Sur l'intégration des langages algébriques et logique

Rachid Echahed

THESE

présentée par

Rachid ECHAHED

pour obtenir le grade de Docteur
de L'institut National Polytechnique de Grenoble
(Arrêté ministériel du 23 novembre 1988)

Spécialité : Informatique

Sur l'intégration des langages algébriques et logiques

Date de soutenance : 26 novembre 1990

Composition du Jury :

J-P. Verjus (Président)
M. Dauchet (rapporteur)
C. Kirchner (rapporteur)
D. Bert (examinateur)
J-P. Jouannaud (examinateur)

Thèse préparée au sein du Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle (LIFIA)

Abstract :

This thesis presents an approach, based on Horn clause logic with equality, to the integration of logic and functional programming languages. We start our investigation by defining the programs we consider. They are syntactically defined as theory presentations in the framework of Horn clause logic with equality. The semantic of a program is its least Herbrand E-model.

We then focus on the computational aspects of this language, and propose a deductive system consisting of a unique rule we call SLDEI-resolution. We prove the soundness, the completeness and the strong completeness of this rule.

Implementations of SLDEI-resolution lean on algorithms of resolution of equations. We consider such algorithms based on narrowing relationship and improve them by using narrowing strategies. These strategies are not complete in general, so we propose sufficient conditions on these strategies that ensure the completeness of the algorithms we consider.

We characterize afterwards a class of programs, called uniform programs, so that any narrowing strategy leads to a complete algorithm of resolution of equations. We propose then a procedure for deciding whether or not a program is uniform. Furthermore, we give syntactical criteria that warrant the uniformity of programs.

Finally, we give an overview of main features of a programming language based on the approach presented in this thesis, and sketch our implementation.

Keywords : Logic Programming, Algebraic Specification, Horn Clause Logic with Equality, SLDEI-resolution, EI-unification, Narrowing.

Résumé :

Ce mémoire présente l'étude d'une approche particulière des langages de programmation logico-fonctionnels, fondée sur la logique des clauses de Horn avec égalité. Nous définissons d'abord la syntaxe et la sémantique des programmes que nous considérons. La syntaxe est celle de la logique des clauses de Horn avec égalité. La sémantique est donnée par le plus petit E-modèle de Herbrand associé à un programme.

Nous nous intéressons ensuite au calcul dans ce langage. Nous proposons pour cela une nouvelle règle appelée SLDEI-résolution comme seule règle de calcul. Nous montrons sa cohérence, sa complétude ainsi que sa complétude forte.

La mise en œuvre de la règle SLDEI-résolution nécessite un algorithme de résolution d'équations. Nous étudions de tels algorithmes fondés sur la relation de surréduction, et améliorons ces algorithmes par l'utilisation de stratégies de surréduction. Cependant, ces stratégies ne sont pas complètes dans le cas général. Nous proposons alors des conditions suffisantes sur ces stratégies afin de préserver la complétude des algorithmes considérés.

Nous caractérisons ensuite une classe de programmes, dits uniformes, pour lesquels l'utilisation de n'importe quelle stratégie de surréduction donne un algorithme complet de résolution d'équations. Nous donnons de plus une méthode de vérification de l'uniformité d'un programme. Par ailleurs, nous proposons des conditions syntaxiques pour qu'un programme soit uniforme.

Enfin, nous décrivons les principaux traits d'un langage de programmation fondé sur l'approche présentée dans ce mémoire, et l'implantation que nous avons réalisée.

Mots-clés : Programmation Logique, Spécification Algébrique, Logique des Clauses de Horn avec Egalité, SLDEI-résolution, EI-unification, Surréduction.

Remerciements

Je tiens à exprimer toute ma gratitute aux membres du jury :

• D. Bert, mon directeur de thèse, pour les conseils et encouragements qu'il m'a prodigués pendant la préparation de ce travail, et sans qui cette thèse n'aurait pas vu le jour;

• M. Dauchet et C. Kirchner pour m'avoir fait l'honneur de juger ce travail en qualité de rapporteurs. L'intérêt qu'ils ont porté à ce travail au cours de sa réalisation m'a été un encouragement précieux;

• J-P. Jouannaud qui a accépté de juger ce travail. Je suis très sensible à sa participation au jury;

• J-P. Verjus pour m'avoir fait l'honneur de présider ce jury.

Je tiens également à remercier tous ceux, qui de près ou de loin, ont participé à ce travail : en particulier

• Ph. Jorrand, directeur du laboratoire LIFIA, qui m'a accueilli au sein de son laboratoire;

• A. Bouajjani et J-M. Hufflen pour les différentes discussions que nous avons eues, leurs critiques constructives et leurs encouragements amicaux;

• F. Renzetti, responsable de la médiathèque, et toute son équipe pour l'aide précieuse qu'ils m'ont apportée dans mes recherches bibliographiques;

• O. Declerfay, B. Demeuse, P-Y. Schobbens et F. Wautier pour l'intérêt qu'ils ont porté au langage LPG1.8 et pour l'implantation qu'ils en ont faite à l'Université Catholique de Louvain la neuve;

• P. Drabik, P. Jacquet, M-L Potet, J-C Reynaud et S. Sebbar Alaoui pour les conseils et encouragements qu'ils m'ont prodigués pendant la préparation de ce travail;

• Tous les membres du LIFIA pour l'ambiance chaleureuse qu'ils ont su créer.

Sommaire

Introduction

L'étude des langages de programmation occupe une place importante dans la recherche en informatique. Cette place n'est pas surprenante lorsqu'on sait que les qualités des logiciels, comme par exemple la cohérence, la complétude, la non-ambiguïté, la réutilisabilité, la lisibilité, l'adaptabilité, la fiabilité, etc., sont directement influencées, évidemment par les méthodes de conception proposées par le Génie Logiciel [Gau89], mais surtout par les langages de programmation utilisés.

Un langage de programmation est caractérisé par sa syntaxe et sa (ou ses) sémantique(s). On distingue en général deux catégories de langages :

- Les langages d'ordre général, destinés à une utilisation dans des domaines quelconques. Cette catégorie de langages est elle-même subdivisée en d'autres sous-catégories de langages permettant des styles de programmation différents. On peut distinguer les langages impératifs (Ada, C, Pascal, etc), les langages à objets (Smalltalk, Eiffel, etc), les langages fonctionnels (Lisp, ML, FP, Miranda, etc) et les langages "logiques" (Prolog I, Prolog II, etc)

- Les langages spécifiques à des domaines particuliers, comme le parallélisme (CSP, OCCAM, etc), le temps réel (Esterel, Lustre, etc), la simulation (SIMULA), les systèmes experts (OPS5), la traduction automatique (Système-Q), etc. Pour chacun de ces domaines, on compte souvent plusieurs langages permettant des styles de programmation différents.

Chaque adepte de tel ou tel langage de programmation voit en son langage de prédilection des avantages qu'il ne trouve pas dans les autres. Par conséquent, le choix d'un langage de programmation, parmi toutes les possibilités susmentionnées (et les propositions futures), relèvera toujours en grande partie de critères subjectifs.

Dans ce mémoire, nous nous intéressons à la combinaison de deux classes de langages de programmation : les langages "logiques" et les langages fonctionnels. Par langage "logique" [Kov74], il est sous-entendu un langage issu de la logique des clauses de Horn sans égalité. Les programmes y sont décrits par des prédicats (relations). Programmer à l'aide d'un langage logique consiste à déclarer des implications comme dans

(*) Add(0, x, x)

$$Add(s(x), y, s(z)) \Leftarrow Add(x, y, z)$$
$$Pair(0)$$
$$Pair\ (s(s(x))) \Leftarrow Pair(x)$$
$$SP(x, y) \Leftarrow Add(x, y, z), Pair(z)$$

où 0 désigne la constante "zéro" et "s" désigne la fonction successeur. Les prédicats Add, Pair et SP spécifient respectivement le graphe de l'opération d'addition sur les naturels, l'ensemble des naturels pairs et l'ensemble des couples (x, y) dont la somme est paire.

L'utilisation d'un programme écrit dans un langage logique consiste à résoudre des "buts". Un but est une conjonction d'atomes comme "Add(i, j, 3)?" ou " Add(i, j, 3), Pair(i)?". La résolution d'un but revient à chercher ses solutions lorsqu'elles existent, ou plutôt une représentation finie ou infinie de ces solutions, c'est ce que l'on appelle "ensemble complet de solutions". Une solution σ pour un but B est une affectation des variables de ce but telle que $\sigma(B)$, le but obtenu en affectant les variables de B par la solution σ, soit une conjonction valide (par rapport au programme considéré). Par exemple, l'affectation $\sigma=\{i\rightarrow 0, j\rightarrow 3\}$ est une solution pour le but "Add(i, j, 3)?".

La sémantique prise pour un programme logique est son modèle initial (on dit aussi le modèle standard ou le plus petit modèle de Herbrand [Llo87][EK76]). Cette sémantique correspond plus à l'intuition du programmeur que celle habituellement adoptée dans le domaine de la démonstration automatique, où un "programme" dénote toute la classe de ses modèles. En effet, considérons par exemple les deux programmes suivants :

$$\{\ Pair1\ (x) \Leftarrow Add(y, y, x)\}$$

$$\{\ Pair2(0)$$
$$Pair2(s(s(x))) \Leftarrow Pair2(x)\ \}$$

Ces deux programmes sont distinguables si l'on définit la sémantique d'un programme comme étant la classe de tous ses modèles. Par contre, lorsqu'on réduit cette classe au modèle initial, les deux programmes Pair1 et Pair2 deviennent équivalents, c'est-à-dire que les prédicats Pair1 et Pair2 dénotent le même sous-ensemble des naturels pairs, ce qui répond bien à l'intuition du programmeur. Quant à la sémantique opérationnelle de ces langages, elle est donnée par la règle SLD-résolution proposée par Kowalski et Kuehner [KK71]. Cette règle est une stratégie particulière de la Résolution de Robinson [Rob65], adaptée à la logique des clauses de Horn sans égalité.

L'aspect déclaratif des langages logiques, les rapprochant de l'utilisateur, leur a conféré un succès évident depuis leur apparition au début des années 1970. De plus, ces langages gardent une certaine similitude avec les langages classiques (procéduraux), comme l'a souligné Kowalski dans [Kov74] où il en donne une interprétation procédurale. La programmation logique trouve ses applications dans des domaines de plus en plus nombreux;

nous citons par exemple les Bases de Données relationnelles ou l'Intelligence Artificielle (programmation de systèmes experts). Par contre l'absence de contrôle de types et l'impossibilité de définir des fonctions constituent les principales critiques envers les langages logiques actuels.

Les langages fonctionnels, quant à eux, permettent un autre style de programmation. Les programmes y sont définis exclusivement à l'aide de fonctions. Ces langages, contrairement aux langages logiques, connaissent plusieurs fondements différents, notamment le λ-calcul de Church qui a influencé la définition du langage LISP [McC60] ou ses dérivés, comme ML[GMW79]. D'un autre côté, la logique combinatoire sous-tend une autre classe de langages fonctionnels, par exemple le langage FP[Bac78]. Les deux approches de langages fonctionnels que nous venons de mentionner donnent lieu à des langages de programmation d'ordre supérieur. Ceci permet une certaine concision des programmes et augmente leur réutilisabilité.

Une troisième approche des langages fonctionnels, que nous considérons particulièrement dans ce travail, est constituée par les langages "algébriques" [EM85] [Wir89]. Cette troisième approche se base sur la logique équationnelle[1] ou conditionnelle[2]. Les programmes y sont décrits par des équations, éventuellement conditionnelles, comme dans

(**) : 0 + x == x
s(x) + y == s(x+y)
pair(0) == vrai
pair(s(0)) == faux
pair(s(s(x))) == pair(x)
sp(x, y) == pair(x+y)

où les deux premières équations constituent la définition de l'opération d'addition (+) sur les naturels ; les trois dernières équations définissent les fonctions caractéristiques des prédicats Pair et SP vu en (*). Le symbole == désigne le prédicat d'égalité.

La méthode de programmation algébrique date des années 1974-75 et fait suite aux travaux de Liskov et Zilles [LZ74], Guttag [Gut75] et du groupe ADJ [GTWW75]. L'avantage majeur de ce style de programmation est l'abstraction ; elle induit une relative facilité dans la conception des programmes. Cependant, les langages algébriques perdent les

---

[1]Un programme dans la logique équationnelle est un ensemble d'équations.

[2]Un programme dans la logique conditionnelle est un ensemble de clauses de Horn où le seul prédicat utilisé est le prédicat d'égalité.

avantages des langages d'ordre supérieur rencontrés dans les deux approches précédentes, mais cette lacune est en partie comblée par la généricité [Gog88]. La sémantique d'un programme algébrique est son modèle initial. L'utilisation de tels programmes consiste habituellement à évaluer des expressions appelées termes, comme "factorielle(8)" ; la réponse escomptée de l'évaluation d'un terme est la "valeur" de ce terme. Cette valeur est donnée par une sémantique opérationnelle basée sur les systèmes de réécriture, un système de règles de réécriture étant un ensemble d'équations orientées de gauche à droite [DJ89].

La course vers un langage de programmation logico-fonctionnel fut lancée par Robinson et Sibert en proposant le langage LOGLISP [RS81] en 1981. Depuis, une pléthore de propositions a vu le jour. On trouve dans [DL86] une compilation d'un échantillon significatif des langages proposés. Ces langages peuvent être groupés en différentes classes. Nous classons ces propositions en cinq groupes:

1- Les langages combinant le λ-calcul et les clauses de Horn (par ex. LOGLISP [RS81]),

2- Les langages basés sur la logique des clauses de Horn avec égalité (par ex. EQLOG [GM84]),

3- Les langages basés sur la logique conditionnelle (par ex. SLOG [Fri85]),

4- Les langages logiques avec contraintes (par ex. PROLOG III [Col87]) et

5- Les autres langages (par ex. UNIFORM [Kah86] qui allie les programmations fonctionnelle, logique et à objets)

Nous considérons dans ce mémoire la deuxième classe des langages logico-fonctionnels. Ces langages sont donc fondés sur la logique des clauses de Horn avec égalité. Cette logique combine les langages algébriques et logiques de façon "naturelle". Elle permet ainsi de bénéficier des avantages des deux approches, algébrique et logique, en plus des avantages de leur combinaison. Les exemples (*) et (**) ci-dessus peuvent être réécrits dans cette logique en

$$0 + x == x$$
$$s(x) + y == s(x+y)$$
$$Pair(x+x)$$
$$SP(x, y) \Leftarrow Pair\ (x+y)$$

Ces définitions font clairement la distinction entre les fonctions et les prédicats. Ceci n'était pas le cas dans les programmes purement logiques où les fonctions sont codées par leurs graphes (par ex. Add dans (*)), ou dans les programmes purement fonctionnels où les prédicats sont codés par leurs fonctions caractéristiques (par ex. pair et sp dans (**)).

L'utilisation des programmes écrits dans la logique des clauses de Horn avec égalité consiste à résoudre des "buts" à la manière des langages logiques ; ceci n'exclut pas d'ailleurs les évaluations à la façon des langages algébriques. Un but est donc une conjonction d'atomes, par exemple "Pair (x), $x \leq 3$ == true ?" ou "i + j == 3?" .

Nous nous intéressons dans la présente étude à la sémantique opérationnelle de tels langages, autrement dit l'étude des moyens de résolution des buts en présence d'un programme donné. La définition de ces moyens n'a de sens qu'avec une définition rigoureuse des programmes. Un programme sera défini comme une présentation de théorie $SP=(\Sigma, C)$ où Σ est une signature (c'est-à-dire un ensemble de déclarations de types, de fonctions et de prédicats) et C un ensemble de clauses de Horn avec égalité, définissant les symboles de la signature Σ. La sémantique que nous donnons à un tel programme est son modèle initial.

Considérant cette sémantique, nous introduisons informellement la notion de calcul et de ses principales caractéristiques, à savoir la complétude et la cohérence. Pour le développement d'un calcul de résolution des buts, nous nous sommes inspiré des travaux antérieurs dans le domaine de la démonstration automatique [Lov78] [CL73] [Rus89]. Nous proposons un système déductif constitué d'une seule règle d'inférence que nous appelons SLDEI-résolution. Cette règle est une extension de la règle SLD-résolution proposée dans le cadre de la logique des clauses de Horn sans égalité, par Kowalski et Kuehner [KK71] . La règle SLDEI-résolution est obtenue à partir de la règle SLD-résolution en remplaçant l'unification par la résolution d'équations dans le modèle initial. Parallèlement à notre travail, Gallier et Raatz [GR89] ainsi que Hölldobler [Höl88] ont proposé (avec deux preuves différentes) la règle SLDE-résolution. Cette dernière règle est implicite dans [GM84] et sous une forme très proche dans [JLM84]. La différence entre la SLDEI-résolution et la SLDE-résolution réside dans les résolutions d'équations : dans la première, les équations sont résolues dans le modèle initial, alors que la seconde résout les équations par rapport à toute la classe des modèles du programme considéré. La différence entre les deux règles vient de la différence des sémantiques pour lesquelles les règles doivent être adéquates.

Une partie importante pour la mise en œuvre de la règle SLDEI-résolution est la résolution d'équations (on dit aussi : E-unification). Là encore, nous nous sommes inspiré des travaux antérieurs, effectués dans le domaine de la E-unification [Kir85][Siek89]. Ce domaine est vaste, on y trouve des études d'algorithmes ciblés pour des théories particulières, comme la commutativité, l'associativité-commutativité, etc. (voir [Siek89] pour un résumé de ces résultats) jusqu'aux algorithmes généraux de E-unification [GS88]. Dans notre étude, nous avons utilisé la relation de surréduction (dite *narrowing* en anglais), introduite par Slagle dans [Sla74]. Son utilisation dans un algorithme de E-unification a été illustré pour la première fois par Fay dans [Fay79]. Notre choix d'utiliser la surréductions

pour la résolution d'équations, a été motivé d'une part par la forme des spécifications équationnelles, celles-ci étant souvent définies par des systèmes de réécriture [HO80][GJ89], et d'autre part, par la généralité de l'algorithme basé sur la surréduction qui sied aux langages algébriques, où les structures de données sont définissables par l'utilisateur.

Les algorithmes de E-unification basés sur la surréduction ont été très étudiés pendant la dernière décennie. Intuitivement, la résolution par surréduction d'une équation t == t' consiste à instancier les variables de l'équation à résoudre "de proche en proche" selon les équations du programme considéré, tout en vérifiant la validité des équations instanciées ainsi obtenues. Par exemple, pour résoudre l'équation (1) i+j==2 en utilisant les règles de définition de l'addition (**), la surréduction tentera par exemple de résoudre les deux équations (2) j == 2 et (3) $s(i_1+j) == 2$ obtenues respectivement en instanciant la variable i par 0 et par $s(i_1)$. Une solution pour l'équation initiale (1) se déduit de toute solution des équations (2) ou (3).

