ensemble_de_substitutions:= {σ}
    pour-chaque [v: variable, t: terme] de σ faire
            si non variable(t)
              alors ens_σ: ensemble_de_substitutions:= ∅
                    pour-chaque s: terme de complément_terme(t. C) faire
                                pour-chaque ρ: substitution de comp_σ faire
                                            δ: substitution:= ρ
                                            changer(δ.[v.r].[v.s])
                                            ajouter(ens_σ. δ)
                                fpour
                                ajouter(comp_σ. ens_σ)
                    fpour
            fsi
    fpour
    supprimer(comp_σ, σ)
    retourner(comp_σ)
fin.
```

Procédure 6.7 *[complément_terme] Cette procédure calcule l'ensemble des termes complémentaires d'un terme t par rapport à un ensemble de constructeurs C.*

Procédure complément_terme(t: terme; C: ensemble_de_opérateurs)
Retourne(ensemble_de_termes)

début
  $f(t_1, \ldots, t_n)$ *est le terme t*
  comp_t: ensemble_de_termes:= $\emptyset$
  pour-chaque c: opérateur de C faire
    si $c \neq f$
      alors $x_1, \ldots, x_{arité(c)}$*: variable:= nouvelle_variable*
        ajouter(comp_t, $c(x_1, \ldots, x_{arité(c)})$*)*
    fsi
  fpour
  pour-chaque t_k*: terme de arguments(t) faire*
    si non variable(t_k*)*
      alors args: tableau_de_termes:= arguments(t)
        pour i: entier de $k+1$ *à n faire*
          args[i]:= nouvelle_variable
        fpour
        pour-chaque s: terme de complément_terme(t_k*.C) faire*
          args[k]:= s
          ajouter(comp_t, construire_terme(
            racine: f, arguments: args))
        fpour
    fsi
  fpour
  retourner(comp_t)
fin.

6.2.3 Réductibilité inductive

Nous allons décrire dans ce paragraphe la réalisation de la méthode utilisée pour tester la réductibilité inductive d'un terme par rapport à un système de réécriture de termes linéaire à gauche. La méthode présentée est celle proposée par E. Kounalis [41].

Nous allons présenter la spécification des différentes procédures utilisées pour ce but.

Soient R un système de réécriture de termes linéaire à gauche et C un ensemble de constructeurs. Un terme t est inductivement réductible par rapport à R si et seulement si toute instance de t par l'arbre de motifs sur C de R est réductible par rapport à R. L'ensemble des constructeurs C étant connu, nous prenons l'arbre de motifs uniquement sur C et non sur tout l'ensemble des opérateurs.

Procédure 6.8 *[ind_réductible] Cette procédure teste la réductibilité inductive d'un terme t par rapport à un système de réécriture de termes R linéaire à gauche. Soit C l'ensemble des constructeurs.*

Procédure ind_réductible(t: terme; R: système_de_réécriture; C: ensemble_de_opérateurs)
Retourne(booléen)

début
  n: entier:= cardinal(variables(t))
  pour-chaque $(s_1, \ldots, s_n)$ *de uplet(test_set(R, C), n) faire*
    σ*: substitution:=* $\{x_i \longleftarrow s_i \mid x_i \in variables(t) \wedge i \in [1..n]\}$
    si non réductible($\sigma(t)$, *R)*
      alors retourner(faux)
    fsi
  fpour
  retourner(vrai)
fin.

Procédure 6.9 *[test_set] Cette procédure calcule l'arbre de motifs d'un système de réécriture de termes R sur un ensemble de constructeurs C.*

Procédure test_set(R: système_de_réécriture; C: ensemble_de_opérateurs)
Retourne(ensemble_de_termes)

début
  TS_1, TS_2: *ensembe_de_termes*:= $\emptyset$
  d: *entier*:= *prof_sys(R)*
  i: *entier*:= 1
  fin: booléen:= *faux*
  tant-que non fin faire
    $TS_2 := TS_1$
    ajouter(TS_1, $\{tête(t,d) \mid t \in T(C)\downarrow_R \wedge prof(t) = i\}$)
    $i := i + 1$
    fin:= $(i = d + 2)$ *ou* $(TS_1 = TS_2)$
  ftq
  retourner(TS_1)
fin.

Soient t un terme de $T(F,X)$, R un système de réécriture de termes et j un entier naturel. Nous posons:

$$prof(t) = \begin{cases} 1 & t \in X \vee t \in F_0. \\ max(\{prof(t_1), \ldots, prof(t_n)\}) + 1 & t = f(t_1 \ldots . t_n) \wedge f \in F_n \wedge n > 0. \end{cases}$$

$$prof\_sys(R) = max(\{prof(t) \mid t \in MG(R)\}).$$

$$tête(t,j) = \begin{cases} t & j \geq prof(t). \\ f(x_1, \ldots, x_n) & j = 0 \wedge t = f(t_1, \ldots, t_n) \wedge f \in F_n \wedge n > 0, \\ f(tête(t_1, j-1), \ldots, tête(t_n, j-1)) & 0 < j < prof(t) \wedge t = f(t_1 \ldots . t_n) \wedge \\ & f \in F_n \wedge n > 0. \end{cases}$$

6.2.4 Automate et grammaire d'arbres