L'algorithme de Fay [Fay79] cité plus haut contient deux niveaux d'indéterminisme, l'un sur les règles de réécriture (équations orientées) utilisées et le second sur le choix des positions de surréduction (grosso modo le choix des variables à instancier). Le premier indéterminisme est intrinsèque à la méthode et ne peut donc être éliminé. Par contre, le deuxième indéterminisme n'est pas toujours nécessaire. Plusieurs propositions ont été faites dans le but de le diminuer. Ainsi, Hullot [Hul80] puis Herold [Her86], Réty [Rét87], Bosco et al [BGM87], Nutt et al [NRS87] et Hölldobler [Höl89] ont proposé des stratégies améliorant ce non déterminisme en utilisant des variantes de la surréduction basique.

Dans notre étude, nous avons essayé de supprimer ce non déterminisme en utilisant des stratégies de choix de position de surréduction. Ces stratégies ressemblent à celles utilisées dans la SLD-résolution pour le choix de l'atome "candidat" à la Résolution. Cependant, la comparaison avec la SLD-résolution en reste là, puisque les stratégies de surréduction ne sont pas complètes dans le cas général contrairement aux stratégies de SLD-résolution. Fribourg a considéré le cas particulier de la stratégie innermost et a proposé dans [Fri85] des conditions syntaxiques assurant la complétude de la surréduction par cette stratégie. Nous généralisons ces résultats en proposant des conditions suffisantes sur les stratégies qui assurent la complétude de la résolution d'équations par surréduction. Ces conditions sont d'ordre "sémantique". Elles expriment une certaine relation entre la réduction (la relation sur les termes induite par les règles de réécriture) et la surréduction. Indépendamment de notre travail, Padawitz a proposé dans [Pad87] [Pad88] des résultats similaires sous des définitions et conditions légèrement différentes des nôtres (par ex. sa définition de stratégie de surréduction est différente de la nôtre), pour lesquelles il a montré la complétude de la E-unification par surréduction.

Nous poursuivons notre étude en caractérisant une classe de programmes, dits "uniformes", pour lesquels l'utilisation de n'importe quelle stratégie de surréduction donne un algorithme complet de E-unification. Nous donnons par la même occasion une méthode pour vérifier l'uniformité d'un programme. Cette méthode de vérification repose en partie sur le même "outil" que les méthodes de test de la réductibilité inductive [JK89] [Pla85]. Cette outil est une représentation finie de l'ensemble des substitutions fermées normalisées [Com88] [Kou90]. D'un autre côté, nous introduisons des conditions syntaxiques sur les programmes [Ech88] qui assurent leur uniformité. Ces conditions nous permettent en particulier de retrouver un résultat de You sur le E-filtrage paru dans [You88].

Le travail que nous présentons ici découle de notre étude concernant la mise en œuvre du langage LPG1.8 [BE86], un langage basé sur la logique des clauses de Horn avec égalité.

Plan du mémoire

Le chapitre 1 rappelle les notions de bases sur la logique des clauses de Horn avec égalité. Nous y définissons la notion de programme par la syntaxe et la sémantique.

Le chapitre 2 introduit deux calculs complets et cohérents. Le premier, classique, est constitué de la règle SLD-résolution. Ce calcul nécessite les axiomes d'égalité. Le deuxième calcul est la SLDEI-résolution qui est défini et montré cohérent et complet.

Le chapitre 3 traite de la E-unification par surréduction. Nous y introduisons la notion de stratégie de surréduction puis donnons des conditions suffisantes sur les statégies assurant la complétude de la E-unification.

Le chapitre 4 propose une classe de programmes pour lesquels toutes les stratégies de surréduction satisfont les conditions du chapitre 3. Une méthode de vérification de ces programmes y est donnée ainsi que des conditions suffisantes d'ordre purement syntaxique assurant l'appartenance d'un programme à cette classe.

Le chapitre 5 introduit informellement le langage LPG1.8. Ce chapitre peut être lu de façon indépendante des chapitres 2, 3 et 4, excepté le §5.7 qui présente succinctement l'implantation que nous avons réalisée.

Note : Pour rendre possible une lecture indépendante des chapitres 2 et 3, certaines définitions ont été répétées dans ces deux chapitres.

Chapitre 1

La logique sous-jacente

Nous introduisons dans ce chapitre préliminaire, la logique sous-jacente à "notre" langage de programmation. Il s'agit de la logique des clauses de Horn avec égalité multi-sortes. Nous définissons dans un premier temps les spécifications dans cette logique d'un point de vue syntaxique (§1.1), ensuite nous présentons la sémantique que nous avons retenue pour les spécifications et en donnons trois descriptions différentes: comme intersection des E-modèles de Herbrand (§1.2.1), comme modèle initial dans la catégorie des E-modèles (§1.2.2) et enfin en tant que plus petit point fixe d'une transformation sur les E-modèles de Herbrand (§1.2.3).

1.1. SYNTAXE

Nous introduisons dans ce paragraphe la syntaxe des spécifications écrites dans la logique des clauses de Horn avec égalité. Nous rappelons d'abord les définitions de signature du premier ordre avec ou sans égalité, de terme, d'équation, de formule atomique et de clause de Horn. Nous définissons ensuite la notion de spécification dans la logique des clauses de Horn avec égalité.

Définition 1.1 : (signature)

Une signature du premier ordre Σ est un triplet $\Sigma = (S, \Omega, \Pi)$ où

– S est un ensemble (de noms de sortes),

– Ω est une famille S^+-indicée d'ensembles (de symboles d'opérateurs),(S^+ étant l'ensemble des chaînes (mots) non vides constituées des éléments de S),

$\Omega = \{\Omega_{us} \mid u \in S^*, s \in S\}$ avec $S^* = S^+ \cup \{\varepsilon\}$ où ε dénote la chaîne vide

– Π est une famille S^+-indicée d'ensembles (de symboles de prédicats).

$\Pi = \{\Pi_u \mid u \in S^+\}$

Un opérateur f, élément d'un ensemble Ω_{us}, est dit d'**arité** u, de **sorte** s et de **profil** u,s. Un tel opérateur sera noté f_{us} ou $f : u \rightarrow s$. De même, un symbole de prédicat P, élément d'un ensemble Π_u, est dit d'arité u. On le notera P_u ou P: u.

Définition 1.2 : (signature avec égalité)

Une signature du premier ordre avec égalité est une signature du premier ordre $\Sigma = (S, \Omega, \Pi)$ telle que : $\forall s \in S, =_{ss} \in \Pi_{ss}$. Le symbole $=_{ss}$ est le prédicat d'égalité relatif à s.

Intuitivement, un symbole d'égalité $=_{ss}$ est un moyen syntaxique pour désigner l'identité sur les ensembles (cf. définition 1.9).

Un ensemble de variables X sera vu comme une famille S-indicée d'ensembles X_s de variables de même sorte ($X = \cup_{s \in S} X_s$).

Définition 1.3 : (terme)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature et X un ensemble de variables. Pour toute sorte s dans S, l'ensemble des termes de sorte s, noté $T_s(\Sigma, X)$, est inductivement défini par:

- $X_s \subseteq T_s(\Sigma, X)$ -- les variables de sorte s sont des termes de sorte s
- $\Omega_s \subseteq T_s(\Sigma, X)$ -- les constantes de sorte s (opérateurs d'arité ε) sont des termes de sorte s
- $\forall f \in \Omega_{s_1 \ldots s_n s}, \forall t_i \in T_{s_i}(\Sigma, X)$ pour $i = 1 \ldots n$, $f(t_1, \ldots t_n) \in T_s(\Sigma, X)$,
- Tous les éléments de $T_s(\Sigma, X)$ sont formés à partir des trois axiomes ci-dessus.

Un terme t est dit **fermé** s'il ne contient pas de variables. L'ensemble des termes fermés de sorte s est noté $T_s(\Sigma, \emptyset)$ ou $T_s(\Sigma)$. On notera $T(\Sigma, X)$ (resp. $T(\Sigma)$) la famille S-indicée des ensembles $T_s(\Sigma, X)$ (resp. $T_s(\Sigma)$).

Nous supposons dans ce mémoire que pour tout symbole s de S $T_s(\Sigma)$ n'est pas vide. Cette hypothèse nous évitera quelques cas pathologiques pour la cohérence des calculs [GM87a][HO80].

Définition 1.4 : (atome)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature et X une famille S-indicée d'ensembles X_s de variables. L'ensemble des atomes $A(\Sigma, X)$, est inductivement défini par :

- $\forall P \in \Pi_{s_1 \ldots s_n}, \forall t_i \in T_{s_i}(\Sigma, X)$ pour $i = 1 \ldots n$, $P(t_1, \ldots t_n) \in A(\Sigma, X)$
- Tous les éléments de $A(\Sigma, X)$ sont formés à partir de l'axiome ci-dessus.

Définition 1.5 : (équation)

Une équation est un atome où le symbole de prédicat est un symbole d'égalité $=_{ss}(t_1, t_2)$. Dans la suite, on notera un tel atome (équation) $t_1 == t_2$.

Définition 1.6 : (clause de Horn)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature et X une famille S-indicée d'ensembles X_s de variables. Soient $A_0, A_1, \ldots, A_n$ des atomes (éléments de $A(\Sigma, X)$). Une clause de Horn est une formule qui se présente sous l'une des trois formes suivantes:

(i) A_0

(ii) $A_1, \ldots, A_n \Rightarrow A_0$

(iii) $A_1, \ldots, A_n \Rightarrow$

une clause de la forme (i) est souvent appelée **fait**. Une clause de la forme (iii) est appelée **clause de but**. Dans la clause (ii), les atomes $A_1, \ldots, A_n$ représentent le **corps** de la clause, l'atome A_0 est la **tête** de la clause.

Toutes les variables figurant dans les atomes d'une clause de Horn sont universellement quantifiées.

Notation : Conformément à la littérature "informatique" nous écrirons les clauses de la forme (ii) et (iii) comme ci-dessous où les corps de clauses figurent à droite des têtes de clauses:

(ii) $A_0 \Leftarrow A_1, \ldots, A_n$

(iii) $\Leftarrow A_1, \ldots, A_n$

En plus des trois formes de clauses définies ci-dessus, nous en introduisons une quatrième, à savoir la **clause vide**. Cette clause, notée [], symbole de la contradiction, est utilisée dans les raisonnements par réfutation (cf. chapitre 2).

Nous pouvons maintenant définir les spécifications de manière syntaxique.

Définition 1.7 : (spécification)

On appelle **spécification** dans la logique des clauses de Horn avec égalité, un couple **SP= (Σ, C)** où Σ est une signature du premier ordre avec égalité et C un ensemble fini de clauses de Horn de la forme (i) ou (ii).

Remarque 1.1 : Les clauses de la dernière forme (iii) serviront à exprimer les buts à résoudre.

Exemple 1.1 : Nous donnons ci-dessous un exemple de spécification . Les opérations ou prédicats spécifiés sont classiques. Nous sous-entendons les déclarations des variables et de leurs sortes. Nous donnons ensuite deux exemples de clauses de but relatives à la spécification ci-dessous.

SP = (Σ, C) avec $\Sigma = (S, \Omega, \Pi)$ telle que :

S ={nat, bool}

Ω = {{0 : → nat}, {s : nat → nat}, {+ : nat nat → nat, * : nat nat → nat},{v : → bool, f : → bool}, {pair : nat → bool}}

Π = {{Pair : nat, Impair : nat}, {CB : nat nat nat nat}, {== : nat nat}, {== : bool bool}}

C ={ 0 + x == x
s(x) + y == s(x + y)
0 * x == 0
s(x) * y ==y + (x * y)
Pair (0)
Impair (s(0))
Pair(s(x)) ⇐ Impair (x)
Impair (s(x)) ⇐ Pair (x)
pair (x) == v ⇐ Pair (x)
pair (x) == f ⇐ Impair (x)
CB(c, o, c+o, c+c+c+c+o+o) } -- Intuitivement, dans CB(c, o, t, p), c, o, t et p représentent respectivement, le nombre de chats, le nombre d'oiseaux, le nombre de têtes (des chats et des oiseaux) et le nombre de pattes.

exemples de clauses de but :
⇐ i * i == s(0)
⇐ CB(i, j, s(s(s(0))), k), Pair (i)

1.2. SEMANTIQUE

Dans ce paragraphe nous nous intéressons à la sémantique des spécifications telles que nous les avons définies. Evidemment, plusieurs possibilités existent pour définir la sémantique d'une spécification. On peut distinguer essentiellement deux sémantiques dans la littérature : la sémantique lâche (loose) et la sémantique initiale. Notre choix a porté sur cette dernière pour son adéquation avec les langages de programmation. Nous en donnons trois caractérisations différentes: comme intersection de E-modèles de Herbrand, comme objet initial dans la catégorie des E-modèles de SP et enfin, comme plus petit point fixe d'une transformation sur les E-interprétations de Herbrand. Nous commençons d'abord par définir les notions de modèle et E-modèle d'une spécification.

1.2.1. Modèles

Ce paragraphe présente une première définition dudit E-modèle standard d'une spécification donnée. Pour cela nous rappelons dans un premier temps, les définitions

classiques d'interprétation (respectivement de E-interprétation) et de modèle (respectivement de E-modèle) correspondant aux cas des signatures du premier ordre sans égalité (respectivement avec égalité). Nous introduisons après, les notions de modèle et E-modèle de Herbrand et terminons par la définition du E-modèle standard en tant qu'intersection des E-modèles de Herbrand.

1.2.1.1. Interprétation , E-interprétation

Définition 1.8 : (interprétation)

Soit $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre. On appelle **Σ-structure** ST (ou Σ-interprétation) la donnée d'un triplet $ST = (A, \alpha, \beta)$ où :

– A est une famille S-indicée d'ensembles non-vides : $A = \{A_s \mid s \in S\}$. Ces ensembles sont appelés ensembles supports de la structure.

– α est une famille $S^* \times S$-indicée d'applications : cette famille est l'interprétation des symboles de fonctions dans la structure.

$\alpha = \{\alpha_{us}: \Omega_{us} \rightarrow [A_u \rightarrow A_s] \mid u \in S^* \text{ et } s \in S\}$ où

$A_{s_1 \ldots s_n}$ désigne le produit cartésien $A_{s_1} \times \ldots \times A_{s_n}$, et

$[A \rightarrow B]$ désigne l'ensemble des applications de A vers B.

– β est une famille S^+-indicée d'applications : cette famille est l'interprétation des symboles de prédicats dans la structure.

$\beta = \{\beta_u: \Pi_u \rightarrow 2^{A_u} \mid u \in S^+\}$ où

2^A désigne l'ensemble des parties de A

Nous notons $\mathbf{STR}_\Sigma$ la classe des Σ-structures. Nous omettrons dans la suite les indices des applications α_{us} et β_u s'ils peuvent être facilement déduits du contexte. Nous éviterons aussi d'écrire Σ dans Σ-structure ou Σ-interprétation.

En présence d'une signature Σ avec égalité, les Σ-interprétations ne reflètent pas toujours l'idée que l'on peut avoir sur la notion d'égalité. Pour cela on introduit la notion de E-interprétation qui associe au prédicat d'égalité une interprétation univoque.

Définition 1.9 : (E-interprétation[1])

Soit $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre avec égalité. On appelle **E-interprétation** une Σ-structure $ST = (A, \alpha, \beta)$ définie comme ci-dessus, mais en imposant de plus le fait d'interpréter tout symbole de prédicat d'égalité $=_{ss}$ comme étant l'identité sur l'ensemble support A_s. Autrement dit : $\forall s \in S, \beta(=_{ss}) = \{(x, x) \mid x \in A_s\}$.

---

[1]Classiquement [CL73], on définit une E-interprétation comme une interprétation "satisfaisant" (voir définition 1.11) l'ensemble d'axiomes AXE(Σ) (voir définition 1.20). Ce qui n'est pas le cas ici.

Remarque 1.2 : Soit $\Sigma = (S, \Omega, \Pi)$ une signature avec égalité. La différence essentielle entre une E-interprétation et une interprétation de Σ réside dans l'interprétation des symboles d'égalité. En effet dans une E-interprétation le choix des ensembles supports fige l'interprétation des prédicats d'égalité, alors que dans une interprétation les prédicats d'égalité, s'ils existent, sont considérés (par définition) au même titre que les autres symboles de prédicats.

Exemple 1.2 : (interprétation, E-interprétation)
Soit la signature $\Sigma = (S, \Omega, \Pi)$ avec

$S = \{nat\}$

$\Omega = \{\{0 : \rightarrow nat\}, \{s : nat \rightarrow nat\}, \{+ : nat\ nat \rightarrow nat\}\}$

$\Pi = \{\{== : nat\ nat\}\}$

Soient $ST1 = (A_1, \alpha_1, \beta_1)$ et $ST2 = (A_2, \alpha_2, \beta_2)$ deux interprétations de Σ définis par:

$A_{1nat} = \mathbb{N}$; $\alpha_1(0) = 0$, $\alpha_1(s) = inc$, $\alpha_1(+) = add$; $\beta_1(==) = \{(i, j) \in \mathbb{N}^2 \mid i$ et j sont de même parité$\}$

où *add* est la fonction d'addition et *inc* la fonction "+1".

$A_{2nat} = \{[0], [1]\}$; $\alpha_2(0) = [0]$, $\alpha_2(s) = not$, $\alpha_2(+) = oux$; $\beta_1(==) = \{([0], [0]), ([1], [1])\}$

où les opération *not* et *oux* sont définies comme suit :

not ([0]) = [1], not ([1]) = [0],
oux ([0],[0]) = [0], oux ([1],[1]) = [0],
oux ([1],[0]) = [1], oux ([0],[1]) = [1].

L'interprétation ST1 n'est pas une E-interprétation de la signature Σ car $\beta_1(==)$ ne reflète pas l'identité sur l'ensemble support $\mathbb{N}$. Par contre ST2 est une E-interprétation de Σ.

Remarque 1.3 : il est clair que toute E-interprétation d'une spécification en constitue une interprétation.

1.2.1.2. Satisfaction, modèle, E-modèle

La définition suivante introduit la notion d'affectation (on dit aussi assignation). Intuitivement, une affectation -ou plutôt son extension sur les termes- interprète un terme

"syntaxique" dans une structure donnée. Cette notion sera utile pour définir la relation de satisfaction, et par la suite, les modèles des spécifications.

Définition 1.10 : (affectation)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature, X une famille S-indicée d'ensembles de variables ($X = \cup_{s \in S} X_s$) et $ST = (A, \alpha, \beta)$ une Σ-structure. On appelle **affectation de sorte s** une application $a_s: X_s \rightarrow A_s$. Une **affectation** *a* est une famille S-indicée d'affectations de sorte s c.-à-d. $a = \{a_s: X_s \rightarrow A_s \mid s \in S\}$. Pour définir la validité des formules nous utiliserons l'unique extension (propriété des structures libres) de *a* sur les termes que l'on note $a^\#$. $a^\#$ est donc la famille S-indicée des extensions des affectations de sorte s : $a^\# = \{a_s^\# : T_s(\Sigma, X) \rightarrow A_s \mid s \in S\}$. $a_s^\#$ est inductivement définie comme suit :

- $\forall x \in X_s$, $a_s^\#(x) = a_s(x)$
- $\forall c \in \Omega_s$, $a_s^\#(c) = \alpha(c)$
- $\forall f \in \Omega_{s_1 \ldots s_n s}, \forall t_i \in T_{s_i}$, $a_s^\#(f(t_1,\ldots,t_n)) = \alpha(f)(a^\#_{s_1}(t_1),\ldots,a^\#_{s_n}(t_n))$

Définition 1.11 : (relation de satisfaction - validité)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature, X une famille S-indicée d'ensembles de variables et $ST = (A, \alpha, \beta)$ une Σ-structure. Soit $a = \{a_s: X_s \rightarrow A_s \mid s \in S\}$ une affectation, on dit :

- $P(a^\#_{s_1}(t_1),\ldots,a^\#_{s_n}(t_n))$, instance par a de l'atome $P(t_1,\ldots,t_n)$ que l'on écrira par abus de notation $a^\#(P(t_1,\ldots,t_n))$, est **vrai** dans l'interprétation ST ssi le n-uplet $(a^\#_{s_1}(t_1),\ldots,a^\#_{s_n}(t_n))$ appartient à $\beta(P)$ (rappelons que $\beta(P)$ est l'interprétation du symbole de prédicat P dans la structure ST), $a^\#(P(t_1,\ldots,t_n))$ est faux sinon.

- Une clause de la forme "$P(t_1,\ldots,t_n)$", un fait, est **valide** dans ST ssi pour toute affectation a, $a^\#(P(t_1,\ldots,t_n))$ est vrai dans ST.

- Une clause de la forme "$A_1,\ldots,A_n \Rightarrow A_0$" est **valide** dans ST ssi pour toute affectation a, $a^\#(A_0)$ est vrai dans ST chaque fois que les instances $a^\#(A_i)$, pour $i=1,\ldots,n$, sont vraies dans ST.

- Une clause de la forme "$A_1,\ldots,A_n \Rightarrow$",une clause de but, est valide dans ST ssi pour toute affectation a, il existe au moins un atome A_j avec $1 \leq j \leq n$ tel que $a^\#(A_j)$ soit faux.

Pour une clause c, on note **ST $\models$ c** la proposition "c est **valide** dans ST". On dit aussi, dans ce cas, que ST **satisfait** la clause c.

Nous introduisons maintenant les notions de modèle et de E-modèle d'une spécification.

Définition 1.12 : (modèle)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature (sans égalité), SP = (Σ, C) une spécification et ST = (A, α, β) une Σ-structure. ST est un **modèle** de SP ssi toutes les clauses dans C sont valides dans ST. On notera par $\mathbf{Mod_{SP}}$ la classe des modèles de la spécification SP.

Définition 1.13 : (E-modèle)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature avec égalité, SP = (Σ, C) une spécification et ST = (A, α, β) une E-interprétation. ST est un **E-modèle** de SP ssi toutes les clauses dans C sont valides dans ST. On notera par $\mathbf{E\text{-}Mod_{SP}}$ la classe des E-modèles de la spécification SP.

Exemple 1.3 : (modèle, E-modèle)

Considérons la spécification suivante : SP = (Σ, C) avec $\Sigma = (S, \Omega, \Pi)$ telle que

S ={nat}

Ω = {{0 : $\rightarrow$ nat}, {s : nat $\rightarrow$ nat}, {+ : nat nat $\rightarrow$ nat} }

Π = {{Pair : nat, Impair : nat}, {== : nat nat}}

C ={ 0 + x == x

s(x) + y == s(x + y)

Pair (0)

Pair(s(x)) $\Leftarrow$ Impair (x)

Impair (s(x)) $\Leftarrow$ Pair (x) }

Soient ST1 = (A_1, α_1, β_1) et ST2 = (A_2, α_2, β_2) deux interprétations de Σ définis par:

$A_{1nat} = \mathbb{N}$; α_1 (0) = 0 , α_1 (s) = inc , α_1 (+) = add ; β_1 (==) ={(i, i) $\in \mathbb{N}^2$ | i $\in \mathbb{N}$ }, β_1 (Pair) = β_1 (Impair) = $\mathbb{N}$

où add est la fonction d'addition et inc la fonction "+1".

$A_{2nat} = \mathbb{N}$; $\alpha_2 = \alpha_1$, β_2 (==) =β_1 (==), β_2 (Pair) = $\mathbb{N}$, β_2 (Impair) ={2i+1 | i $\in \mathbb{N}$ }.

Les deux structures ST1 et ST2 sont deux E-interprétations pour Σ. ST1 est un E-modèle de SP. ST2 n'est pas un E-modèle de SP parce que, par exemple, Pair(s(0)) est vrai dans ST2 mais Impair (s(s(0))) n'est pas vrai dans ST2, ce qui rend la dernière clause non valide dans ST2.

Définition 1.14 : (**satisfaisabilité**: suite)

Soient Σ une signature et C un ensemble de clauses de Horn, on dit que :

- l'ensemble C est satisfaisable s'il existe au moins un modèle pour C
- l'ensemble C est insatisfaisable s'il n'existe pas de modèle pour C

Définition 1.15 : (conséquence logique)

Soit SP = (Σ, C) une spécification. Une clause (sur la même signature) Cl est une **conséquence logique** des clauses dans C si la clause Cl est valide dans tout modèle de SP.

1.2.1.3. E-modèle standard

Classiquement [Llo87] [EK76], une spécification SP = (Σ , C), où C est un ensemble de clauses de Horn, a pour sémantique le plus petit modèle de Herbrand (ou modèle standard) . Nous rappelons brièvement ci-dessous les définitions des interprétations et modèles de Herbrand dans le cas classique, c'est-à-dire le cas de la logique des clauses de Horn sans égalité. Ces définitions n'étant pas satisfaisantes lorsqu'on tient compte du prédicat d'égalité, nous introduisons dans un deuxième temps, les E-interprétations et E-modèles de Herbrand dans le cas des clauses de Horn avec égalité. Ce qui nous permettra par la suite de définir le E-modèle standard en tant qu'intersection des E-modèles de Herbrand.

Définition 1.16 : (interprétation de Herbrand)

Soit $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre. On appelle **interprétation de Herbrand** une Σ-structure particulière **IH** = $(T(\Sigma), \alpha, \beta)$ où

- $T(\Sigma)$ est la famille des ensembles de termes fermés
- La famille α est définie dans ce cas comme suit:

$$\forall\, c \in \Omega_s,\ \alpha(c) = c$$

$$\forall\, f \in \Omega_{s_1 \ldots s_n s}, \forall\, t_i \in T_{s_i}(\Sigma),\ i = 1 \ldots n,\ \alpha(f)\,(t_1, \ldots t_n) = f(t_1, \ldots t_n)$$

- La famille β vérifie:

$$\forall\, P \in \Pi_{s_1 \ldots s_n},\ \beta(P) \subseteq T_{s_1}(\Sigma) \times \ldots \times T_{s_n}(\Sigma)$$

On remarquera que les interprétations de Herbrand ne se différencient entre elles que par les interprétations des prédicats (la famille β). Les ensembles supports et l'interprétation des symboles de fonctions étant fixés.

Définition 1.17 : (modèle de Herbrand)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature sans égalité et SP = (Σ, C) une spécification. Un Σ-modèle de Herbrand de SP est une Σ-interprétation de Herbrand qui est modèle de SP.

Exemple 1.4 : Soit la spécification SP ci-dessous dont nous ne donnons que les clauses, la signature s'en déduisant facilement. x et y sont des variables, 0 est une constante

et s est une fonction unaire. Nous utilisons deux symboles de prédicats différents : P et ==. Ici, le symbole == a le même statut que P.

$P(0, s(0))$

$s(x) == s(s(y)) \Leftarrow P(x, y)$

Soient les deux interprétations de Herbrand IH1 = $(T(\Sigma), \alpha, \beta_1)$ et IH2 = $(T(\Sigma), \alpha, \beta_2)$ avec β_1 et β_2 définis comme suit :

$\beta_1(==) = \{(s(0), s(s(s(0))))\} \cup \{(s(i), s(s(i))) \mid i \in T(\Sigma)\}$

$\beta_1(P) = \{(0, s(0))\} \cup \{(i, i) \mid i \in T(\Sigma)\}$

$\beta_2(==) = \{(s(0), s(s(s(0)))), (0, s(0))\}$

$\beta_2(P) = \{(0, s(0))\}$

IH1 et IH2 sont tous les deux modèles de la spécification SP.

Définition 1.18 : (intersection de modèles de Herbrand)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature sans égalité, SP = (Σ, C) une spécification et IH = $(T(\Sigma), \alpha, \beta)$ et IH' = $(T(\Sigma), \alpha, \beta')$ deux modèles de Herbrand de SP. On définit l'intersection de IH et IH' par IH $\cap$ IH' = $(T(\Sigma), \alpha, \beta'')$ où β'' est définie comme suit :

- $\forall P \in \Pi_u$, $\beta''(P) = \beta(P) \cap \beta'(P)$

Exemple 1.5 : Considérons à nouveau la spécification de l'exemple 1.4 et ses deux modèles IH1 et IH2. Soit IH3 = $(T(\Sigma), \alpha, \beta_3)$ l'intersection de IH1 et IH2. β_3 est définie par

$\beta_3(==) = \{(s(0), s(s(s(0))))\}$

$\beta_3(P) = \{(0, s(0))\}$

Notons que IH3 est aussi un modèle de SP. Plus généralement, on a la propriété suivante :

Proposition 1.1 : L'intersection de deux modèles de Herbrand est un modèle de Herbrand.

Preuve : directe (par l'absurde)

Remarque 1.4 : la proposition ci-dessus n'est pas vraie pour la logique du premier ordre en général [MM84]. En effet, si on considère une spécification dans la logique du premier ordre SP=(Σ, C) où l'ensemble C est , par exemple, la phrase "P(a) v P(b)" où a et b sont des constantes, P un symbole de prédicat et "v" symbole de la disjonction. Les deux interprétations de Herbrand IH1 = $(T(\Sigma), \alpha, \beta_1)$ et IH2 = $(T(\Sigma), \alpha, \beta_2)$, avec β_1 et β_2 définis par $\beta_1(P) = \{a\}$ et $\beta_2(P) = \{b\}$, sont deux modèles de SP. Mais IH3 =IH1 $\cap$ IH2 = $(T(\Sigma), \alpha, \beta_3)$ avec $\beta_3(P) = \emptyset$, n'est pas un modèle de la spécification SP.

Définition 1.19 : (modèle standard)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature et $SP = (\Sigma, C)$ une spécification. On appelle plus petit modèle de Herbrand de SP, l'intersection de tous les modèles de Herbrand de SP. Ce modèle est souvent appelé **modèle standard**. Nous le noterons $\mathbf{MS_{SP}}$.

Examinons maintenant le cas des spécifications dans la logique des clauses de Horn **avec égalité**. Dans les définitions précédentes des interprétations (de Herbrand) et de modèles (de Herbrand), tous les symboles de prédicats ont le même statut y compris les symboles d'égalité. Par contre dans toute E-interprétation et tout E-modèle, les prédicats d'égalité ont une interprétation unique. Si l'on considère la spécification SP de l'exemple 1.4, tout modèle de Herbrand $IH = (T(\Sigma), \alpha, \beta)$ de SP est tel que (s(0), s(s(s(0)))) appartienne à $\beta(==)$. Or s(0) est différent de s(s(s(0))) dans $T(\Sigma)$. On conclut donc qu'aucun modèle de Herbrand pour SP ne peut être un E-modèle pour SP.

Ceci nous amène alors à définir les E-interprétations de Herbrand pour une spécification SP donnée. Cela consiste à prendre comme ensembles supports les quotients des ensembles de termes fermés par une (famille de) relation de congruence $\equiv$. La relation $\equiv$ définie ci-dessous (voir aussi §1.2.2, proposition 1.6 pour une autre caractérisation), est la plus petite congruence engendrée par la spécification SP sur les termes fermés (proposition 1.2). Nous définissons la relation $\equiv$ comme l'interprétation des prédicats d'égalité dans le modèle standard de la spécification SP dont l'ensemble des clauses a été augmenté des axiomes de l'égalité (voir définition 1.20 ci-dessous). Ces axiomes d'égalité expriment les propriétés de congruence.

Définition 1.20 : (axiomes d'égalité AXE)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature avec égalité et $SP = (\Sigma, C)$ une spécification. On définit l'ensemble des axiomes de l'égalité pour la signature Σ, que nous noterons par $\mathbf{AXE}(\Sigma)$, comme étant l'ensemble des clauses suivantes (nous omettons les indices des symboles ==):

$x == x$ pour tout symbole d'égalité

$x == y \Leftarrow y == x$ pour tout symbole d'égalité

$x == z \Leftarrow x == y, y == z$ pour tout symbole d'égalité

$f(x_1,...,x_n) == f(y_1,...,y_n) \Leftarrow x_1 == y_1, ..., x_n == y_n$ pour tout opérateur f dans Ω_{us}

$P(x_1,...,x_n) \Leftarrow P(y_1,...,y_n), x_1 == y_1, ..., x_n == y_n$ pour tout prédicat P dans $\Pi_u - \{==_{ss} \mid s \in S\}$

Proposition 1.2 : Soient $\Sigma = (S, \Omega, \Pi)$ une signature avec égalité, $SP = (\Sigma, C)$ une spécification et $SP' = (\Sigma, C \cup AXE(\Sigma))$. Soit $MS_{SP'} = (T(\Sigma), \alpha, \beta)$ le plus petit modèle de Herbrand défini par SP'. Soit $\equiv$ la famille S-indicée : $\equiv = \{\equiv_s \mid s \in S\}$ telle que $\equiv_s$ est égale à $\beta (=_{ss})$. $\equiv$ est la plus petite congruence engendrée par la spécification SP sur la famille $T(\Sigma)$.

Preuve : (indication) $\equiv$ est une congruence sur les termes fermés vu les clauses qu'elle satisfait ; c'est la plus petite congruence de par sa définition en tant qu'interprétation des symboles d'égalité dans $MS_{SP'}$.

On note $T(\Sigma)_{\equiv}$ la famille $\{T_S(\Sigma)_{\equiv} \mid s \in S\}$. $T_S(\Sigma)_{\equiv}$ est l'ensemble quotient de $T_S(\Sigma)$ par la congruence $\equiv_S$. Par [t] nous désignons la classe d'équivalence du terme t. Nous définissons maintenant les E-interprétations de Herbrand.

Définition 1.21 : **(E-interprétation de Herbrand)**

Soit $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre avec égalité. On appelle **E-interprétation de Herbrand** une E-interprétation particulière $EIH = (T(\Sigma)_{\equiv}, \alpha, \beta)$ où

- $T(\Sigma)_{\equiv}$ est la famille définie plus haut
- La famille α est définie par :

$$\forall c \in \Omega_S, \alpha(c) = [c]$$

$$\forall f \in \Omega_{s_1 \ldots s_n s}, \forall [t_i] \in T_{S_i}(\Sigma)_{\equiv}, i = 1 \ldots n,$$

$$\alpha(f)([t_1],\ldots,[t_n]) = [f(t_1,\ldots,t_n)]$$

- La famille β vérifie:

$$\forall P \in \Pi_{s_1 \ldots s_n}, \beta(P) \subseteq T_{s_1}(\Sigma)_{\equiv} \times \ldots \times T_{s_n}(\Sigma)_{\equiv}$$

Notons que dans toute E-interprétation de Herbrand $EIH = (T(\Sigma)_{\equiv}, \alpha, \beta)$ on a $\beta(=_{ss}) = \{(x, x) \mid x \in T_S(\Sigma)_{\equiv}\}$. D'autre part, les E-interprétations de Herbrand d'une spécification donnée ne se différencient entre elles que par les interprétations des prédicats -autres que ==- (la famille β).

Définition 1.22 : **(E-modèle de Herbrand)**

Soient $\Sigma = (S, \Omega, \Pi)$ une signature avec égalité et $SP = (\Sigma, C)$ une spécification. Un **E-modèle de Herbrand** de SP est une E-interprétation de Herbrand qui est modèle de SP.

Exemple 1.6 : Nous donnons ci-dessous un exemple d'interprétation de Herbrand (le symbole d'égalité est considéré comme tout autre prédicat) et un exemple de E-interprétation de Herbrand (où l'égalité reflète l'identité sur les ensembles supports).

$SP = (\Sigma, C)$ avec $\Sigma = (S, \Omega, \Pi)$ telle que :

$S = \{nat\}$

$\Omega = \{\{0 : \rightarrow nat\}, \{s : nat \rightarrow nat\}, \{+ : nat\ nat \rightarrow nat\}\}$

$\Pi = \{\{Pair : nat\}, \{== : nat\ nat\}\}$

$C = \{$ $s(s(x)) == x$

$0 + x == x$

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

Pair (0)

Pair(s(s(x))) $\Leftarrow$ Pair (x) }

Soient ST1 = (A_1, α_1, β_1) et ST2 = (A_2, α_2, β_2) deux interprétations de Σ définies respectivement par:

$A_{1nat} = T(\Sigma) = \{0, s(0), 0+0, \ldots\}$;

α_1 est définie comme dans la définition 1.16;

$\beta_1(Pair) = \{x \in T(\Sigma) \mid (x,0) \in \beta_1(==)\}$

$\beta_1(==)$ = la plus petite congruence engendrée par C sur $T(\Sigma)$.

$= \{(0, 0), (0, ss(0)), (ss(0), 0), (0+0, 0), (s(0), s(0)), \ldots\}$

$A_{2nat} = \{[0], [1]\}$, avec [0] $=\{0, ss(0), 0+0, \ldots\}$, [1] $= \{s(0), sss(0), 0+ s(0), \ldots\}$;

α_2 est définie comme dans la définition 1.21;

$\beta_2(Pair) = \{[0]\}$

$\beta_2(==) = \{([0],[0]), ([1], [1])\}$

L'interprétation ST1 est un modèle de Herbrand pour SP mais non pas un E-modèle de Herbrand. ST2 est un exemple de E-modèle de Herbrand pour SP.