Nous allons décrire dans ce paragraphe la réalisation de la méthode utilisée pour la reconnaissance et la génération des formes normales closes par rapport à un système de réécriture de termes linéaire à gauche.

Nous allons présenter la spécification des différentes procédures utilisées pour ce but.

Soit R un système de réécriture de termes linéaire à gauche. Pour construire la grammaire d'arbres engendrant le langage d'arbres des formes normales closes par rapport à R, nous construisons l'automate d'arbres déterministe reconnaissant ce même langage. Pour cela, nous construisons d'abord un automate d'arbres indéterministe reconnaissant le langage d'arbres complémentaire, c'est à dire le langage des termes réductibles par rapport à R et nous réduisons ensuite l'indéterminisme puis nous calculons son complémentaire.

Procédure 6.10 *[const_auto] Cette procédure construit un automate indéterministe avec Λ reconnaissant le langage des termes clos réductibles par rapport à un système de réécriture de termes R linéaire à gauche.*

Procédure const_auto(R: système_de_réécriture) Retourne(automate)

début
  F: ensemble_de_opérateurs:= opérateurs(R)
  $e_?, ok$: état
  E: ensemble_de_états:= $\{e_?, ok\}$
  S: ensemble_de_états:= $\{ok\}$
  P: ensemble_de_règles:= $\emptyset$
  pour-chaque t: terme *de* membre_gauche(R) *faire*
    e_t: état:= état_règle($t, < E, S, P >$)
    ajouter($P, e_t \rightarrow ok$)
  fpour
  pour-chaque e: état *de* $E \ominus (\{e_?, ok\})$ *faire*
    ajouter($P, e \rightarrow e_?$)
  fpour
  pour-chaque f: opérateur *de* F *faire*
    ajouter($P, f(\underbrace{e_?, \ldots, e_?}_{arité(f)}) \rightarrow e_?$)
  fpour
  pour-chaque f: opérateur *de* F *faire*
    pour i: entier *de* 0 *à* arité(f)-1 *faire*
      ajouter($P, f(\underbrace{e_?, \ldots, e_?}_{i}, ok, e_?, \ldots, e_?) \rightarrow ok$)
    fpour
  fpour
  retourner($< E, S, P >$)
fin.

Procédure 6.11 *[état_règle] Cette procédure engendre la règle et les états des arguments d'un terme t par rapport à un automate $< E, S, P >$. Elle retourne l'état créé pour t, avec un effet de bord sur $< E, S, P >$.*

Procédure état_règle(t: terme; $< E, S, P >$: automate) Retourne(état)

début
  si variable(t)
    alors retourner($e_?$)
    sinon $f(t_1, \ldots, t_n)$ *est le terme t*
      e_t*: état:= nouvel_état(t)*
      ajouter(E, e_t*)*
      pour-chaque t_i*: terme de arguments(t) faire*
        e_{t_i}*: état:= état_règle(*t_i*,* $< E, S, P >$*)*
      fpour
      ajouter(P, $f(e_{t_1}, \ldots, e_{t_n}) \rightarrow e_t$*)*
      retourner(e_t*)*
  fsi
fin.

Procédure 6.12 *[réd_indéter] Cette procédure réduit l'indéterminisme d'un automate indéterministe avec $\Lambda < E, S, P >$ sur un ensemble d'opérateur F. S est égal à $\{ok\}$.*

Procédure réd-indéter($< E, S, P >$: automate: F: ensemble_de_opérateurs)
Retourne(automate)

début
E': ensemble_de_états:= $\emptyset$
S': ensemble_dc_états:= $\emptyset$
P': ensemble_de_règles:= $\emptyset$
E'': ensemble_de_états:= $\emptyset$
tant-que $E' \neq E''$ *faire*
$E'' := E'$
pour-chaque f: opérateur *de* F *faire*
n: entier:= arité(f)
pour-chaque $(q_1, \ldots, q_n)$ *de* uplet(E', n) *faire*
q: ensemble_de_états:=
$\{ \; e_0 \mid \exists \;\; e_1 \in q_1, \ldots, e_n \in q_n . e'_0 \in E.$
$e'_1 \in E, \ldots, e'_n \in E$
$e_1 \xrightarrow{*}_P e'_1 \wedge \cdots \wedge e_n \xrightarrow{*}_P e'_n \wedge$
$f(e'_1, \ldots, e'_n) \longrightarrow_P e'_0 \wedge e'_0 \xrightarrow{*}_P e_0 \; \}$
ajouter(E', q)
ajouter($P', f(q_1, \ldots, q_n) \rightarrow q$)
fpour
fpour
ftq
pour-chaque q: état *de* E' *faire*
si $ok \in q$
alors ajouter(S', q)
fsi
fpour
retourner($< E', S', P' >$)
fin.

Procédure 6.13 *[const_gram] Cette procédure construit la grammaire associée à un automate* $< E, S, P >$.

Procédure const_gram$(< E, S, P >$*: automate) Retourne(grammaire)*

début

ξ*: état*

Γ *: ensemble_de_états:=* $\{\xi\} \cup E$

Q*: ensemble_de_règles:=* $\emptyset$

pour-chaque e*: état de* S *faire*

ajouter$(Q, \xi \rightarrow e)$

fpour

pour-chaque $f(e_1, \ldots, e_n) \rightarrow e$*: règle de* P *faire*

ajouter$(Q, e \rightarrow f(e_1, \ldots, e_n))$

fpour

retourner$(< \Gamma, \xi, Q >)$

fin.

6.3 Session de REVE

Les exemples suivants forment un jeux d'essai d'exécution du système REVE. Les textes en caractères majuscules ou soulignés sont entrés à la console par l'utilisateur. les autres sont imprimés par le système.