Comme pour le modèles de Herbrand, nous définissons l'intersection des E-modèles de Herbrand afin de définir le E-modèle standard.

Définition 1.23 : (intersection de E-modèles de Herbrand)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature avec égalité, SP = (Σ, C) une spécification et EIH = $(T(\Sigma)_{\equiv}, \alpha, \beta)$ et EIH' = $(T(\Sigma)_{\equiv}, \alpha, \beta')$ deux E-modèles de Herbrand de SP. On définit l'intersection de EIH et EIH' par EIH $\cap$ EIH' = $(T(\Sigma)_{\equiv}, \alpha, \beta'')$ où β'' est défini comme suit :

$\forall P \in \Pi_u$, $\beta''(P) = \beta(P) \cap \beta'(P)$

De cette définition on déduit le résultat suivant :

Proposition 1.3 : L'intersection de deux E-modèles de Herbrand est un E-modèle de Herbrand.

Preuve : la même que dans le cas sans égalité.

Nous sommes prêts maintenant pour donner une première définition du E-modèle standard d'une spécification donnée. Rappelons que dans tout ce mémoire la sémantique d'une spécification est le E-modèle standard.

Définition 1.24 : (E-modèle standard)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature avec égalité et SP=(Σ, C) une spécification. On appelle plus petit E-modèle de Herbrand de SP - ou E-modèle standard de SP- noté $\mathbf{E\text{-}MS_{SP}}$ l'intersection de tous les E-modèles de Herbrand de SP.

Avant de conclure ce paragraphe, nous définissons deux congruences sur les termes ($T(\Sigma,X)$), notées $=_{SP}$ et $=^{i}_{SP}$. Ces deux congruences sont utilisées dans les deux chapitres suivants respectivement dans les définitions de la E-unification et de la EI-unification. $=_{SP}$ (respectivement $=^{i}_{SP}$) désigne la théorie équationnelle (respectivement la théorie équationnelle inductive -d'où le "i" dans $=^{i}_{SP}$ -) définie par SP.

Définition 1.25 : (congruences $=_{SP}$ et $=^{i}_{SP}$)

Soient SP=(Σ, C) une spécification dans la logique des clauses de Horn avec égalité et X un ensemble de variables. On définit les deux relations sur $T(\Sigma,X)$ $=_{SP}$ et $=^{i}_{SP}$ par :

$$\forall\, s \in S, \forall\, t, t' \in T_s(\Sigma,X),\ t =_{SP} t' \Leftrightarrow \forall\, M \in \text{E-Mod}_{SP},\ M \models t == t'$$

$$\forall\, s \in S, \forall\, t, t' \in T_s(\Sigma,X),\ t =^{i}_{SP} t' \Leftrightarrow \text{E-MS}_{SP} \models t == t'$$

Proposition 1.4 : Soient SP=(Σ, C) une spécification dans la logique des clauses de Horn avec égalité. Les relations $=_{SP}$ et $=^{i}_{SP}$ sont des congruences sur $T(\Sigma,X)$. De plus, on a les inclusions suivantes : $\equiv \subseteq =_{SP} \subseteq =^{i}_{SP}$.

Preuve : directe.

1.2.2. Sémantique initiale

Une manière abstraite de caractériser le plus petit E-modèle de Herbrand d'une spécification SP fait appel à la propriété d'initialité d'un objet dans une catégorie (la catégories des E-modèles dans notre cas). Le but de cette approche est de faire ressortir le fait que la structure du E-modèle standard d'une spécification se reflète dans tous les autres E-modèles de celle-ci. Nous commençons ce paragraphe par la définition des Σ-homomorphismes entre Σ-structures, ensuite, après quelques définitions techniques supplémentaires, nous montrons que le E-modèle standard est initial.

Définition 1.26 : (Σ-homomorphismes)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature, $ST = (A, \alpha, \beta)$ et $ST' = (A', \alpha', \beta')$ deux Σ-structures. On appelle **Σ-homomorphisme** h entre les structures ST et ST' ($h : ST \rightarrow ST'$) toute famille S-indicée d'applications h_s: $A_s \rightarrow A'_s$ telle que :

- $\forall\, c \in \Omega_s,\ h_s(\alpha(c)) = \alpha'(c)$

- $\forall f \in \Omega_{s_1 \ldots s_n s}, \forall t_i \in A_{s_i}, i = 1 \ldots n$

$$h_s(\alpha(f)(t_1,\ldots t_n)) = \alpha'(f)\ (h_{s_1}(t_1),\ldots,h_{s_n}(t_n))$$

- $\forall P \in \Pi_{s_1 \ldots s_n}, \forall t_i \in A_{s_i}, i = 1 \ldots n$

$$(t_1,\ldots t_n) \in \beta(P) \Rightarrow (h_{s_1}(t_1),\ldots,h_{s_n}(t_n)) \in \beta'(P)$$

Définition 1.27 : (composition de Σ-homomorphismes)

Soient $\Sigma = (S, \Omega, \Pi)$ une signature, $ST_1 = (A_1, \alpha_1, \beta_1)$, $ST_2 = (A_2, \alpha_2, \beta_2)$ et $ST_3 = (A_3, \alpha_3, \beta_3)$ trois Σ-structures telles qu'il existe deux Σ-homomorphismes h_1 et h_2 , $h_1 : ST_1 \rightarrow ST_2$ et $h_2 : ST_2 \rightarrow ST_3$. On définit la composition des Σ-homomorphismes h_2 et h_1, notée $h_2 \circ h_1 : ST_1 \rightarrow ST_3$, comme étant la famille S-indicée $\{h_{2_s} \circ h1_s : A_{1_s} \rightarrow A_{3_s} \mid s \in S\}$. $h_2 \circ h_1$ définit évidemment un Σ-homomorphisme entre ST_1 et ST_3.

Définition 1.28 : (Σ-isomorphisme)

Un Σ-homomorphisme $h : ST \rightarrow ST'$ est un Σ-isomorphisme si toutes les applications h_s sont bijectives et pour tout P: $s1 \ldots s_n$, on a $\beta'(P) = \{(h_{s_1}(t_1),\ldots,h_{s_n}(t_n)) \mid (t_1,\ldots t_n) \in \beta(P)\}$. Une autre caractérisation d'un Σ-isomorphisme h est l'existence d'un autre Σ-homomorphisme $g : ST' \rightarrow ST$ tel que pour toute sorte s on ait : $h_s \circ g_s = id_{A'_s}$ et $g_s \circ h_s = id_{A_s}$. g est alors unique.

Nous introduisons maintenant la définition des catégories et celle d'objet initial. Pour une définition plus complète voir par exemple[AM75].

Définition 1.29 : (catégorie)

Une catégorie Cat est un triplet Cat = (Obj, Mor, o) où :

- Obj est une classe d'objets,
- Mor (O_1, O_2) est une classe de flèches (de morphismes) entre les objets O_1 et O_2.On notera $f : O_1 \rightarrow O_2$ un morphisme entre les objets O_1 et O_2.
- o est une loi de composition (partielle) sur les flèches. Soient $f \in$ Mor (O_1, O_2) et $g \in$ Mor(O_2, O_3) c.à.d. $f : O_1 \rightarrow O_2$ et $g : O_2 \rightarrow O_3$ alors $g \circ f : O_1 \rightarrow O_3$ est défini et appartient à Mor(O_1, O_3).

tels que :

- $\forall O \in$ Obj, $id_O \in$ Mor (O, O). id_O est appelé l'identité de O et est tel que : $\forall O_1 \in$ Obj, $\forall f \in$ Mor (O_1, O), $\forall g \in$ Mor (O, O_1) on a : $id_O \circ f = f$ et $g \circ id_O = g$.
- $\forall h \in$ Mor(O_1, O_2), $\forall g \in$ Mor(O_2, O_3), $\forall f \in$ Mor(O_3, O_4), on a : $f \circ (g \circ h) = (f \circ g) \circ h$

Définition 1.30 : (objet initial)

Soit Cat = (Obj, Mor, o) une catégorie. Un objet I dans Obj est **initial** dans Cat ssi, entre I et chaque objet O dans Obj, il existe un morphisme unique h : I $\rightarrow$ O.

Nous allons montrer maintenant que le E-modèle standard est initial dans la classe E-Mod$_{SP}$. On a d'abord le résultat suivant :

Proposition 1.5 : La classe E-Mod$_{SP}$ des E-modèles d'une spécification SP forme une catégorie dont les objets sont les E-modèles de SP et les morphismes sont les Σ-homomorphismes.

Preuve : directe

Pour montrer l'initialité de E-MS$_{SP}$, nous introduisons la notion d'évaluation de terme dans une structure ST donnée. Cette notion est un cas particulier de l'affectation où l'ensemble de variables pour l'affectation est vide. L'évaluation des termes d'une spécification nous sera utile pour la définition d'un homomorphisme unique entre E-MS$_{SP}$ et un E-modèle quelconque de SP.

Définition 1.31 : **(évaluation de terme)**

Soient $\Sigma = (S, \Omega, \Pi)$ une signature et ST = (A, α,β) une Σ-structure. On définit la fonction d'évaluation de sorte s, $e_s^{ST} : T_s(\Sigma) \rightarrow A_s$, comme suit :

- $\forall\, c \in \Omega_s$, $e_s^{ST}(c) = \alpha\,(c)$
- $\forall\, f \in \Omega_{s_1 \ldots s_n s}, \forall\, t_i \in T_{s_i}$,

$e_s^{ST}(f(t_1,\ldots t_n)) = \alpha(f)\ (e^{ST}{}_{s_1}(t_1),\ldots,e^{ST}{}_{s_n}(t_n))$

On notera par $e^{ST} : T(\Sigma) \rightarrow A$ la famille S-indicée de fonctions d'évaluation $\{e_s^{ST} \mid s \in S\}$.

La proposition qui suit donne une nouvelle caractérisation de la congruence $\equiv$ définie dans la proposition 1.2.

Proposition 1.6 : Soient $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre avec égalité et SP = (Σ, C) une spécification, on a :

$$\forall\, t, t' \in T_s(\Sigma),\ t \equiv_s t' \Leftrightarrow \forall\ ST = (A,\alpha,\beta) \in \text{E-Mod}_{SP},\ e_s^{ST}(t) = e_s^{ST}(t')$$

Preuve : ($\Leftarrow$) évident par définition de $\equiv_s$. ($\Rightarrow$) $t \equiv_s t'$ signifie que l'atome $t =_s t'$ est une formule valide dans E-Mod$_{SP}$. Puisque t et t' ne contiennent pas de variables, les affectations ($a^{\#}$) se réduisent aux évaluations (e^{ST}). On a alors $\forall$ ST = (A,α,β) $\in$ E-Mod$_{SP}$, $e_s^{ST}(t) = e_s^{ST}(t')$ par définition de la validité d'une formule.

Nous montrons dans le lemme suivant que dans tout E-modèle d'une spécification, se reflète la structure d'un E-modèle de Herbrand. Ce résultat nous servira dans la preuve du théorème1.1 où nous montrons l'initialité de E-MS$_{SP}$ dans la catégorie E-Mod$_{SP}$.

Lemme 1.1 : Soient $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre avec égalité et SP $= (\Sigma, C)$ une spécification. Pour tout E-modèle $M= (A, \alpha^M, \beta^M)$ de SP, il existe au moins un E-modèle de Herbrand H tel qu'il existe un homomorphisme h entre H et M, $h : H \rightarrow M$.

Preuve : soit $H = (T(\Sigma)_{\equiv}, \alpha^H, \beta^H)$ la E-interprétation de Herbrand où β^H est défini comme suit (α^H étant commune à toutes les E-interprétations de Herbrand) :

$\forall P \in \Pi_{s_1...s_n}$, $([t_1],..., [t_n]) \in \beta^H(P) \Leftrightarrow ({e_{s_1}}^M(t_1),...{e_{s_n}}^M(t_n)) \in \beta^M(P)$

Soit $h : H \rightarrow M$ la famille S-indicée d'applications $h_S: T_S(\Sigma)_{\equiv} \rightarrow A_S$ telles que $h_S([t]) = e_S^M(t)$. h est clairement un homomorphisme. Il reste à prouver que la E-interprétation de Herbrand H ainsi définie est un E-modèle de SP. Faisons d'abord la remarque suivante : Soit X un ensemble de variables, pour toute affectation $aff : X \rightarrow T(\Sigma)_{\equiv}$ et T un terme dans $T(\Sigma, X)$ on a : $h(aff^{\#}(T)) = a^{\#}(T)$ où a est l'affectation $a : X \rightarrow A$ définie par $a(x) = h(aff(x))$, h étant l'homomorphisme défini ci-dessus. Pour montrer que H est un E-modèle, nous montrons que toutes les clauses de la spécification sont satisfaites. Par définition, les clauses d'une spécification sont de deux formes:

(i) $P(T_1,...T_n)$

avec $T_i \in T_{s_i}(\Sigma, X)$ et $P \in \Pi_{s_1...s_n}$. Dans ce cas, par définition de M (M est un E-modèle de SP) on a : $\beta^M(P) \supseteq \{(a^{\#}{}_{s_1}(T_1),\ldots,a^{\#}{}_{s_n}(T_n)) \mid a : X \rightarrow A$ est une affectation$\}$. Par définition de H et compte tenu de la remarque ci-dessus on en déduit que $\beta^H(P) \supseteq \{([aff^{\#}{}_{s_1}(T_1)],\ldots,[aff^{\#}{}_{s_n}(T_n)]) \mid aff : X \rightarrow T(\Sigma)_{\equiv}$ est une affectation$\}$. Ceci prouve que de telles clauses sont satisfaites dans H. Examinons le deuxième cas de clauses

(ii) $P(T_1,...T_n) \Leftarrow Q_1(\ldots),\ldots, Q_i(\tau^i{}_1,...,\tau^i{}_{q_i}),\ldots, Q_p(\ldots)$

Soit $aff : X \rightarrow T(\Sigma)_{\equiv}$ une affectation telle que pour tout i=1...p, $([aff^{\#}{}_{s_1}(\tau^i{}_1)],\ldots,[aff^{\#}{}_{s_{q_i}}(\tau^i{}_{q_i})])$ appartient à $\beta^H(Q_i)$. Soit $a : X \rightarrow A$ l'affectation telle que $h \circ aff^{\#} = a^{\#}$ (avec h défini plus haut).Puisque, par hypothèse, M est un E-modèle de SP on a $(a^{\#}{}_{s_1}(T_1),\ldots,a^{\#}{}_{s_n}(T_n)) \in \beta^M(P)$ car , par définition de H et de l'affectation a, pour tout i= 1...p, $(a^{\#}(\tau^i{}_1),\ldots, a^{\#}(\tau^i{}_{q_i})) \in \beta^M(Q_i)$. On déduit donc par définition de H que le n-uplet $([aff^{\#}{}_{s_1}(T_1)],\ldots,[aff^{\#}{}_{s_n}(T_n)]) \in \beta^H(P)$. H satisfait donc toute clause de la deuxième forme. H est donc un E-modèle de Herbrand de SP.cqfd.

Théorème 1.1 : Soient $\Sigma = (S, \Omega, \Pi)$ une signature du premier ordre avec égalité et SP $= (\Sigma, C)$ une spécification. Le modèle standard E-MS$_{SP}$ est initial dans E-Mod$_{SP}$.

Preuve :

On montre que pour tout E-modèle M dans E-Mod$_{SP}$, il existe un homomorphisme unique entre E-MS$_{SP}$ et M. L'existence est immédiate en utilisant le lemme ci-dessus et

sachant que d'une part, par définition de E-MS_{SP}, il existe un homomorphisme entre E-MS_{SP} et tout autre E-modèle de Herbrand de SP. Et d'autre part la composition de deux homomorphismes est aussi un homomorphisme. Quant à l'unicité on peut la prouver par induction sur la hauteur des termes (hauteur minimale des termes dans [t]). En effet pour le cas des constantes quel que soit l'homomorphisme g : E-$MS_{SP} \rightarrow$ M, et c une constante, $g([c]) = \alpha^M(c) = h([c])$ (h est le même que dans la preuve du lemme 1.1). Supposons que g coïncide avec h sur les termes de hauteur inférieure ou égale à n, soit t un terme de hauteur égale à n+1, t est de la forme $f(t_1,...,t_n)$. $g([f(t_1,...,t_n)]) = [\alpha^M(f)\ (g(t_1),...,g(t_n))] = [\alpha^M(f)\ (g([t_1]),...,g([t_n]))] = [\alpha^M(f)\ (h([t_1]),...,h([t_n]))] = h([f(t_1,...,t_n)])$ (par hypothèse d'induction).cqfd.

Remarque 1.5 : Tout modèle initial dans E-Mod_{SP} est isomorphe à E-MS_{SP}. En effet, les objets initiaux dans une catégorie, s'ils existent, sont isomorphes entre eux.

1.2.3. Plus petit point fixe

Une autre façon de caractériser le E-modèle standard d'une spécification repose sur le calcul de point fixe (le plus petit dans notre cas) d'une transformation sur les E-interprétations de Herbrand. La différence majeure entre cette approche et les deux précédentes réside dans le calcul de point fixe, qui caractérise le modèle standard de manière constructive. Il correspond, en effet, aux déductions par une stratégie en chaînage-avant (voir par exemple [Kow79] pour plus de détails sur les stratégies). Nous rappelons dans un premier temps les principaux résultats qui nous intéressent sur les points fixes. Ensuite, comme application de ces résultats, nous montrons que le E-modèle standard est le plus petit point fixe d'une transformation que nous définissons.

1.2.3.1. Préliminaires

Dans ces préliminaires, nous rappelons les définitions classiques d'ordre, de treillis, d'application continue et de point fixe. Nous rappelons enfin un résultat qui donne une définition constructive du plus petit point fixe (proposition 1.9). (voir [LNS82] pour un historique de ces résultats et par exemple [Sto77] pour plus de détails).

Définition 1.32 : (ordre partiel)

Soient E un ensemble et $\leq$ une relation binaire sur E. La relation $\leq$ définit un ordre partiel sur E ssi :

- $\forall x \in E, x \leq x$ réflexivité
- $\forall x, y \in E, x \leq y$ et $y \leq x \Rightarrow x = y$ antisymétrie

- $\forall x, y, z \in E, x \leq y$ et $y \leq z \Rightarrow x \leq z$ transitivité

Définition 1.33 : (majorant, minorant)

Soit $\leq$ un ordre partiel défini sur un ensemble E.

- a est un minorant d'un ensemble $X \subseteq E$ ssi $\forall x \in X, a \leq x$
- a est un majorant d'un ensemble $X \subseteq E$ ssi $\forall x \in X, x \leq a$

Définition 1.34 : (borne supérieure, borne inférieure)

Soit $\leq$ un ordre partiel défini sur un ensemble E.

- a est la borne supérieure d'un ensemble $X \subseteq E$, notée bsup (X), ssi
  - a est un majorant de X
  - $\forall$ a' majorant de X, $a \leq a'$
- a est la borne inférieure d'un ensemble $X \subseteq E$, notée binf(X), ssi
  - a est un minorant de X
  - $\forall$ a' minorant de X, $a' \leq a$

Remarque 1.6 : la borne supérieure est unique quand elle existe, de même que la borne inférieure

Définition 1.35 : (treillis complet)

Soit $(E, \leq)$ un ensemble E muni d'un ordre partiel $\leq$. $(E, \leq)$ est un treillis complet si pour chaque sous-ensemble X de E les bornes inférieure (binf(X)) et supérieure (bsup(X)) existent.

Définition 1.36 : (application monotone)

Soient $(E, \leq)$ un treillis complet et $T : E \rightarrow E$ une application. T est monotone ssi $\forall x, y \in E, x \leq y \Rightarrow T(x) \leq T(y)$.

Définition 1.37 : (ensemble dirigé)

Soit $(E, \leq)$ un treillis complet. Un sous ensemble X de E est dirigé ssi $\forall Y \subseteq X$ et Y fini, $bsup(Y) \in X$.

Définition 1.38 : (application continue)

Soient $(E, \leq)$ un treillis complet et $T : E \rightarrow E$ une application. T est continue ssi $\forall X \subseteq E$ et X dirigé, $T(bsup(X)) = bsup(T(X))$

Proposition 1.7 : Soient $(E, \leq)$ un treillis complet et $T : E \rightarrow E$ une application. Si T est continue, alors T est monotone.

Définition 1.39 : (point fixe, plus petit point fixe)

Soient $(E, \leq)$ un treillis complet et $T : E \rightarrow E$ une application. Un élément e dans X est un point fixe de T si $T(e) = e$. Un élément p, point fixe de T, est le plus petit point fixe de T, noté pppf(T), si pour tout x point fixe de T, $p \leq x$.

Proposition 1.8 : Soient $(E, \leq)$ un treillis complet et $T : E \rightarrow E$ une application monotone. pppf(T) existe et $pppf(T) = binf(\{x \mid T(x) = x\}) = binf(\{x \mid T(x) \leq x\})$.

La définition suivante introduit une notation qui sert à calculer le plus petit point fixe de manière itérative. Ces notations et résultats sont dûs à Kleene et Stoy.

Définition 1.40 : ($T\uparrow n$)

Soient $(E, \leq)$ un treillis complet et $T : E \rightarrow E$ une application monotone. on note :

- $T\uparrow 0 = binf(E)$
- $T\uparrow n = T(T\uparrow(n - 1))$ si n est un ordinal successeur
- $T\uparrow n = binf(\{T\uparrow m \mid m < n\})$ si n est un ordinal limite

Proposition 1.9 : Soient $(E, \leq)$ un treillis complet et $T : E \rightarrow E$ une application continue. Alors $pppf(T) = T\uparrow\omega$.

1.2.3.2. Application aux spécifications

Nous montrons dans cette section que le E-modèle standard est le plus petit point fixe d'une transformation. Nous appliquons les résultats précédents. Ainsi, nous définissons un treillis complet des E-interprétations de Herbrand $(EIH_{SP}, \leq)$ et une transformation continue sur EIH_{SP}.

Nous commençons par préciser au moyen des deux définitions suivantes l'ensemble sur lequel est définie la structure de treillis ainsi que l'ordre utilisé.

Définition 1.41 : (EIH_{SP})

Soit SP une spécification. On désigne par EIH_{SP} l'ensemble des E-interprétations de Herbrand pour la spécification SP.

Définition 1.42 : (ordre partiel sur les E-interprétations de Herbrand)

Soit SP une spécification. Soient $EIH_1 = (T(\Sigma)_{\equiv}, \alpha, \beta_1)$ et $EIH_2 = (T(\Sigma)_{\equiv}, \alpha, \beta_2)$ deux E-interprétations de Herbrand de SP. On définit la relation $\leq$ sur les E-interprétations de

Herbrand comme suit : $EIH_1 \leq EIH_2$ ssi il existe un homomorphisme $h : EIH_1 \rightarrow EIH_2$. $\leq$ est clairement une relation d'ordre partiel sur les E-interprétations de Herbrand.

Proposition 1.10 : $(EIH_{SP}, \leq)$ est un treillis complet.

La preuve de cette proposition est immédiate ; c'est une simple extension du cas du treillis complet $(2^A, \subseteq)$ où 2^A désigne l'ensemble des parties d'un ensemble A.

Nous définissons maintenant une application ET sur EIH_{SP} et montrons qu'elle est continue (proposition 1.11).

Définition 1.43 : (**transformation ET : $EIH_{SP} \rightarrow EIH_{SP}$**)

Soit SP une spécification. On définit la transformation sur les E-interprétations de Herbrand comme étant l'application ET : $EIH_{SP} \rightarrow EIH_{SP}$ telle que pour toute E-interprétation de Herbrand $EIH = (T(\Sigma)_{\equiv}, \alpha, \beta)$, $ET(EIH) = (T(\Sigma)_{\equiv}, \alpha, \beta')$ avec β' telle que:

$\forall P \in \Pi_{s_1 \ldots s_n}$, $\beta'(P) = \{([t_1],\ldots,[t_n]) \mid$ il existe une instance fermée d'une clause dans SP: $P(v_1,\ldots,v_n) \Leftarrow Q_1(t^1_1,\ldots, t^1_{n_1}), \ldots, Q_q(t^q_1,\ldots, t^q_{n_q})$ avec $t_i \equiv v_i$ pour $i = 1\ldots n$ et $([t^j_1],\ldots, [t^j_{n_j}]) \in \beta(Q_j)$ pour $j = 1\ldots q\}$

Remarque 1.7 : le cas où l'indice q est égal à 0 dans la définition ci-dessus correspond aux clauses exprimant des "faits". Dans ce cas la condition " $([t^j_1],\ldots, [t^j_{n_j}]) \in \beta(Q_j)$ pour $j = 1\ldots q$" dans la définition de β' est trivialement vraie.

Proposition 1.11 : ET est continue.

Preuve : Soit X un ensemble dirigé inclus dans EIH_{SP}. Montrons que ET(bsup(X)) = bsup(ET(X)). Soient $ET(bsup(X)) = (T(\Sigma)_{\equiv}, \alpha, \beta_1)$, $bsup(X) = (T(\Sigma)_{\equiv}, \alpha, \beta_2)$ et $bsup(ET(X)) = (T(\Sigma)_{\equiv}, \alpha, \beta_5)$. Soit P un élément de $\Pi_{s_1 \ldots s_n}$, on a :

$([t_1],\ldots,[t_n]) \in \beta_1(P) \Leftrightarrow$

(par définition de la transformation ET) il existe une instance fermée d'une clause de SP: $P(v_1,\ldots,v_n) \Leftarrow Q_1(t^1_1,\ldots, t^1_{n_1}), \ldots, Q_q(t^q_1,\ldots, t^q_{n_q})$ telle que $t_i \equiv v_i$ pour $i = 1\ldots n$, et $([t^j_1],\ldots, [t^j_{n_j}]) \in \beta_2(Q_j)$ pour $j = 1\ldots q$. $\Leftrightarrow$

(par définitions de β_2 (bsup(X)) et de X (dirigé)) il existe une interprétation de Herbrand $EIH_3 = (T(\Sigma)_{\equiv}, \alpha, \beta_3)$ dans X telle que $([t^j_1],\ldots, [t^j_{n_j}]) \in \beta_3(Q_j)$ pour $j = 1\ldots q$. $\Leftrightarrow$

(par définition de la transformation ET) il existe $EIH_4 = (T(\Sigma)_{\equiv}, \alpha, \beta_4)$ dans ET(X) tel que $([t_1],\ldots,[t_n]) \in \beta_4(P)$ avec $ET(EIH_3) = (T(\Sigma)_{\equiv}, \alpha, \beta_4) \Leftrightarrow$

(par définition de bsup(ET(X))) $([t_1],\ldots,[t_n]) \in \beta_5(P)$. cqfd.

En application des résultats du paragraphe §1.2.3.1, des propositions 1.10 et 1.11 nous pouvons énoncer le

Théorème 1.2 : Soient SP une spécification. Le plus petit point fixe pour l'application ET existe et $pppf(ET) = ET\uparrow\omega$. De plus, on a $E\text{-}MS_{SP} = ET\uparrow\omega$.

Preuve : $pppf(ET) = ET\uparrow\omega$ est une conséquence directe des résultats rappelés plus haut. Montrons que $E\text{-}MS_{SP}$ est le plus petit point fixe de ET. On a $ET(E\text{-}MS_{SP}) \leq E\text{-}MS_{SP}$ puisque $E\text{-}MS_{SP}$ est un E-modèle de SP. D'autre part $E\text{-}MS_{SP} \leq ET(E\text{-}MS_{SP})$ puisque $ET(E\text{-}MS_{SP})$ est un E-modèle et $E\text{-}MS_{SP}$ est initial dans $E\text{-}Mod_{SP}$. D'où $E\text{-}MS_{SP}$ est un point fixe pour la transformation ET. Tout point fixe pour ET étant un E-modèle de SP, $E\text{-}MS_{SP}$ étant initial dans $E\text{-}Mod_{SP}$, on en déduit que $E\text{-}MS_{SP}$ est le plus petit point fixe pour ET.

1.3. CONCLUSION

Nous avons essentiellement défini dans ce chapitre la notion de spécification dans la logique des clauses de Horn avec égalité. Nous avons défini la présentation syntaxique d'une spécification quelconque $SP = (\Sigma, C)$ (définition 1.7). De même, nous avons précisé la sémantique d'une spécification donnée SP comme le $E\text{-}MS_{SP}$: le E-modèle standard. Nous avons donné par ailleurs trois caractérisations différentes du E-modèle standard (définition 1.19, théorème 1.1 et théorème 1.2).

Chapitre 2

SLDEI-Résolution

Nous nous intéressons dans ce chapitre à l'aspect déduction ou calcul dans la logique des clauses de Horn avec égalité. Ce sujet trouve ses racines dans le domaine de la démonstration automatique, voir par exemple [Pla90] [CL73]. Les résultats de ce chapitre s'inspirent fortement de ces travaux. Nous garderons cependant un regard d'informaticien. C'est-à-dire que nous considérons toute spécification en tant que "programme", ceci se traduit par le choix du E-modèle standard comme sémantique au lieu de la classe E-Mod_{SP}, et notre intérêt portera alors sur le calcul des "solutions" dans le E-modèle standard pour les questions possibles. Les formes des questions et des réponses seront précisées. Par contre, nous ne nous intéresserons pas, par exemple, aux propriétés des clauses telles que la validité, l'insatisfaisabilité ou la satisfaisabilité dans une classe quelconque de modèles.

Après un paragraphe préliminaire dans lequel nous présentons informellement ce que nous entendons par *calcul*, nous rappelons dans le paragraphe §2.2 la règle d'inférence SLD-résolution et illustrons son application dans le cadre de la logique des clauses de Horn avec égalité, puis nous proposons dans un nouveau paragraphe §2.3, une nouvelle règle d'inférence qui prend en compte la sémantique initiale des spécifications. Cette règle est dite SLDEI-résolution. A chaque fois, la cohérence et la complétude de ces deux calculs seront discutées.

2.1. PRELIMINAIRES

Nous avons défini dans le premier chapitre ce que nous entendons par spécification dans la logique des clauses de Horn avec égalité ; à la fois par la syntaxe et la sémantique. Ainsi, une spécification SP = (Σ, C) dénote le E-modèle standard E-MS_{SP}. Dans ce chapitre nous nous intéressons à la résolution dans le E-modèle standard, de certaines formules que nous appelons "but".

Définition 2.1 : (but)

Soient SP= (Σ, C) une spécification. Nous appelons but toute conjonction d'atomes $A_1 \ldots A_n$.

Remarques 2.1 :

(i) Notons la distinction entre un but et une clause de but (définition 1.6). Un but est une conjonction d'atomes, soit "$A_1 \ldots A_n$". La "clause de but" correspondante est la négation du but, notée "$\Leftarrow A_1 \ldots A_n$".

(ii) Etant donné une spécification SP=(Σ, C), les symboles des opérations et des prédicats figurant dans les buts que nous serons amené à résoudre appartiennent à la signature Σ.

Etant donné une spécification SP=(Σ, C) et un but à résoudre, les calculs que nous présentons ont pour objectif la recherche des cas où le but considéré est valide dans E-MS_{SP}. Autrement dit, ce que nous attendons d'un calcul est la déduction d'un ensemble d'instances du but considéré tel que chacune de ces instances soit une formule valide dans le E-modèle standard E-MS_{SP}. Considérons un exemple.

Soit SP = (Σ, C), avec Σ = (S, Ω, Π), la spécification suivante :

S={nat}

Ω= {{0 : $\rightarrow$ nat},{s : nat $\rightarrow$ nat}, {+ : nat, nat $\rightarrow$ nat}}

Π= {{P: nat, nat, nat},{Q: nat}, {== : nat, nat}}

C ={ 1: 0 + x == x
2: s(x) + y == s(x + y)
3: P(x, y, z) $\Leftarrow$ x + y == z
4: Q(x) $\Leftarrow$ P(y, y, x) }

La sémantique de cette spécification est donnée par E-MS_{SP} = ($T(\Sigma)_{\equiv}$, α, β) défini ci-après. Informellement , les opérations 0 (zéro), s (successeur) et + (addition) désignent les opérations classiques sur les entiers naturels. Les prédicats P et Q représentent respectivement le graphe de l'addition et le sous-ensemble des entiers naturels pairs. Ainsi on a :

$T_{nat}(\Sigma)_{\equiv}$ = {[$s^n(0)$] | n $\in$ $\mathbb{N}$ } où $s^0(0) = 0$

β(==) = {([x], [x]) | x $\in$ $T_{nat}(\Sigma)$ }

β(P) = {([x], [y], [x + y]) | x, y $\in$ $T_{nat}(\Sigma)$ }

β (Q) = {[$s^{2p}(0)$] | p $\in$ $\mathbb{N}$ }

Considérons chacun des buts suivants qui nous permettent d'expliciter ce que nous entendons par calcul :

(i) s(s(0)) == s(0) + s(0)

(ii) P(s(0), s(0), s(s(0)))

(iii) s(s(0)) == s(0) + s(0), P(s(0), s(0), s(s(0)))

(iv) $P(x, y, s(s(0)))$

(v) $Q(s(0))$

(vi) $P(x, y, s(s(0))), Q(x)$

(vii) $Q(x + x)$

Les buts (i) et (ii) sont constitués d'un seul atome fermé. Ces deux buts sont vrais dans E-MS_{SP}. Dans ces deux cas, comme dans le cas (iii) qui est la conjonction de (i) et (ii), tout calcul doit se contenter d'indiquer la validité des buts. Le but (iv) n'est pas valide dans E-MS_{SP}, mais ce but (iv) est satisfaisable dans E-MS_{SP}. En effet, si les variables x et y prennent respectivement les valeurs 0 et s(s(0)), on obtient l'atome P(0, s(s(0)), s(s(0))) qui est vrai dans E-MS_{SP}. De même, on obtient des atomes vrais dans le E-modèle standard si les variables x et y sont respectivement affectées par s(0) et s(0), P(s(0), s(0), s(s(0))), ou bien respectivement affectées par s(s(0)) et 0, P(s(s(0)), 0, s(s(0))). Ces instances de variables nous amènent à introduire la notion de substitution, puis celle de solution pour un but.

Définition 2.2 : (substitution)

Soient Σ une signature et X un ensemble de variables. Une **substitution** σ est un endomorphisme de $T(\Sigma, X)$. On note **SUB** l'ensemble des substitutions. L'ensemble $\mathbf{D}(\sigma) = \{x \in X \mid \sigma(x) \neq x\}$ est appelé **domaine** de la substitution σ et $\mathbf{I}(\sigma) = \cup_{x \in D(\sigma)}$ vars $(\sigma(x))$ est l'ensemble des **variables introduites** par σ. La composition des substitutions est notée $\sigma\sigma'$ avec $\sigma\sigma'(t) = \sigma(\sigma'(t))$. Dans la suite, on confondra une substitution σ avec l'application $\sigma'': X \rightarrow T(\Sigma, X)$ dont σ est l'unique extension ($T(\Sigma, X)$ étant libre sur X). Une substitution est **fermée** si $I(\sigma)$ est vide. On note **SUBF** l'ensemble des substitutions fermées. Soient V un ensemble de variables et σ une substitution, on note par $\sigma|_V$ la restriction de la substitution σ aux variables de V, définie par : $\sigma|_V(x) = \sigma(x)$ si $x \in V$, $\sigma|_V(x) = x$ si $x \notin V$. Nous supposons dans la suite que toutes les substitutions sont idempotentes, c'est-à-dire $\sigma\sigma = \sigma$ pour toute substitution σ.

Soient σ une substitution et $D(\sigma)=\{x_i , i =1...m\}$ son domaine tel que $\sigma(x_i)=t_i$ pour $i=1...m$. On notera une telle substitution par $\sigma=\{\mathbf{x_1 \rightarrow t_1, ..., x_m \rightarrow t_m}\}$.

Nous définissons maintenant deux préordres sur les substitutions. Ces préordres sont utilisés pour définir la complétude des calculs.

Définition 2.3 : (préordres (SUB, $\leq_{SP}$) et (SUB, $\leq^{i}_{SP}$))

Soit V un ensemble de variables. On définit deux préordres sur SUB, $\leq_{SP}$ et $\leq^{i}_{SP}$, en fonction de deux relations de congruence sur l'ensemble des substitutions, $\equiv_{SP}$ et $\equiv^{i}_{SP}$ définis comme suit :

$\forall \sigma, \theta \in SUB, \sigma \equiv_{SP} \theta\ [V] \Leftrightarrow \forall x \in V, \sigma(x) =_{SP} \theta(x)$

$\forall \sigma, \theta \in SUB, \sigma \equiv^{i}_{SP} \theta\ [V] \Leftrightarrow \forall x \in V, \sigma(x) =^{i}_{SP} \theta(x)$

$\forall \sigma, \theta \in SUB, \sigma \leq_{SP} \theta\ [V] \Leftrightarrow \exists \delta \in SUB \mid \delta\sigma \equiv_{SP} \theta\ [V]$

$\forall \sigma, \theta \in SUB, \sigma \leq^{i}_{SP} \theta\ [V] \Leftrightarrow \exists \delta \in SUB \mid \delta\sigma \equiv^{i}_{SP} \theta\ [V]$

Intuitivement, $\sigma \leq_{SP} \beta$ (respectivement $\sigma \leq^{i}_{SP} \beta$) veut dire que la substitution σ est "moins instanciée" modulo la congruence $=_{SP}$ (respectivement la congruence $=^{i}_{SP}$) que la substitution β. On vérifie facilement que $\leq_{SP}$ et $\leq^{i}_{SP}$ sont des préordres (réflexivité + transitivité). Lorsque $=_{SP}$ désigne l'identité syntaxique, nous utiliserons les symboles = et $\leq$ au lieu de $=_{SP}$ et $\leq_{SP}$ respectivement.