```
-> READ ~/exp/retenu

User equations:

1.   f(x, x, y) == x
2.   f(x, y, x) == x
3.   f(x, y, y) == y

No critical pair equations.

No rewrite rules.

-> KB

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1.   f(x, x, y) -> x
2.   f(x, y, x) -> x
3.   f(x, y, y) -> y

Your system is complete!

-> REL-COMPLETENESS

The operators
     f
may not be completely defined;
it's hard to tell for sure because the system is not linear.
Try adding rules with the following left-hand sides:
     f(x1, x2, x3)

All following terms are defined more than once:
```

```
        f(v1, v1, v2)
        f(v1, v2, v1)
        f(v1, v2, v2)

    Whould you like to give a complete and not ambiguous reference
                    for the operator 'f'? YES
    Please type the sequence terms of the reference
                    separated by ';', terminated with <ESC>:
    f(0,x,y);f(1,x,y)

    Whould you like to give some constructors? YES
    Constructors: 0 1

    Precedence extended.

    The operator 'f' is completely defined.

    The following terms are defined more than once:
    f(1, 1, 1)

    f(0, 0, 0)
```

```
-> READ ~/exp/flatten

User equations:

 1.   nill @ x == x
 2.   x @ nill == x
 3.   (x @ y) @ z == x @ (y @ z)
 4.   flatten(nill) == nill
 5.   flatten(list(x)) == flatten(x)
 6.   flatten(a) == a
 7.   flatten(b) == b
 8.   flatten(a @ x) == a @ flatten(x)
 9.   flatten(b @ x) == b @ flatten(x)

No critical pair equations.

No rewrite rules.

Note that the following identifiers were parsed as nullary operators:
     nill  a     b

-> CONSTRUCTORS a b @ nill list

Precedence extended.

-> KB

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1.   nill @ x -> x
2.   x @ nill -> x
3.   flatten(nill) -> nill
```

```
4.   flatten(list(x)) -> flatten(x)
5.   flatten(a) -> a
6.   flatten(b) -> b
7.   flatten(a @ x) -> a @ flatten(x)
8.   flatten(b @ x) -> b @ flatten(x)
9.   (x @ y) @ z -> x @ (y @ z)

Your system is complete!

-> REL-COMPLETENESS

There are relations among the constructors.

The operators
     flatten
are not completely defined.

Try adding rules with the following left-hand sides:
     flatten(list(x1) @ x2)

-> ADDITIONAL flatten(list(x)@y)==flatten(x)@flatten(y)

-> KB

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

 1.   nill @ x -> x
 2.   x @ nill -> x
 3.   flatten(nill) -> nill
 4.   flatten(list(x)) -> flatten(x)
 5.   flatten(a) -> a
 6.   flatten(b) -> b
```

```
 7.   flatten(a @ x) -> a @ flatten(x)
 8.   flatten(b @ x) -> b @ flatten(x)
 9.   (x @ y) @ z -> x @ (y @ z)
10.   flatten(list(x) @ y) -> flatten(x) @ flatten(y)

Your system is complete!

-> REL-COMPLETENESS

There are relations among the constructors.

All operators are completely defined.

-> PROVE flatten(x@y)==flatten(x)@flatten(y)

The equation

     flatten(x @ y) == flatten(x) @ flatten(y)

IS NOT an equational theorem of your system.

How do you wish to proceed with the proof:
Reducible-proof, Consistency-proof, Cancel, Interrupt? CONSISTENCY-PROOF

A proof by consistency of the equation

     flatten(x @ y) == flatten(x) @ flatten(y)

involves completing your system augmented by that equation.

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:
```

```
1.   nill @ x -> x
2.   x @ nill -> x
3.   flatten(nill) -> nill
4.   flatten(list(x)) -> flatten(x)
5.   flatten(a) -> a
6.   flatten(b) -> b
7.   (x @ y) @ z -> x @ (y @ z)
8.   flatten(x @ y) -> flatten(x) @ flatten(y)

Your system is complete!

The equation

     flatten(x @ y) == flatten(x) @ flatten(y)

IS an inductive theorem of your system.

Do you want to restore the initial system? NO

-> PROVE flatten(flatten(x))==flatten(x)

The equation

     flatten(flatten(x)) == flatten(x)

IS NOT an equational theorem of your system.

How do you wish to proceed with the proof:
Reducible-proof, Consistency-proof, Cancel, Interrupt? CONSISTENCY-PROOF

A proof by consistency of the equation

     flatten(flatten(x)) == flatten(x)

involves completing your system augmented by that equation.

Starting Knuth-Bendix...
```

```
No user equations.

No critical pair equations.

Rewrite rules:

1.   nill @ x -> x
2.   x @ nill -> x
3.   flatten(nill) -> nill
4.   flatten(list(x)) -> flatten(x)
5.   flatten(a) -> a
6.   flatten(b) -> b
7.   (x @ y) @ z -> x @ (y @ z)
8.   flatten(x @ y) -> flatten(x) @ flatten(y)
9.   flatten(flatten(x)) -> flatten(x)

Your system is complete!

The equation

     flatten(flatten(x)) == flatten(x)

IS an inductive theorem of your system.

Do you want to restore the initial system? YES

-> PROVE flatten(flatten(x))==x

The equation

     flatten(flatten(x)) == x

IS NOT an equational theorem of your system.

How do you wish to proceed with the proof:
Reducible-proof, Consistency-proof, Cancel, Interrupt? CONSISTENCY-PROOF

A proof by consistency of the equation
```

```
    flatten(flatten(x)) == x

involves completing your system augmented by that equation.

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1.  nill @ x -> x
2.  x @ nill -> x
3.  flatten(nill) -> nill
4.  flatten(a) -> a
5.  flatten(b) -> b
6.  (x @ y) @ z -> x @ (y @ z)
7.  flatten(x @ y) -> flatten(x) @ flatten(y)
8.  flatten(flatten(x)) -> x
9.  list(x) -> x

Your system is complete!