Définition 2.4 : (substitution cohérente, solution)

Soient SP= (Σ, C) une spécification et D = $A_1 \ldots A_n$ un but. Une substitution σ est une substitution **cohérente** pour D (ou une **solution**) pour D ssi $\sigma(A_i)$ est valide dans le E-modèle standard E-MS_{SP}, pour i =1...n. Autrement dit : $\forall$ i =1...n, E-$MS_{SP} \models \sigma(A_i)$. On note **Sol(SP, D)** l'ensemble de toutes les solutions possibles pour le but D.

Comme exemple de solution, la substitution identité, notée id, avec D(id) = $\emptyset$, est une solution pour les buts (i), (ii) et (iii) (voir page 2-2). Les substitutions σ_1, σ_2 et σ_3 définies par :

$\sigma_1 = \{x \rightarrow 0, y \rightarrow s(s(0))\}$,

$\sigma_2 = \{x \rightarrow s(0), y \rightarrow s(0)\}$ et

$\sigma_3 = \{x \rightarrow s(s(0)), y \rightarrow 0\}$

sont des solutions pour le but (iv). Par contre la substitution fermée α qui affecte 0 à x et à y, soit $\alpha = \{x \rightarrow 0, y \rightarrow 0\}$, n'est pas une solution pour le but (iv) puisque l'atome α(P(x, y, s(s(0)))), soit P(0, 0, s(s(0))), n'est pas vrai dans le E-modèle standard.

Le but (v) est un exemple de but insatisfaisable. Il est immédiat qu'il n'existe aucun élément y dans $T_{nat}(\Sigma)$ tel que y + y == s(0) soit vrai. Le but (v) n'a donc pas de solutions.

Considérons maintenant le but (vi) . Nous avons vu que l'atome P(x, y, s(s(0))) a trois solutions : $\sigma_1 = \{x \rightarrow 0, y \rightarrow s(s(0))\}$, $\sigma_2 = \{x \rightarrow s(0), y \rightarrow s(0)\}$ et $\sigma_3 = \{x \rightarrow s(s(0)), y \rightarrow 0\}$. Puisque les atomes Q(0) et Q(s(s(0))) sont vrais dans E-MS_{SP} et Q(s(0)) est faux dans E-MS_{SP}, nous déduisons l'ensemble de solutions pour le but (vi) comme étant l'ensemble $\{\sigma_1, \sigma_3\}$.

Considérons le but (vii). Ce but admet une infinité de solutions, en particulier l'identité id car Q(x+x) est valide dans E-MS_{SP}. Soient les ensembles de substitutions suivants:

$S_1 = \{id\}$

$S_2 = \{\{x \rightarrow 0\}, \ldots, \{x \rightarrow s^n(0)\}, \ldots\}$ avec $n \in \mathbb{N}$

$S_3 = \{\{x \rightarrow 0\}, \{x \rightarrow s(y)\}\}$

$S_4 = \{\{x \rightarrow s(y)\}\}$

Chacun de ces ensembles est composé de solutions pour le but (vi). Dans le cas général, il peut exister une infinité de solutions lorsqu'un but est satisfaisable. Les calculs que nous considérons dans la suite engendrent ce que l'on appelle des ensembles complets de solutions. Intuitivement, un ensemble complet de solutions pour un but donné est un ensemble à partir duquel on peut déduire toutes les solutions fermées pour ce but.

Définition 2.5 : (ensemble complet de solutions)

Soient SP= (Σ, C) une spécification, B = $A_1 \ldots A_n$ un but et W un ensemble de variables contenant V=vars(B). Soit S un ensemble de substitutions. On dira que S est un ensemble complet de solutions pour B en dehors de W ssi :

(i) $\forall\, \sigma \in S, \sigma \in Sol(SP, B)$. (cohérence)

(ii) $\forall \sigma \in SUBF \cap Sol(SP, B)$, et $D(\sigma) = V, \exists\, \theta \in S \mid \theta \dot{\leq}_{SP} \sigma$ [V]. (complétude)

(iii) $\forall\, \sigma \in S, D(\sigma) \subseteq V$ et $I(\sigma) \cap W = \emptyset$.

Les conditions (i) et (ii) ci-dessus expriment respectivement la cohérence et la complétude de l'ensemble S. La troisième condition (iii) est d'ordre technique. Elle préserve des manipulations incorrectes des variables. L'ensemble W dénote les variables utilisées dans un contexte global.

Les ensembles S_1, S_2 et S_3, définis précédemment sont des ensembles complets de solutions pour le but (vii). L'ensemble S_4 n'est pas un ensemble complet de solutions pour le but (vii) puisque la substitution $\{x \rightarrow 0\}$ ne peut pas être engendrée par une substitution dans S_4.

Remarque 2.2 : La notion de cohérence des substitutions, dans le domaine de la démonstration automatique, est définie par rapport à tous les modèles d'une spécification ($E\text{-}Mod_{SP} \models \sigma(t)$). Avec cette définition de cohérence, le but x+y == y+x, où + est l'opération d'addition sur les naturels, qui est valide seulement dans les modèles inductifs, n'admettrait pas la substitution identité (id) comme solution. La définition de cohérence utilisée dans ce mémoire a été motivée par le fait que nous soyons intéressé par un langage de "spécification" dont la sémantique des spécifications est précisément le E-modèle standard (cf. chapitre 5). De même que pour la cohérence, nous définissons la complétude par rapport aux solutions dans le E-modèle standard, et non par rapport à tous les E-modèles d'une spécification. Ceci se traduit par l'utilisation du préordre $\dot{\leq}_{SP}$ au lieu de $\leq_{SP}$. Evidemment, nous nous sommes contenté de considérer la complétude vis-à-vis des solutions fermées, la complétude par rapport aux solutions non fermées dans le E-modèle standard étant exclue dans le cas général, voir les résultats d'incomplétude de Gödel [NNGG89].

Nous étudions maintenant comment aboutir à des ensembles complets de solutions. D'où la notion de calcul que nous présentons informellement ci-dessous.

Définition 2.6 : (calcul, ensemble de substitutions calculé)

Nous définissons un calcul χ comme étant un ensemble de règles de déduction de solutions pour les buts. Soient une spécification SP et un but B, nous désignons par $\mathbf{Sol_\chi(SP, B)}$ l'ensemble de substitutions déduites par le calcul χ pour le but B en utilisant les axiomes de la spécification SP. On dira qu'un calcul χ est cohérent si pour tout but B, toutes les substitutions dans $Sol_\chi(SP, B)$ sont solutions du but à résoudre. Le calcul χ est complet si pour tout but B, $Sol_\chi(SP, B)$ est un ensemble complet de solutions pour B.

Les calculs que nous présentons dans la suite procèdent par réfutation. La réfutation, déduction de la clause vide à partir d'un ensemble d'axiomes A, indique l'insatisfaisabilité de l'ensemble d'axiomes A (non existence de (E-)modèle pour A). Cette méthode de preuve de l'insatisfaisabilité d'un ensemble d'axiomes découle des travaux sur la démonstration automatique, en particulier ceux de Herbrand. Ce point de vue sur la déduction est quelque peu artificiel dans notre cas puisque notre intérêt porte sur la déduction d'ensembles complets de solutions pour les buts, et non sur la preuve de l'inconsistance d'un ensemble de formules. Nous ne rappelons donc pas le théorème de Herbrand qui justifie les preuves par réfutation (voir par exemple [Gal86]). Tous les résultats seront prouvés directement.

Avant de clore ce paragraphe préliminaire, nous rappelons deux définitions classiques d'"unification" et de "variante" d'entité syntaxiques, utiles pour la suite du mémoire.

Définition 2.7 : (unification)

Soient t_1 et t_2 deux termes. On dit que t_1 et t_2 sont unifiables s'il existe une substitution σ telle que $\sigma(t_1) = \sigma(t_2)$ (ici le signe = désigne l'égalité syntaxique). σ est appelé unificateur de t_1 et t_2. On notera $U(t_1,t_2)$ l'ensemble des unificateurs de t_1 et t_2. Si $U(t_1,t_2)$ n'est pas vide, il contient toujours un élément minimal par rapport au préordre sur les substitutions $\leq$ ($\leq$ est un cas particulier de $\leq_{SP}$ où $=_{SP} = \{(t, t) \mid t \in T(\Sigma, X)\}$). Cet **unificateur minimal**, unique à un renommage près [Rob65], est souvent appelé l'**unificateur le plus général** (most general unifier). Par extension aux atomes, on dira que les atomes $A(t_1,...,t_n)$ et $B(s_1,...,s_n)$ sont unifiables via σ si et seulement si les symboles A et B sont identiques et $\sigma(t_i) = \sigma(s_i)$ pour i=1...n. Plusieurs propositions d'algorithmes d'unification ont été faites. Voir par exemple [PW78][MM82][Hue76].

Définition 2.8 : (variante)

Soit E une entité syntaxique: terme, atome ou clause. On dit que E' est une variante de E si et seulement si E' est obtenue à partir de E en renommant les variables de E.

2.2. SLD-RESOLUTION

Le premier calcul que nous considérons est basé sur la règle dite SLD-résolution[1]. Cette règle est un cas particulier de la règle de Résolution de Robinson [Rob65]. Elle a été introduite par Kowalski et Kuehner [KK71] dans le cadre des clauses de Horn sans égalité. Nous montrons dans ce paragraphe que la SLD-résolution reste complète dans le cadre de la logique des clauses de Horn avec égalité lorsqu'on ajoute, aux axiomes de chaque spécification, les axiomes caractérisant le prédicat d'égalité (AXE(Σ)). Un résultat similaire peut être consulté dans [Höl89].

Nous définissons ci-dessous la SLD-résolution ainsi que l'ensemble de solutions calculé dans le cas des spécifications dans la logique des clauses de Horn avec égalité. Nous illustrons ensuite ce calcul sur un exemple simple, puis nous montrons sa cohérence et sa complétude.

2.2.1. SLD-résolution

Définition 2.9 : (SLD-résolution)

Soient SP=(Σ, C) une spécification et B une clause de but $\Leftarrow A_1,\ldots, A_m,\ldots, A_n$. Soit Cl une variante de clause dans C de la forme $A \Leftarrow D_1,\ldots,D_k$. On déduit le but B' de B et Cl par **SLD-résolution** ssi :

- B' est le but $\Leftarrow \theta\ (A_1,\ldots, A_{m-1},D_1,\ldots,D_k,A_{m+1},\ldots, A_n)$
- A_m est un atome appelé l'**atome sélectionné**,
- θ est le plus petit unificateur de A et A_m

La clause de but B' est appelé **résolvante** des clauses B et Cl. Nous noterons cette dérivation par : $B \xrightarrow{SLD}_{[m,cl,\theta]} B'$.

Définition 2.10 : (SLD-dérivation)

Soient SP=(Σ, C) une spécification et B une clause de but. Une **SLD-dérivation** de C et B consiste en :

- une suite de clauses de buts : $B_0\ B_1\ldots\ B_n$ (où B_0=B),
- une suite de variantes de clauses dans C : $Cl_0\ Cl_1\ldots\ Cl_{n-1}$ et
- une suite d'unificateurs minimaux : $\theta_0\theta_1\ldots\ \theta_{n-1}$

---

[1]En anglais *SLD-Resolution* est mis pour *SL-resolution for Definite clauses*. De même *SL-resolution* est l'abréviation de *Linear resolution with Selection function. Definite clauses* est synonyme de *Horn clauses*.

telles que une clause de but B_{i+1} est la résolvante de B_i et Cl_i, obtenue en utilisant l'unificateur θ_i.

$B_0 \xrightarrow{SLD} [i_0, cl_0, \theta_0] B_1 \ldots\ldots \xrightarrow{SLD} [i_{n-2}, cl_{n-2}, \theta_{n-2}] B_{n-1} \xrightarrow{SLD} [i_{n-1}, cl_{n-1}, \theta_{n-1}] B_n$

SLD-dérivation

Définition 2.11 : (SLD-réfutation)

Une **SLD-réfutation** est une SLD-dérivation dont la suite des clauses de but (B_0 B_1...) se termine par la clause vide (B_0 B_1...[]).

Définition 2.12 : (solution calculée)

Soient SP=(Σ, C) une spécification, B une clause de but et V l'ensemble des variables de B (V = vars (B)). Une **solution calculée** par SLD-résolution est la restriction aux variables de B de la composition $\theta_{n-1}...\theta_0$, c'est-à-dire $\theta_{n-1}...\theta_0|_V$, où les θ_i représentent la suite des unificateurs minimaux lors d'une SLD-réfutation de C et B : $BB_1...B_{n-1}[]$.

Nous sommes maintenant en mesure de définir l'ensemble de solutions calculé par SLD-résolution, étant donné une spécification dans la logique des clauses de Horn avec égalité SP=(Σ, C) et une clause de but B= $\Leftarrow A_1...A_n$.

Définition 2.13 : ($Sol_{SLD\text{-}résolution}$)

Soient SP=(Σ, C) une spécification, $A_1...A_n$ un but et V l'ensemble des variables figurant dans les A_i. On définit l'ensemble de solutions pour le but $A_1...A_n$ par : $\mathbf{Sol_{SLD\text{-}résolution}}$ **(SP, $A_1...A_n$)** = {$\theta_{n-1}...\theta_0|_V$ telle que $\theta_{n-1}...\theta_1$ est une solution calculée par SLD-résolution à partir des clauses dans **C** ∪ **AXE(Σ)** et de la clause de but $\Leftarrow$ $A_1...A_n$}.

Rappelons que AXE(Σ) est l'ensemble des clauses de Horn spécifiant la congruence du prédicat d'égalité (définition 1.20). La justification de cette approche est démontrée dans le §2.2.3. Elle s'appuie d'une part, sur la cohérence et la complétude de la SLD-résolution pour la logique des clauses de Horn sans égalité, et d'autre part sur le théorème 2.1 où nous montrons que tout atome est valide dans E-MS_{SP} si et seulement si il est valide dans MS_{SPE}, avec SPE la spécification obtenue à partir de SP en ajoutant aux axiomes de SP les axiomes AXE(Σ).

2.2.2. Exemple

Nous considérons la spécification SP=(Σ, C) vue dans le premier paragraphe. Nous rappelons d'abord la signature Σ et les clauses de C. Ensuite, sachant que les clauses utilisées lors des SLD-dérivations font partie des axiomes de la spécification SPE = (Σ, C ∪ AXE(Σ)) (définition 2.13), nous donnons aussi les clauses de AXE(Σ).

$\Sigma = (S, \Omega, \Pi)$

$S=\{nat\}$

$\Omega= \{\{0 : \rightarrow nat\},\{s : nat \rightarrow nat\}, \{+ : nat, nat \rightarrow nat\}\}$

$\Pi= \{\{P: nat, nat, nat\},\{Q: nat\}, \{== : nat, nat\}\}$

C ={ Cl1 : $0 + x == x$

Cl2 : $s(x) + y == s(x + y)$

Cl3 : $P(x, y, z) \Leftarrow x + y == z$

Cl4 : $Q(x) \Leftarrow P(y, y, x)$ }

AXE(Σ) = {

Cl5 : $x == x$

Cl6 : $x == y \Leftarrow y == x$

Cl7 : $x == z \Leftarrow x == y, y == z$

Cl8 : $s(x) == s(y) \Leftarrow x == y$

Cl9 : $x + y == z + u \Leftarrow x == z, y == u$

Cl10: $P(x, y, z) \Leftarrow P(u, v, w), x == u, y == v, z == w$

Cl11: $Q(x) \Leftarrow Q(y), x == y$ }

Remarque 2.3 : Rappelons que les variables d'une clause sont universellement quantifiées. L'utilisation de variantes de clauses lors des SLD-dérivations se traduit ci-dessous par l'introduction de variables indicées. Nous supposons que toutes les expressions sont bien typées.

Nous nous proposons de résoudre les deux buts suivants : Q(s(s(0))) puis Q(x + x). Nous avons :

$\Leftarrow Q(s(s(0)))$ $\xrightarrow{SLD}_{[1,cl_4,\theta_0]} \Leftarrow P(y_1, y_1, s(s(0)))$

$\xrightarrow{SLD}_{[1,cl_3,\theta_1]} \Leftarrow y_1 + y_1 == s(s(0))$

$\xrightarrow{SLD}_{[1,cl_7,\theta_2]} \Leftarrow y_1 + y_1 == y_3, y_3 == s(s(0))$

$\xrightarrow{SLD}_{[1,cl_2,\theta_3]} \Leftarrow s(x_4 + y_4) == s(s(0))$

$\xrightarrow{SLD}_{[1,cl_8,\theta_4]} \Leftarrow x_4 + y_4 == s(0)$

$\xrightarrow{SLD}_{[1,cl_1,\theta_5]} []$

Les substitutions θ_i utilisées ci-dessus sont définies par :

$\theta_0 = \{x_1 \rightarrow s(s(0))\}$

$\theta_1 = \{x_2 \rightarrow y_1, y_2 \rightarrow y_1, z_2 \rightarrow s(s(0))\}$

$\theta_2 = \{x_3 \rightarrow y_1 + y_2, z_3 \rightarrow s(s(0))\}$

$\theta_3 = \{y_1 \rightarrow s(x_4), y_4 \rightarrow s(x_4), y_3 \rightarrow s(x_4 + y_4)\}$

$\theta_4 = \{x_5 \rightarrow x_4 + y_4, y_5 \rightarrow s(0)\}$

$\theta_5 = \{x_4 \rightarrow 0, x_6 \rightarrow s(0), y_4 \rightarrow s(0)\}$

Puisque le but Q(s(s(0))) ne contient pas de variables, la réfutation de la clause de but "$\Leftarrow$ Q(s(s(0)))" prouve la validité de l'atome Q(s(s(0))) dans E-MS_{SP}. Il est clair qu'il existe une infinité de SLD-dérivations issues de la clause "$\Leftarrow$ Q(s(s(0)))". Pour s'en convaincre il suffit de considérer, par exemple, la clause Cl6 (symétrie) ou bien la clause Cl7 (transitivité).