Do you want suppose that the based system IS w-complete? YES

The theory IS NOT consistent because the folowing rules modified the initial
algebra:
list(x) -> x

The equation

    flatten(flatten(x)) == x

IS NOT an inductive theorem of your system.

Do you want to restore the initial system? YES
```

```
-> READ ~/exp/modulo2

User equations:

1.   s(s(0)) == 0
2.   0 + x == x
3.   s(0) + x == s(x)

No critical pair equations.

No rewrite rules.

Note that the following identifiers were parsed as nullary operators:
     0

-> CONSTRUCTORS 0 s

Precedence extended.

-> KB

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1.   s(s(0)) -> 0
2.   0 + x -> x
3.   s(0) + x -> s(x)

Your system is complete!

-> REL-COMPLETENESS

There are relations among the constructors.
```

```
All operators are completely defined.

-> PROVE s(s(x))==x

The equation

     s(s(x)) == x

IS NOT an equational theorem of your system.

How do you wish to proceed with the proof:
Reducible-proof, Consistency-proof, Cancel, Interrupt? REDUCIBLE-PROOF

A proof by reducible-inductive of the equation

     s(s(x)) == x

involves completing your system augmented by that equation.

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1.   0 + x -> x
2.   s(0) + x -> s(x)
3.   s(s(x)) -> x

Your system is complete!

The equation

     s(s(x)) == x
```

```
IS an inductive theorem of your system.

Do you want to restore the initial system? NO

-> PROVE s(x)+y==s(x+y)

The equation

     s(x) + y == s(x + y)

IS NOT an equational theorem of your system.

How do you wish to proceed with the proof:
Reducible-proof, Consistency-proof, Cancel, Interrupt? REDUCIBLE-PROOF

A proof by reducible-inductive of the equation

     s(x) + y == s(x + y)

involves completing your system augmented by that equation.

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1.   0 + x -> x
2.   s(s(x)) -> x
3.   s(x) + y -> s(x + y)

Your system is complete!

The equation

     s(x) + y == s(x + y)
```

```
IS an inductive theorem of your system.

Do you want to restore the initial system? YES

-> PROVE s(x)+y==s(x+y)

The equation

     s(x) + y == s(x + y)

IS NOT an equational theorem of your system.

How do you wish to proceed with the proof:
Reducible-proof, Consistency-proof, Cancel, Interrupt? CONSISTENCY-PROOF

A proof by consistency of the equation

     s(x) + y == s(x + y)

involves completing your system augmented by that equation.

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1.   0 + x -> x
2.   s(s(x)) -> x
3.   s(x) + y -> s(x + y)

Your system is complete!

The equation
```

```
        s(x) + y == s(x + y)

    IS an inductive theorem of your system.

    Do you want to restore the initial system? NO

    -> GRAMMAR

    The set of derivation rules is:
    G -> E6
    E6 -> 0
    G -> E7
    E7 -> s(E6)
```

```
-> READ ~/exp/egalite

User equations:

1.   0 + x == x
2.   s(x) + y == s(x + y)
3.   0 = 0 == 0
4.   0 = s(x) == s(0)
5.   s(x) = 0 == s(0)
6.   s(x) = s(y) == x = y

No critical pair equations.

No rewrite rules.

Note that the following identifiers were parsed as nullary operators:
     0

-> CONSTRUCTORS 0 s

Precedence extended.

-> KB

Starting Knuth-Bendix...

No user equations.

No critical pair equations.

Rewrite rules:

1.   0 + x -> x
2.   s(x) + y -> s(x + y)
3.   0 = 0 -> 0
4.   0 = s(x) -> s(0)
5.   s(x) = 0 -> s(0)
6.   s(x) = s(y) -> x = y
```

```
Your system is complete!

-> GRAMMAR

The set of derivation rules is:
G -> E16
E16 -> 0
G -> E17
E17 -> s(E16)
E17 -> s(E17)

-> DELETE 5

Because you deleted a rewrite rule, your rewriting system is no longer
guaranteed to be complete with respect to your original equational system.

-> GRAMMAR

The set of derivation rules is:
G -> E25
E25 -> 0
G -> E26
E26 -> s(E25)
E26 -> s(E26)
E26 -> s(E)
G -> E
E -> E26 = E25
E -> E + E25
E -> E + E26
E -> E + E
E -> E25 = E
E -> E = E25
E -> E26 = E
E -> E = E26
E -> E = E

-> DELETE 1
```

```
Because you deleted a rewrite rule, your rewriting system is no longer
guaranteed to be complete with respect to your original equational system.

-> GRAMMAR

The set of derivation rules is:
G -> E33
E33 -> 0
G -> E
E -> E33 + E33
E -> E33 = E
E -> E = E33
E -> E34 = E33
E -> E = E
E -> E = E34
E -> E34 = E
E -> E33 + E
E -> E + E33
E -> E33 + E34
E -> E + E
E -> E + E34
G -> E34
E34 -> s(E33)
E34 -> s(E)
E34 -> s(E34)
```

CONCLUSION

Au cours de ce travail, nous nous sommes attaché à la preuve de théorèmes, d'un point de vue pratique, dans le cadre de la logique équationnelle.

Les deux méthodes qui existent pour faire des preuves de validités de théorèmes inductifs dans l'algèbre initiale d'une variété équationnelle et fondées sur la réécriture sont: la méthode de preuve par consistance et celle basée sur la réductibilité inductive.

Chacune de ces méthodes pose des inconvénients pratiques dans la mise en œuvre. La première suppose que la spécification de base est ω-complète. Propriété que nous ne pouvons vérifier que dans des cas particuliers. La deuxième méthode utilise la notion de réductibilité inductive. Concept que nous ne pouvons encore décider efficacement que dans le cas des systèmes de réécriture linéaires à gauche. La résolution de ces deux problèmes ne peut pas être traitée en pratique à l'heure actuelle. ainsi nous avons proposé une méthode qui est la synthèse de ces deux méthodes.

La méthode de preuve par consistance est très puissante et il s'est avéré que souvent le système de réécriture associé à la spécification de base est linéaire à gauche. D'autre part, il se trouve que la réductibilité inductive est décidable dans ce cas. Ainsi notre méthode consiste à utiliser la méthode de preuve par consistance sans supposer que la spécification de base soit ω-complète, mais si une nouvelle équation entre les constructeurs est engendrée par complétion, nous essayons de prouver sa validité par la deuxième méthode.

La propriété de la complétude relative d'une spécification par rapport à la spécification de base est une propriété fondamentale dans les spécifications structurées. Elle est aussi requise dans la méthode de preuve par consistance. Pour tester cette propriété nous avons proposé une extension de la méthode de recouvrement. Nous avons étendu cette méthode dans le cas où il y a des relations entre les constructeurs et même lorsque la spécification n'est pas linéaire à gauche. La correction totale de cette méthode est démontrée.

La propriété de la ω-complétude d'une spécification est très intéressante dans l'étude des théorèmes d'une spécification et son axiomatisation. Elle facilite aussi la métho-

de de preuve par consistance. Nous avons essayé de mener une étude systématique de cette propriété qui est très difficile. Nous avons exposé les deux méthodes, dégagées d'un certain nombre d'exemples de spécifications ω-complètes, pour la tester. Puis nous avons énuméré les problèmes qui se posent lors de la construction d'un enrichissement ω-complet d'une spécification donnée. Ainsi nous nous sommes contenté de décider la ω-complétude relative d'une spécification par rapport à un ordre nœthérien. Nous avons donné un théorème sur sa décidabilité dans ce cas particulier mais il ne peut pas être implantable dès maintenant. Nous avons aussi proposé un algorithme de décision de la ω-complétude d'une spécification dans certains cas où le domaine de l'algèbre initiale de la variété équationnelle est fini. Nous l'avons décidée dans le cas où le domaine de l'algèbre générique de la variété équationnelle sur un ensemble de variables est fini et dans le cas où la profondeur des termes en formes normales, par rapport au système de réécriture associé à la spécification, est fini.

Par ailleurs nous avons donné une méthode, basée sur les automates et les grammaires d'arbres et sur la réécriture, de reconnaissance et de génération du langage des formes normales closes d'un système de réécriture de termes linéaire à gauche. La caractérisation de tel langage peut être utilisée pour le test de la réductibilité inductive, l'ω-complétude et dans la preuve de théorème inductif. Pour cette dérnière, il suffit de démontrer l'équivalence entre le domaine de l'algèbre initiale et son enrichissement par l'équation à prouver. Nous avons utilisé cette méthode de construction pour tester la finitude et le vacuité d'un langage d'arbres. Ainsi nous pouvons décider la finitude et le vacuité du domaine de l'algèbre générique et de l'algèbre initiale d'une variété équationnelle.

Tous les résultats ci-dessus sont implantés dans le logiciel de réécriture REVE. Nous avons eu l'occasion de manipuler un logiciel de grande taille. Nous nous sommes attaché à maintenir les qualités de modularité, de clarté et d'adaptabilité que présente REVE. Par ailleurs nous avons testé ce logiciel sur de nombreuses spécifications. Ce travail expérimental nous a permis de nous assurer des qualités d'efficacité et de puissance de notre logiciel.

Nous nous sommes limité à l'étude de la complétude relative aux cas de systèmes de réécriture de termes et aux cas de systèmes de réécriture équationnelle. Des prolongements sont à envisager dans le sens d'une généralisation de la méthode de test de la complétude relative au cas des systèmes de réécriture conditionnelle. L'extension de cette méthode dans le cas des spécifications multi-sortes est une autre perspective envisageable, vu sa grande efficacité.

La richesse de la propriété d'ω-complétude ouvre des perspectives dans un avenir plus long.