$\Leftarrow Q(x + x) \quad \overset{SLD}{\dashrightarrow} [1, cl_4, \sigma_0] \Leftarrow P(y_1, y_1, x + x)$

$\overset{SLD}{\dashrightarrow} [1, cl_3, \sigma_1] \Leftarrow y_1 + y_1 == x + x$

Posons B2 = $\Leftarrow y_1 + y_1 == x + x$. On a alors :

B2 $\overset{SLD}{\dashrightarrow} [1, cl_5, \sigma_2] []$

B2 $\overset{SLD}{\dashrightarrow} [1, cl_7, \sigma'_2] \Leftarrow y_1 + y_1 == z_3, z_3 == x + x$

$\overset{SLD}{\dashrightarrow} [1, cl_1, \sigma_3] \cdots$

L'ensemble de solutions calculé pour le but Q(x+x) est infini. On peut vérifier que $Sol_{SLD\text{-résolution}}$ (SP, Q(x + x)) = {id, {x $\rightarrow$ 0}, {x$\rightarrow$ s(y)}, ..., {x $\rightarrow$ $s^n(0)$}, {x $\rightarrow$ $s^n(y)$}.

2.2.3. Complétude de la SLD-résolution

Nous montrons dans ce paragraphe la complétude de $Sol_{SLD\text{-résolution}}$(SP, B). Nous commençons par donner le lien entre l'ensemble des atomes valides dans E-MS_{SP} et l'ensemble des atomes valides dans MS_{SPE} pour une spécification dans la logique des clauses de Horn avec égalité SP.

Théorème 2.1 : Soit SP=(Σ, C) une spécification dans la logique des clauses de Horn avec égalité. Soit SPE = (Σ, C $\cup$ AXE(Σ)). Pour tout atome $A(t_1,...,t_n)$, on a :

E-MS_{SP} $\models A(t_1,...,t_n) \Leftrightarrow$ $MS_{SPE} \models A(t_1,...,t_n)$.

Preuve : Nous utilisons dans la démonstration la transformation classique T_{SPE}[1] définie sur les interprétations de Herbrand ainsi que la transformation ET_{SP} définie sur les E-interprétations de Herbrand (définition 1.43) dont MS_{SPE} et E-MS_{SP} sont respectivement les plus petits points fixes ($MS_{SPE} = T_{SPE}\uparrow\omega$, E-$MS_{SP} = ET_{SP}\uparrow\omega$). Dans ce qui suit nous omettrons les indices des transformations T et ET.

($\Rightarrow$) E-$MS_{SP} \models A(t_1,...,t_n)$ signifie que pour toute affectation $a : X \rightarrow T(\Sigma)_{\equiv}$, où $X=vars(A(t_1,...,t_n))$, $(a^{\#}(t_1),...,a^{\#}(t_n))$ est un élément de l'interprétation du symbole de prédicat A : $A_{E\text{-}MS_{SP}}$. D'après la remarque sur les affectations (cf.preuve du lemme 1.1), ceci est équivalent à la proposition suivante : "pour toute affectation $\alpha : X \rightarrow T(\Sigma)$, $([\alpha^{\#}(t_1)],...,[\alpha^{\#}(t_n)])$ est un élément de $A_{E\text{-}MS_{SP}}$". Afin d'établir l'implication, montrons que $(\alpha^{\#}(t_1),...,\alpha^{\#}(t_n))$ est un élément de $A_{MS_{SPE}}$. Pour cela nous procédons par induction sur n, le nombre de pas nécessaire pour que le n-uplet $([\alpha^{\#}(t_1)],...,[\alpha^{\#}(t_n)])$ appartienne à $ET\uparrow n$.

Si n=1, par définition de la transformation ET, il existe une clause (un "fait") de la forme $A(\tau_1,...,\tau_n)$ dans C et une affectation $\beta : X' \rightarrow T(\Sigma)$, où $X'=vars(A(\tau_1,...,\tau_n))$, telles que pour tout i =1...n, $\alpha^{\#}(t_i) \equiv \beta^{\#}(\tau_i)$. D'où l'on a, par définition de $\equiv$ (proposition 1.2): pour tout i=1...n, $(\alpha^{\#}(t_i), \beta^{\#}(\tau_i))$ est un élément de $==_{T\uparrow\omega}$. D'autre part, par définition de la transformation T, $(\beta^{\#}(\tau_1),...,\beta^{\#}(\tau_n))$ est élément de $T\uparrow 1$. La clause "$A(x_1,...,x_n) \Leftarrow A(y_1,...,y_n)$, $x_1=y_1,...,x_n=y_n$" étant élément de AXE(Σ), On en conclut que $(\alpha^{\#}(t_1),...,\alpha^{\#}(t_n))$ est élément de $A_{T\uparrow\omega}$.

Hypothèse d'induction : pour toute affectation $\alpha : X \rightarrow T(\Sigma)$, $([\alpha^{\#}(t_1)],...,[\alpha^{\#}(t_n)]) \in A_{ET\uparrow n}$ ($n \geq 1$) implique que $(\alpha^{\#}(t_1),...,\alpha^{\#}(t_n)) \in A_{T\uparrow\omega}$.

Soit $([\alpha^{\#}(t_1)],...,[\alpha^{\#}(t_n)])$ un élément de $A_{ET\uparrow(n+1)}$. Deux cas sont possibles, à savoir :

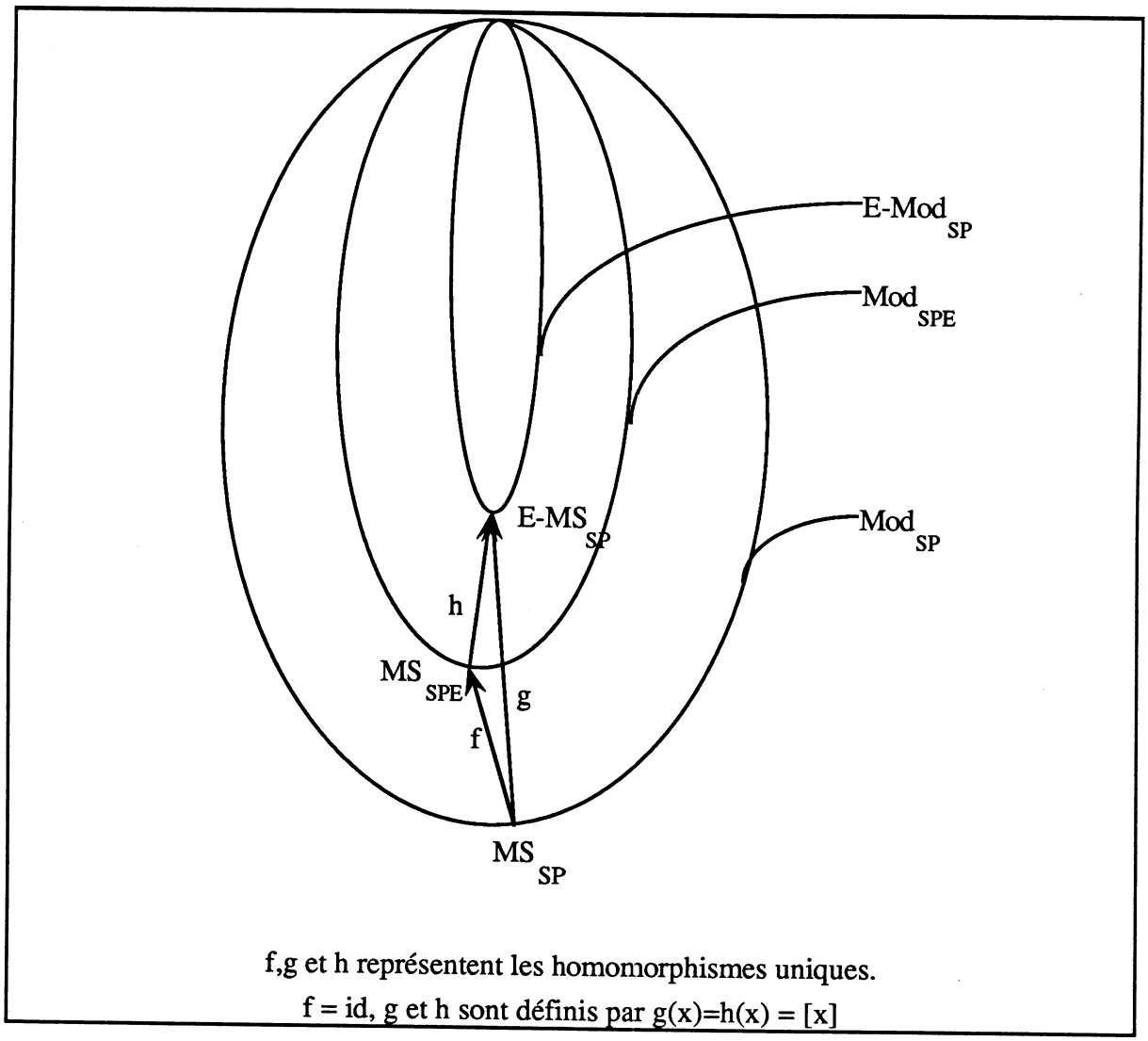
1 : $([\alpha^{\#}(t_1)],...,[\alpha^{\#}(t_n)]) \in A_{ET\uparrow n}$. Par hypothèse, $(\alpha^{\#}(t_1),...,\alpha^{\#}(t_n)) \in A_{T\uparrow\omega}$.

2 : $([\alpha^{\#}(t_1)],...,[\alpha^{\#}(t_n)]) \in (A_{ET\uparrow(n+1)} - A_{ET\uparrow n})$. Dans ce cas, par définition de ET, il existe une clause Cl dans C de la forme "$A(\tau_1,...,\tau_n) \Leftarrow Q_1(u^1_1,...,u^1_{q_1}),...,Q_p(u^p_1,...,u^p_{q_p})$" et une affectation $\beta : X'' \rightarrow T(\Sigma)$ avec X" représentant l'ensemble des variables figurant dans la clause Cl, telles que $\alpha^{\#}(t_i) \equiv \beta^{\#}(\tau_i)$ pour tout i=1...n et $([\beta^{\#}(u^j_1)],...,[\beta^{\#}(u^j_{q_j})]) \in Q_{j_{ET\uparrow n}}$ pour j =1...p. D'après l'hypothèse d'induction, on déduit $(\beta^{\#}(u^j_1),...,\beta^{\#}(u^j_{q_j})) \in Q_{j_{T\uparrow\omega}}$ pour j =1...p. On a alors par définition de la transformation T: $(\beta^{\#}(\tau_1),...,\beta^{\#}(\tau_n)) \in A_{T\uparrow\omega}$. Comme $\alpha^{\#}(t_i) \equiv \beta^{\#}(\tau_i)$ et d'après la clause "$A(x_1,...,x_n) \Leftarrow A(y_1,...,y_n)$, $x_1=y_1,...,x_n=y_n$", on déduit immédiatement que $(\alpha^{\#}(t_1),...,\alpha^{\#}(t_n)) \in A_{T\uparrow\omega}$.

Ceci étant valable pour n'importe quelle affectation α . On en conclut que $MS_{SPE} \models A(t_1,...,t_n)$.

---

[1]La définition de la transformation T_{SP} est définie sur les interprétations de Herbrand. Sa définition peut être obtenu à partir de celle de ET_{SP} (cf. §1.2.3.2) en remplaçant $\equiv$ par =. T_{SP} est continue et $pppf(T_{SP}) = MS_{SP}=T_{SP}\uparrow\omega$. Pour plus de détails sur cette transformation, voir par exemple [AE82][Llo84].



f,g et h représentent les homomorphismes uniques.
f = id, g et h sont définis par g(x)=h(x) = [x]

(⇐) MS_{SPE} étant initial dans la classe des modèles Mod_{SPE}. $E\text{-}MS_{SP}$ est un modèle de SP. L'homomorphisme unique h : $MS_{SPE} \rightarrow E\text{-}MS_{SP}$ (cf. figure ci-dessus) rend le résultat immédiat.

Théorème 2.2 : (Complétude de $Sol_{SLD\text{-résolution}}(SP, A_1 \ldots A_n)$)

Soient SP=(Σ, C) une spécification dans la logique des clauses de Horn avec égalité, $A_1 \ldots A_n$ un but et W un ensemble de variables contenant $\cup_{i=1..n} vars(A_i)$. Alors, $Sol_{SLD\text{-résolution}}$ (SP, $A_1 \ldots A_n$) est un ensemble complet pour le but $A_1 \ldots A_n$ en dehors de W.

Preuve : conséquence immédiate du théorème 2.1 et de la cohérence et la complétude de la SLD-résolution. Pour les preuves de la cohérence et de la complétude de la SLD-résolution, voir par exemple [Llo87, p.43] pour la cohérence et [Llo87,p.49] pour la complétude. Notons que ces preuves doivent subir une légère modification pour tenir compte

des définitions que nous donnons de la cohérence (dans E-MS_{SP}) et de la complétude (par rapport aux solutions fermées).

2.3. SLDEI-RESOLUTION

La SLD-résolution, présentée dans le paragraphe précédent, prend en compte le prédicat d'égalité par l'intermédiaire de son axiomatisation (utilisation de AXE(Σ)). Nous étudions dans ce paragraphe un autre calcul que nous appelons SLDEI-résolution. Ce nouveau calcul tente à éviter les déductions "gloutonnes" provoquées par les axiomes dans AXE(Σ). La SLDEI-résolution est une forme particulière de la SLD-résolution où le traitement du prédicat d'égalité se fait par des algorithmes spécialisés, appelés algorithmes de E-unification ou plutôt EI-unification dans notre cas. EI-unifier deux termes t et t' revient à résoudre l'équation t == t' dans le E-modèle standard E-MS_{SP}. L'idée d'utiliser des algorithmes destinés à la résolution d'équations pour généraliser la règle de Résolution a été proposée pour la première fois par Plotkin dans [Plo72].

Nous commençons ce paragraphe par la définition de la EI-unification. Nous introduisons ensuite la règle de SLDEI-résolution : une extension de la SLD-résolution où l'unification "syntaxique" est remplacée par la EI-unification. Puis, nous montrons la cohérence et la complétude de ce deuxième calcul. Nous terminons ce paragraphe en prouvant la complétude "forte" de la SLDEI-résolution, justifiant ainsi notre implantation décrite dans le cinquième chapitre.

2.3.1. EI-unification

La EI-unification peut être informellement définie comme étant la résolution d'un système d'équations $\{t_i == t_i', 1 \leq i \leq n\}$ dans le E-modèle standard. Nous donnons dans ce paragraphe les notions de base sur la EI-unification nécessaires pour la compréhension des définitions de la SLDEI-résolution. Nous détaillerons davantage la EI-unification dans le chapitre suivant.

Définition 2.14 : (EI-unification)

Soit SP=(Σ, C) une spécification dans la logique des clauses de Horn avec égalité. On dit que deux termes t et t' sont **EI-unifiables** si et seulement si il existe une substitution θ telle que $\theta(t) =_{SP} \theta(t')$, c'est-à-dire E-$MS_{SP} \models \theta(t) == \theta(t')$. Une telle substitution θ est appelée **EI-unificateur** de t et t'. On notera par **EIU(t,t',SP)** l'ensemble des EI-unificateurs de t et t', $EIU(t,t',SP) = \{\theta \in SUB \mid \text{E-}MS_{SP} \models \theta(t) == \theta(t')\}$. Par extension, nous noterons $EIU((t_1,...,t_n), (t'_1,...,t'_n), SP)$ l'ensemble des EI-unificateurs pour le système d'équations $\{t_i == t'_i , 1 \leq i \leq n\}$. L'ensemble $EIU(P(t_1,...,t_n), P(s_1,...,s_n), SP)$ est

égal à l'ensemble EIU($(t_1,...,t_n)$, $(s_1,...,s_n)$, SP). D'autre part, si Q et P sont deux symboles de prédicats distincts alors EIU($P(t_1,...,t_n)$, $Q(s_1,...,s_n)$, SP) est vide. On notera **EIUF(t, t', SP)** l'ensemble des EI-unificateurs fermés.

Exemple 2.1 : Soient les deux spécifications SP=(Σ, C) et SP'=(Σ, C') définies par :

Σ = ({n}, {{0:→ n}, {s: n→ n}, {+: n n → n}, {{== : n n},{P: n n n}})

C = { 0 + x == x
s(x) + y == s(x + y)
P(x, y, z) ⇐ x + y == z }

C' = { x + y == z ⇐ P(x, y, z)
P(0, x, x)
P(s(x), y, s(z)) ⇐ P(x, y, z) }

Considérons l'équation "v+w == s(0)". Elle admet deux solutions (EI-unificateurs) incomparables σ_1 et σ_2 définies par : σ_1 = {v→ 0, w→ s(0)} et σ_2 = {v→ s(0), w→ 0}. Il est clair que ces solutions sont valables aussi bien dans E-MS_{SP} que dans E-$MS_{SP'}$.

Définition 2.15 : **(ensemble complet de EI-unificateurs)**

Soit SP=(Σ, C) une spécification. Soient deux termes t et t', et V = vars(t) ∪ vars(t'). Un ensemble U de substitutions est appelé un ensemble complet de EI-unificateurs des deux termes t et t' ssi :

$\forall \sigma \in U, \sigma \in EIU(t, t', SP)$ (cohérence)

$\forall \sigma \in EIUF(t, t', SP), \exists \rho \in U \mid \rho \leq^{i}_{SP} \sigma [V]$ (complétude)

Exemple 2.2 : Considérons les spécifications SP et SP' de l'exemple2.1. Soit l'équation x + 0 == x. les ensembles U1 = {id} et U2 = {{x→ 0}, {x→ s(y)}} sont des ensembles complets de EI-unificateurs pour les termes x + 0 et x. Par contre l'ensemble U3 = {{x→ s(y)}} n'est pas un ensemble complet de EI-unificateurs.

Remarque 2.4 : Un ensemble complet de EI-unificateurs n'est pas forcément constitués de substitutions fermées. Notons aussi la non unicité des ensembles complets de EI-unificateurs.

2.3.2. SLDEI-Résolution

Définition 2.16 : **(SLDEI-résolution)**

Soient SP= (Σ, C) une spécification et B une clause de but ⇐ $A_1,..., A_m,..., A_n$, où A_m est dit atome sélectionné. On déduit le but B' à partir de B par **SLDEI-résolution** dans les deux cas suivants :

(i) A_m est de la forme $P(t_1,...t_n)$ (A_m n'est pas une équation)

Soit Cl une variante de clause dans C de la forme A ⇐ $D_1,...,D_k$.