Une extension de la méthode de reconnaissant et de génération du langage des formes normales closes dans le cas des systèmes de réécriture non linéaires à gauche est envisageable en utilisant toujours la réécriture mais des automates asynchrones. introduits pour les monoïdes partiellements commutatives.

LISTE DES SYMBOLES

N :	ensemble des entiers naturels,
F :	ensemble d'opérateurs,
C :	ensemble de constructeurs,
D :	ensemble d'opérateurs définis.
$arité(f)$:	arité d'un opérateur f,
F_i :	ensemble des opérateurs d'arité i avec $i \geq 0$,
$(F, arité)$:	signature,
(S, F) :	algèbre de domaine S et d'opérateurs F,
X :	ensemble de variables,
$G(F, X)$:	algèbre générique sur un ensemble de variables X,
$I(F)$:	algèbre initiale,
$T(F, X)$:	ensemble des termes d'opérateurs F et de variables X.
$T(F)$:	ensemble des termes clos d'opérateurs F,
$O(t)$:	ensemble des occurrences d'un terme t,
ε :	occurrence vide,
$V(t)$:	ensemble des variables d'un terme t,
$G(t)$:	ensemble des occurrences closes d'un terme t.
t/u :	sous-terme à l'occurrence u d'un terme t,
$t[u \longleftarrow s]$:	remplacement du sous-terme à l'occurrence u par le terme s dans un terme t,
$\Sigma(F, X)$:	ensemble des substitutions sur $T(F, X)$,
$\Sigma(F)$:	ensemble des substitutions closes sur $T(F)$.
$D(\sigma)$:	domaine d'une substitution σ,
$s = t$:	équation,
E :	équation,
A :	ensemble d'axiomes,
$M(A)$:	variété équationnelle engendrée par A,
$=_A$:	égalité équationnelle engendrée par A,
$=_{ind(A)}$:	égalité inductive engendrée par A,
$G(F, X, A)$:	algèbre générique sur X de $M(A)$.

$I(F, A)$:	algèbre initiale de $M(A)$.
(S, F, A) :	spécification de sorte S, d'opérateurs F et d'axiomes A.
$s \to t$:	règle de réécriture.
R :	système de réécriture de termes.
$s \longrightarrow_R t$:	réécriture du terme s vers le terme t par rapport à R.
$\xrightarrow{+}_R$:	fermeture transitive de la relation $\longrightarrow_R$ définie dans $T(F, X)$.
$\xrightarrow{*}_R$:	fermeture réflexive-transitive de la relation $\longrightarrow_R$ définie dans $T(F, X)$,
$\xleftrightarrow{*}_R$:	fermeture réflexive-symétrique-transitive de la relation $\longrightarrow_R$ définie dans $T(F, X)$,
$t \downarrow_R$:	forme normale d'un terme t par rapport à R.
$MG(R)$:	ensemble des membres gauches de R,
$MG(f, R)$:	ensemble des membres gauches de R de racine un opérateur f.
$T(F, X) \downarrow_R$:	ensemble des termes de $T(F, X)$ en formes normales par rapport à R.
$T(F) \downarrow_R$:	ensemble des termes clos de $T(F)$ en formes normales par rapport à R.
$T(F, X) \uparrow^R$:	ensemble des termes de $T(F, X)$ réductibles par rapport à R.
$T(F) \uparrow^R$:	ensemble des termes clos de $T(F)$ réductibles par rapport à R.
$T_f(F, X)$:	ensemble des termes de $T(F, X)$ de racine un opérateur f.
$T_f(F)$:	ensemble des termes clos de $T(F)$ de racine un opérateur f.
$I(t)$:	ensemble des instances closes d'un terme t.
$I(T)$:	ensemble des instances closes de tous les termes de T.
$A(t, C)$:	complémentaire d'un terme t par rapport à un ensemble d'opérateurs C,
$B(\sigma, C)$:	complémentaire d'une substitution σ par rapport à un ensemble d'opérateurs C,
$prof(t)$:	profondeur d'un terme t,
(E, S, P) :	automate dont l'ensemble d'états E, d'états de satisfaction S et de règles de transition P,
(Γ, ξ, Q) :	grammaire dont l'ensemble de vocabulaires auxiliaires Γ, d'axiome ξ et de règles de dérivation Q,
Λ :	arbre vide,
ok :	état de satisfaction,
$e_?$:	état d'attente,
$\wedge$:	et booléen,
$\vee$:	ou booléen,
$\ominus$:	différence ensembliste,
$\oplus$:	somme ensembliste,
$\otimes$:	produit ensembliste.

Bibliographie

[1] R. Aubin. Mechanizing structural induction. In *Theoretical Computer Science 9*, pages 329–362, 1979.

[2] L. Bachmair, N. Dershowitz, and D. Plaisted. Completion without failure. In *Proceedings of the Colloquium on the Resolution of Equations in Algebraic Structures*, 1987.

[3] A. BenCherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–160, October 1987.

[4] M. Bidoit. Une méthode de présentation des types abstraits, applications. Thèse de 3ième cycle, Université Paris-Sud, 1981.

[5] G. Birkhoff. On the structure of abstract algebras. In *Proceedings Cambridge Phil. Soc. 31*, pages 433–454, 1935.

[6] R.S. Boyer and J.S. Moore. *A Computational Logic.* Academic Press, New York, 1979.

[7] B. Buchberger. History and basic features of the critical-pair / completion approach. In *Proceedings of the 1st Conference on Rewriting techniques and applications*, Springer Verlag, Lecture Notes in Computer Science 202, 1985.

[8] R-M. Burstall. Proving properties of programs by structural induction. In *Computer Journal 12*, pages 41–48, 1969.

[9] H. Comon. Sufficient completeness, term rewriting system and anti-unification. In J. Siekmann, editor, *Proceedings 8th Conference on Automated Deduction*, pages 128–140, Springer Verlag, Lecture Notes in Computer Science 230, 1986.

[10] H. Comon. Unification et disunification, théories et applications. Thèse de l'Institut Polytechnique de Grenoble, 1988.

[11] H. Comon and J-L. Rémy. *How to Characterize the Language of Ground Normal Forms.* Technical Report 676, INRIA-Lorraine, 1987.

[12] M. Davis, Y. Matijasevic, and J. Robinson. Hilbert's tenth problem: positive aspects of a negative solution. In *F. E. Browder, Editor, Mathematical Developments Arising from Hilbert Problems, American Mathematical Society*, pages 323–378, 1976.

[13] N. Dershowitz. *Applications of the Knuth-Bendix Completion Procedure.* Technical Report ATR-83(8478)-2, The Aerospace Corporation, El Segundo, Calif. 90245, May 1983.

[14] N. Dershowitz. A note on simplification orderings. In *Information Processing Letters 9*, pages 212–215, 1979.

[15] N. Dershowitz. Termination. In *Proceedings 1st Conference on Rewriting Techniques and Applications*, pages 180–224, Springer Verlag, Dijon (France), May 1985.

[16] J-P. Finance. Etude de la construction des programmes: méthode et langage de spécification et de résolution de problèmes. Thèse de Doctorat d'Etat, Université de Nancy, 1979.

[17] R. Forgaard and J. Guttag. *REVE: A term rewriting system generator with failure-resistant Knuth-Bendix.* Technical Report, MIT-LCS, 1984.

[18] L. Fribourg. A strong restriction of the inductive completion procedure. In *Proceedings 13th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science 226*, pages 105–115, 1986.

[19] J. Goguen, C. Kirchner, H. Kirchner, A. Megrelis, J. Meseguer, and T. Winkler. An introduction to OBJ-3. In J-P. Jouannaud and S. Kaplan, editors, *Proceedings 1st International Workshop on Conditional Term Rewriting Systems*, Springer-Verlag, June 1988. Also as internal report CRIN: 88-R-001.

[20] J.A. Goguen. How to prove algebraic inductive hypotheses without induction, with applications to the correctness of data type implementation. In W. Bibel and Kowalski, editors, *Proceedings 5th Conference on Automated Deduction*, pages 356–373, Springer-Verlag, Les Arcs (France), 1980.

[21] G. Grătzer. *Universal Algebra.* Springer-Verlag, 1979. Second Edition.

[22] J. V. Guttag and J. J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, 10:27–52, 1978.

[23] J. Heering. Partial evaluation and ω-completeness of algebraic specifications. *Theoretical Computer Science*, 43:149–167, 1986.

[24] L. Henkin. The logic of equality. In *Mathematical Monthly*, pages 597–612, 1977.

[25] G. Huet. A complete proof of the Knuth-Bendix completion algorithm. In *JCSS 23*, pages 11–21, 1981.

[26] G. Huet and J-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25(2):239–266, October 1982. Preliminary version in Proceedings 21st Symposium on Foundations of Computer Science, IEEE, 1980.

[27] G. Huet and D-S. Lankford. *On the Uniform Halting Problem for Term Rewriting Systems.* Technical Report 283, Laboria, France, 1978.

[28] G. Huet and D. Oppen. Equations and rewrite rules: a survey. In R. Book, editor, *Formal Language Theory: Perspectives and Open Problems*, pages 349–405, Academic Press, New-York, 1980.

[29] J-M. Hullot. Compilation de formes canoniques dans les théories équationelles. Thèse de 3ième cycle, Université de Paris Sud, Orsay, France, 1980.

[30] J-P. Jouannaud. La terminaison finie des systèmes de réécriture. *Actes du seminaire du LITP*, 1981.

[31] J.P. Jouannaud and E. Kounalis. Proof by induction in equational theories without constructors. In *Proceedings 1st Symp. on Logic In Computer Science*, pages 358–366, Boston (USA), 1986.

[32] J-P. Jouannaud and P. Lescanne. On multiset orderings. In *Information Processing Letters 10*, pages 57–63, 1982.

[33] J.P. Jouannaud, P. Lescanne, and F. Reinig. Recursive decomposition ordering. In Bjørner D., editor, *Formal Description of Programming Concepts 2*, pages 331–348, North Holland, Garmisch-Partenkirchen, RFA, 1982.

[34] S. Kamin and J-J. Lévy. Attempts for generalizing the recursive path ordring. *Inria, Rocquencourt*, 1982.

[35] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[36] D. Kapur, P. Narendran, and H. Zhang. Proof by induction using test sets. In *8th International Conference on Automated Deduction. Lecture Notes in Computer Science 230*, pages 99–117, Springer Verlag, 1986.

[37] D. Kapur and G. Sivakumar. Experiments with an architecture of RRL, a rewrite rule laboratory. In *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory*, pages 33–56, 1983.

[38] H. Kirchner. A general inductive completion algorithm and application to abstract data types. In R. Shostak, editor, *Proceedings 7th international Conference on Automated Deduction*, pages 282–302, Springer-Verlag, Lecture Notes in Computer Science, 1984.

[39] D. Knuth and P. Bendix. *Simple Word Problems in Universal Algebra.* pages 263–297. Pergamon Press, 1970.

[40] E. Kounalis. Completeness in data type specifications. In B. Buchberger, editor. *Proceedings EUROCAL Conference*, Springer-Verlag, Linz (Austria), 1985.

[41] E. Kounalis. Validation de spécifications algébriques par complétion inductive. Thèse de Doctorat, Université de Nancy 1, CRIN, Nancy, 1985.

[42] W. Küchlin. Inductive completion by ground proof transformation. *University of Delaware USA*, Rept. 87-08, 1987.

[43] D-S. Lankford. A simple explanation of inductionless induction. *Louisiana Tech. University, Dep. of Math.*, Memo MTP-14, Ruston, 1981.

[44] J-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–318, 1986.

[45] A. Lazrek, P. Lescanne, and J-J. Thiel. *Proving inductive equalities. Algorithms and Implementation.* Technical Report, Internal Report CRIN 86-R-087, 1986.

[46] A. Lazrek, P. Lescanne, and J-J. Thiel. *Tools for Proving Inductive Equalities, Relative Completeness and ω-Completeness.* Technical Report 88-R-131, CRIN, 1988. to be published in Information and Computation.

[47] P. Lescanne. Computer experiments with the REVE term rewriting systems generator. In *Proceedings, 10th ACM Symposium on Principles of Programming Languages*, ACM, 1983.

[48] P. Lescanne. Uniform termination of term rewriting systems, recursive decomposition ordering with status. In B. Courcelle, editor, *Proceedings 9th Colloque les Arbres en Algebre et en Programmation*, pages 182–194, Cambridge University Press, Bordeaux (France), 1984.

[49] B. H. Liskov, E. Moss, C. Schaffert, B. Scheifler, and A. Snyder. *CLU Reference Manual.* Technical Report, MIT, Lab for Computer Science, 1979.

[50] R. C. Lyndon. Identities in finite algebras. In *Proceedings American Mathematical Society 5*, pages 8–9, 1954.

[51] Z. Manna and Ness. On the termination of markov algorithms. In *Third Hawai International Conference on System Sciences*, pages 789–792, 1970.

[52] P. Marchand. Langages d'arbres, langages dans les algèbres libres. Thèse d'état de l'Université de Nancy I, 1981.

[53] J. Meseguer and J-A. Goguen. Initiality, induction and computability. *CSL Technical Report 140, SRI International, Menlo Park California*, 1983.

[54] D.L. Musser. On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162. Association for Computing Machinery. 1980.

[55] M-H-A. Newman. On theories with a combinatorial definition of *Equivalence*. In *Annals of Math*, pages 223–243. 1942.

[56] T. Nipkow and G. Weikum. A decidability result about sufficient completeness of axiomatically specified abstract data types. In *6th GI Conference*, pages 257–268, Springer-Verlag, Lecture Notes in Computer Science, 1983.

[57] E. Paul. Proof by induction in equational theories with relations between constructors. In B. Courcelle, editor. *Proceedings 9th Colloquium on Trees in Algebra and Programming*, pages 210–225, Cambridge University Press, 1984.

[58] D. Plaisted. A recursively defined ordering for proving termination of term rewriting sytems. *Dept. of Computer Science Report 78-943*. 1978.

[59] D. Plaisted. Semantic confluence tests and completion methods. In *Journal Information and Control 65*, pages 182–215, 1985.

[60] G. D. Plotkin. The λ-calculus is ω-incomplete. *Journal of Symbolic Logic*, 39:313–317, 1974.

[61] L. Puel. Proof in the final algebra. In B. Courcelle, editor. *Proceedings 9th Colloquium on Trees in Algebra and Programming*, pages 227–242. Cambridge University Press, 1984.

[62] F. Reinig. L'ordre de décomposotion: un outil incrémental pour prouver la terminaison finie des systèmes de réécriture équationnels. Thèse de 3ième cycle. Université de Nancy, 1981.

[63] J-L. Rémy. Etude des systèmes de réécriture conditionnels et applications aux types abstraits algébriques. Thèse d'Etat de l'Institut National Polytechnique de Lorraine, 1982.

[64] M. Rusinowitch. Path of subterm ordring and recursive decomposition ordring revisited. In *Proceedings 1st Conference Rewriting Technique and Applications. Lecture Notes in Computer Science 202*, pages 225–240, Springer Verlag, Dijon France, 1985.

[65] A. Tarski. Equational logic and equational theories of algebras. In *Contributions to Mathematical Logic, K. Schütte, Ed.*, North-holland, Amsterdam, 1968.

[66] W. Taylor. Equational logic. *Houston Journal of Mathematics*, 1979. Appears also in [21], Appendix 4.

[67] J-J. Thiel. Stop loosing sleep over incomplete data type specifications. In *Proceeding 11th ACM Symp. on Principles of Programming Languages*. pages 76–82. Association for Computing Machinery. 1984.

[68] Y. Toyama. Counterexamples to terminating for the direct sum of term rewriting systems. *Information Processing Letters*. 25(3):141–143. May 1986.

[69] H. Trad. Contribution à l'étude des langages reconnaissables d'arbres. Thèse de 3ième cycle. Université de Paris 6. 1987.

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

VU LES RAPPORTS ETABLIS PAR :
Monsieur FINANCE, Professeur CRIN/INPL,
Monsieur DAUCHET, Professeur Université de Lille.

Le Président de l'Institut National Polytechnique de Lorraine,
autorise :

Monsieur LAZREK Azzeddine

à soutenir devant l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

"Etude et réalisation de méthodes de preuve par récurrence en logique équationnelle"

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : "Informatique"

Fait à Vandoeuvre le, **4 Novembre 1988**
Le Président de l'I.N.P.L.,

M.GANTOIS

2, avenue de la Forêt de Haye - B.P. 3 - 54501 VANDŒUVRE CEDEX
Téléphone : 83. 57. 48. 48 - Télex : 961 715 F - Télécopie : 83. 57. 49. 55

RESUME

L'objet de cette thèse est l'étude pratique de preuve par récurrence en logique équationnelle. Elle fournit ainsi un outil puissant pour la programmation logique et/ou fonctionnelle et pour la démonstration automatique.

Notre objectif principal est l'étude et la mise en œuvre d'une méthode de preuve de théorèmes inductifs dans l'algèbre initiale d'une variété équationnelle. La méthode étudiée est la synthèse de la méthode de preuve par consistance et de celle basée sur la réductibilité inductive.

Le deuxième objectif est l'étude de la complétude relative, souhaitable dans la conception et la correction des spécifications algébriques structurées et requise par la méthode de preuve par consistance.

Une partie de ce travail est consacrée à la présentation de la ω-complétude; elle constitue, à notre connaissance, la première tentative d'étude systématique de cette propriété.

Par ailleurs nous donnons une méthode de reconnaissance et de génération des formes normales closes d'un système de réécriture.

Enfin il est à noter que toutes ces méthodes de preuves de validité, de complétude relative ainsi que la reconnaissance et la génération des formes normales closes d'un système de réécriture ont été implantées dans le démonstrateur automatique REVE.

Mots-clés: algèbre initiale, spécification, consistance relative, complétude relative, ω-complétude, réécriture, procédure de complétion, récurrence, automate, grammaire.