- B' est le but $\Leftarrow \theta (A_1,\ldots, A_{m-1},D_1,\ldots,D_k,A_{m+1},\ldots, A_n)$
- θ est élément de U(A, A_m, SP), avec U(A, A_m, SP) un ensemble complet de EI-unificateurs de A et A_m.

(ii) A_m est une équation (A_m est de la forme s == t)

- B' est le but $\Leftarrow \theta (A_1,\ldots, A_{m-1},A_{m+1},\ldots, A_n)$
- θ est élément de U(s, t, SP), avec U(s, t, SP) un ensemble complet de EI-unificateurs de s et t.

Le but B' est appelé **EI-résolvante** du but B. Nous noterons une telle dérivation par : $B \text{-----}\overset{SLDEI}{>}_{[m,Cl,\theta]} B'$ dans le premier cas (i) et par $B \text{---}\overset{SLDEI}{>}_{[m,\theta]} B'$ dans le deuxième cas (ii).

Remarque 2.5 : Il existe autant de EI-résolvantes que de EI-unificateurs dans U(A, A_m, SP) ou dans U (s, t, SP).

Exemple 2.3 : Considérons la spécification SP=(Σ, C) dont nous ne donnons que les clauses :

C ={ Cl1 : 0 + x == x
Cl2 : s(x) + y == s(x + y)
Cl3 : P(x, y, x + y) }

Calculons les résolvantes issues des clauses de buts "$\Leftarrow$ P(i, j, s(0))", "$\Leftarrow$ x+0 == x", puis "$\Leftarrow$ P(i, j, s(0)), x+0 == x". Nous prendrons pour cela les ensembles suivants comme ensemble complets de EI-unificateurs.

U(x+0, x, SP) = {id} et
U(P(i, j, s(0)), P(x_1, y_1, x_1+y_1), SP) = {θ_1, θ_2} avec
θ_1 = {i→ 0, x_1→ 0, j→ s(0), y_1→ s(0)} et
θ_2 = {i→ s(0), x_1→ s(0), j→ 0, y_1→ 0}.

$\Leftarrow$ P(i, j, s(0)) $\text{----}\overset{SLDEI}{>}_{[1,Cl3,\theta_1]}$ []
$\Leftarrow$ P(i, j, s(0)) $\text{----}\overset{SLDEI}{>}_{[1,Cl3,\theta_2]}$ []

$\Leftarrow$ x+0 == x $\text{----}\overset{SLDEI}{>}_{[1,id]}$ []

$\Leftarrow$ P(i, j, s(0)), i+0 == i $\text{----}\overset{SLDEI}{>}_{[1,Cl3,\theta_1]}$ $\Leftarrow$ 0+0 == 0 $\text{----}\overset{SLDEI}{>}_{[1,id]}$ []
$\Leftarrow$ P(i, j, s(0)), i+0 == i $\text{----}\overset{SLDEI}{>}_{[1,Cl3,\theta_2]}$ $\Leftarrow$ s(0)+0 == s(0) $\text{----}\overset{SLDEI}{>}_{[1,id]}$ []
$\Leftarrow$ P(i, j, s(0)), i+0 == i $\text{----}\overset{SLDEI}{>}_{[2,id]}$ $\Leftarrow$ P(i, j,s(0)) $\text{----}\overset{SLDEI}{>}_{[1,Cl3,\ldots)]}$ []

Définition 2.17 : (SLDEI-dérivation)

Soient SP=(Σ, C) une spécification et B une clause de but. Une **SLDEI-dérivation** à partir de B consiste en :

- une suite de clauses de buts : $B_0\ B_1 \ldots\ B_n$ (où B_0=B),
- une suite de EI-unificateurs : $\theta_0\theta_1 \ldots\ \theta_{n-1}$

telles qu'une clause de but B_{i+1} est la EI-résolvante de B_i, obtenue en utilisant le EI-unificateur θ_i.

Définition 2.18 : (SLDEI-réfutation)

Une **SLDEI-réfutation** est une SLDEI-dérivation dont la suite des clauses de but ($B_0\ B_1 \ldots$) se termine par la clause vide ($B_0\ B_1 \ldots \square$).

Définition 2.19 : (solution calculée)

Soient SP=(Σ, C) une spécification et B une clause de but et V l'ensemble des variables figurant dans B. Une **solution calculée** pour le but B par SLDEI-résolution est la restriction aux variables dans B de la composition $\theta_{n-1} \ldots \theta_0$, c'est-à-dire $\theta_{n-1} \ldots \theta_0|_V$, où les θ_i représentent la suite des EI-unificateurs lors d'une SLDEI-réfutation issue de B.

2.3.3. Exemple

Nous considérons le même exemple qu'au §2.2.2 dont nous rappelons les clauses:

C ={ Cl1 : $0 + x == x$
Cl2 : $s(x) + y == s(x + y)$
Cl3 : $P(x, y, z) \Leftarrow x + y == z$
Cl4 : $Q(x) \Leftarrow P(y, y, x)$ }

Suivant l'exemple du §2.2.2, nous nous proposons de résoudre les deux buts Q(s(s(0))) puis Q(x + x). On a :

En utilisant les ensembles de EI-unificateurs suivants :

$U(Q(s(s(0))), Q(x_1), SP) = \{\theta_1\}$ avec $\theta_1 = \{x_1 \rightarrow s(s(0))\}$

$U(P(y_1, y_1, s(s(0))), P(x_2, y_2, z_2), SP) = \{\theta_2\}$
avec $\theta_2 = \{x_2 \rightarrow y_1, y_2 \rightarrow y_1, z_2 \rightarrow s(s(0))\}$

$U(y_1 + y_1, s(s(0)), SP) = \{\theta_3\}$
avec $\theta_3 = \{y_1 \rightarrow s(0)\}$

On obtient :

$\Leftarrow Q(s(s(0)))$ ---SLDEI->$[1,Cl4,\theta_1] \Leftarrow P(y_1, y_1, s(s(0)))$
---SLDEI->$[1,Cl3,\theta_2] \Leftarrow y_1 + y_1 == s(s(0))$

$$\text{---}\overset{SLDEI}{\text{---}}\!>_{[1,\theta_3]}\ []$$

Considérons maintenant le but $\Leftarrow Q(x+x)$. On utilisera pour cela les ensembles de EI-unificateurs suivants :

$$U(Q(x+x), Q(x_1), SP) = \{\sigma_1\}$$
$$\text{avec } \sigma_1 = \{x_1 \rightarrow x + x\}$$
$$U(P(y_1, y_1, x+x), P(x_2, y_2, z_2), SP) = \{\sigma_2\}$$
$$\text{avec } \sigma_2 = \{x_2 \rightarrow y_1, y_2 \rightarrow y_1, z_2 \rightarrow x+x\}$$
$$U(y_1+y_1, x+x, SP) = \{\sigma_3\}$$
$$\text{avec } \sigma_3 = \{y_1 \rightarrow x\}$$

On obtient :

$$\Leftarrow Q(x+x) \ \text{---}\overset{SLDEI}{\text{---}}\!>_{[1,Cl4,\sigma 1)]} \Leftarrow P(y_1, y_1, x+x)$$
$$\text{---}\overset{SLDEI}{\text{---}}\!>_{[1,Cl4,\sigma 2)]} \Leftarrow y_1+y_1 == x+x$$
$$\text{---}\overset{SLDEI}{\text{---}}\!>_{[1,\sigma 3)]}\ [].$$

Le calcul des ensembles complets de EI-unificateurs est important dans ce genre de dérivation. Pour le deuxième exemple ci-dessus, le calcul par SLDEI-résolution donne une seule solution qui est l'identité ($\sigma_3\sigma_2\sigma_1|_{\{x\}} = id$). Cependant, nous aurions pu choisir des ensembles complets de EI-unificateurs de manière à retrouver le même ensemble infini de solutions calculé par la SLD-résolution (voir l'exemple du §2.2.2). La SLDEI-résolution paraît alors meilleurs que la SLD-résolution. Le calcul des ensembles complets de EI-unificateurs est étudié dans le chapitre suivant.

2.3.4. Cohérence et Complétude de la SLDEI-résolution

Commençons par étudier la cohérence des solutions calculées par SLDEI-résolution.

Théorème 2.3 : (Cohérence)

Soient SP=(Σ, C) une spécification et B= $\Leftarrow A_1 \ldots A_q$ une clause de but. Toute solution calculée pour B par SLDEI-résolution est cohérente.

Preuve : par induction sur la longueur des dérivations. Soit $\theta_{r-1}\ldots\theta_0$ la solution calculée; r étant la longueur de la SLDEI-réfutation.

Cas de base (r = 0) On distingue deux cas :

(i) le but est de la forme : $P(t_1,\ldots,t_n)$

La longueur de la réfutation étant égale à 1, il existe un "fait" dans C de la forme $P(s_1,...,s_n)$ et θ_0 est un EI-unificateur de $P(t_1,...,t_n)$ et $P(s_1,...,s_n)$. $\theta_0(P(t_1,...,t_n))$ est donc un atome valide dans E-MS_{SP}.

(ii) le but est de la forme : s == t.

Dans ce cas θ_0 est un EI-unificateur de s et de t. Donc $\theta_0(s = t)$ est valide dans E-MS_{SP}.

Hypothèse d'induction : Toute solution calculée par une SLDEI-réfutation de longueur inférieure ou égale à r est cohérente.

Soit $\theta_r\theta_{r-1}...\theta_0$ une solution calculée par une SLDEI-réfutation de longueur (r+1) pour le but $\Leftarrow A_1...A_q$. Soit A_m l'atome sélectionné à la première dérivation. Deux cas sont possibles : (i) A_m est de la forme $P(t_1,...,t_n)$ ou bien (ii) A_m est une équation de la forme (s == t).

(i) A_m est de la forme $P(t_1,...,t_n)$

Dans ce cas, il existe une variante de clause dans C : $P(s_1,...,s_n) \Leftarrow Q_1...Q_u$ telle que θ_0 est un élément de $U(P(t_1,...,t_n), P(s_1,...,s_n), SP)$. le but B_1 obtenu est de la forme "$\Leftarrow \theta_0(A_1...A_{m-1}Q_1...Q_uA_{m+1}...A_q)$". Par hypothèse d'induction, la solution $\theta_r\theta_{r-1}...\theta_1$ est cohérente pour le but B_1. Donc tous les atomes $\theta_r\theta_{r-1}...\theta_1\theta_0(A_j)$ avec j dans $\{1...q\}-\{m\}$ sont valides dans E-MS_{SP} ainsi que les atomes $\theta_r\theta_{r-1}...\theta_1\theta_0(Q_j)$ avec j dans $\{1...u\}$. D'après la définition de la validité d'une clause, on déduit la validité de $\theta_r\theta_{r-1}...\theta_1\theta_0(A_m)$ dans E-MS_{SP}. D'où la solution calculée est cohérente.

(ii) A_m est de la forme (s == t)

Dans ce cas θ_0 appartient à U(s, t, SP). B_1 est de la forme "$\Leftarrow\theta_0(A_1...A_{m-1}\ A_{m+1}...A_q)$". D'après l'hypothèse d'induction, tous les atomes $\theta_r\theta_{r-1}...\theta_1\theta_0(A_j)$, avec j dans $\{1...q\}$ - $\{m\}$, sont valides dans E-MS_{SP}. De plus $\theta_r\theta_{r-1}...\theta_1\theta_0(A_m)$ est valide dans E-MS_{SP} car $\theta_r\theta_{r-1}...\theta_1\theta_0$ est une instance de θ_0. cqfd.

Afin de prouver la complétude de la SLDEI-résolution, nous montrons d'abord les deux lemmes suivants. Le premier (lemme 2.1) établit un lien entre le calcul itératif du E-modèle standard par la transformation ET et la réfutation par SLDEI-résolution. Dans le second lemme (lemme 2.2), nous montrons la complétude de la SLDEI-résolution dans le cas des buts fermés.

Lemme 2.1 : Soient SP=(Σ, C) une spécification et P un symbole de prédicat. Si $([t_1],..., [t_n])$ appartient à $\beta_k(P)$, avec β_k tel que $ET\uparrow k = (T(\Sigma)_{\equiv}, \alpha, \beta_k)$, alors il existe une SLDEI-réfutation pour la clause de but "$\Leftarrow P(t_1,...,t_n)$".

Preuve: par induction sur k.

Cas de base : k = 1. Dans ce cas il existe un "fait", $P(s_1,...,s_n)$, dans C et une substitution fermée σ tels que $\sigma(s_i) \equiv^1 t_i$ pour i =1...n. σ est un EI-unificateur de $P(s_1,...,s_n)$ et $P(t_1,...,t_n)$. Il existe donc une substitution α élément de $U(P(s_1,...,s_n), P(t_1,...,t_n), SP)$ telle que $\alpha \leq^i_{SP} \sigma$ [vars $(P(s_1,...,s_n))$]. D'où la possibilité de SLDEI-réfutation, en un pas, pour "$\Leftarrow P(t_1,...,t_n)$".

Hypothèse d'induction : $([t_1],..., [t_n]) \in \beta_k(P) \Rightarrow$ Il existe une SLDE-réfutation pour "$\Leftarrow P(t_1,...,t_n)$".

Soit $([t_1],..., [t_n]) \in \beta_{k+1}(P)$. Alors, par définition de la transformation ET (définition1.43), il existe une instance fermée de clause dans C de la forme "$P(s_1,...,s_n) \Leftarrow Q_1(s^1_1,...,s^1_{n_1})...Q_q(s^q_1,...,s^q_{n_q})$" telle que $t_i \equiv s_i$ pour i =1...n et pour tout j=1...q, $(s^j_1,...,s^j_{n_j}) \in \beta_k(Q_j)$. En appliquant q fois l'hypothèse d'induction, on conclut à l'existence d'une SLDEI-réfutation pour le but " $\Leftarrow Q_1(s^1_1,...,s^1_{n_1})...Q_q(s^q_1,...,s^q_{n_q})$". D'où la SLDEI-réfutation pour le but "$\Leftarrow P(t_1,...,t_n)$". (Notons que lorsque q vaut 0, cela revient au cas de base).

Lemme 2.2 : Soient SP=(Σ, C) une spécification et B= $\Leftarrow A_1...A_q$ une clause de but telle que les atomes A_i soient fermés. Si tous les A_i sont valides dans E-MS_{SP}, alors il existe une SLDEI-réfutation pour B.

Preuve : De la clause B, on peut déduire par SLDEI-résolution la clause B' où toutes les équations présentes dans la clause B ont été éliminées. Soit B' de la forme "$\Leftarrow A'_1...A'_{q'}$". Soit $A'_i = P(t_1,...,t_n)$, A'_i étant valide dans E-MS_{SP}, d'après (théorème 1.2), il existe un entier naturel k tel que $([t_1],..., [t_n])$ appartient à $\beta_k(P)$, avec β_k tel que $ET\uparrow k = (T(\Sigma)_{\equiv}, \alpha, \beta_k)$. D'après le lemme 2.1 ci-dessus, il existe une SLDEI-réfutation pour A'_i. La SLDEI-réfutation pour B s'en déduit immédiatement.

Théorème 2.4 : (Complétude)

Soient SP=(Σ, C) une spécification, B= $\Leftarrow A_1...A_q$ une clause de but et σ une solution fermée pour B. Alors, il existe une solution θ calculée par SLDEI-résolution telle que $\theta \leq^i_{SP} \sigma$ [vars(B)].

Preuve : σ est une solution (fermée) pour B signifie que $\sigma(A_i)$ est valide dans E-MS_{SP} pour i =1...n. D'après le lemme2.2, il existe une SLDEI-réfutation pour σ(B) de longueur L. Montrons le théorème par induction sur la longueur L.

Cas de base : L = 1. Dans ce cas q vaut 1 et A_1 est soit une équation s==t ou bien un atome de la forme $P(t_1,...,t_n)$.

(i) A_1 est de la forme (s == t). σ est alors un EI-unificateur de s et t. U(s, t, SP) étant complet, il existe un EI-unificateur θ dans U(s, t, SP) tel que $\theta \leq^i_{SP} \sigma$ [vars(B)].

---

[1] cf. définition 1.43 .

(ii) A_1 est de la forme $P(t_1,...,t_n)$. Dans ce cas, il existe un "fait" $P(s_1,...,s_n)$ tel que $\sigma(P(t_1,...,t_n))$ et $P(s_1,...,s_n)$ sont EI-unifiables. D'où l'existence d'une substitution θ élément de $U(P(t_1,...,t_n), P(s_1,...,s_n), SP)$ telle que $\theta \leq^{i}_{SP} \sigma$ [vars(B)].

Hypothèse d'induction : Soit σ une solution fermée pour un but $B= \Leftarrow A_1...A_n$. Si $\sigma(B)$ admet une SLDEI-réfutation de longueur r inférieure ou égale à L, alors il existe une SLDEI-réfutation pour B de longueur r telle que $\theta_{r-1}...\theta_0 \leq^{i}_{SP} \sigma$ [vars(B)].

Soit σ une solution fermée pour un but $B= \Leftarrow A_1...A_q$. D'après le lemme2.2, $\sigma(B)$ admet une SLDEI-réfutation. Supposons que cette SLDEI-réfutation soit de longueur r+1, avec r = L. Montrons qu'il existe une SLDEI-réfutation pour B de longueur r+1 telle que $\theta_r\theta_{r-1}...\theta_0 \leq^{i}_{SP} \sigma$ [vars(B)]. Dans la SLDEI-réfutation de $\sigma(B)$, supposons que $\sigma(A_m)$ soit l'atome sélectionné à la première SLDEI-résolution. Deux cas sont possibles selon la forme de A_m :

(a) A_m est de la forme (s == t) :

On a $\sigma(s) \equiv \sigma(t)$ car $\sigma(A_m)$ est valide dans E-MS_{SP}. D'où l'on déduit, de la complétude de U(s, t, SP), l'existence d'une substitution θ_0 élément de U(s, t, SP) telle que $\theta_0 \leq^{i}_{SP} \sigma$ [vars (s == t)]. Le but $B_1 = \theta_0(\Leftarrow A_1...A_{m-1}\ A_{m+1}...A_q)$ est alors une EI-résolvante de B. Or, par définition de $\leq^{i}_{SP}$, il existe une substitution fermée β telle que $\beta\theta_0 \equiv^{i}_{SP} \sigma$[vars (s == t)]. D'où la clause de but "$\beta(\theta_0(\Leftarrow A_1...A_{m-1}A_{m+1}...A_q))$", qui est une instance fermée du but B_1, admet une SLDEI-réfutation de longueur r. Par hypothèse d'induction, il existe une SLDEI-réfutation pour B_1 de longueur r telle que $\theta_r\theta_{r-1}...\theta_1 \leq^{i}_{SP} \beta$ [vars(B1)]. D'où la SLDEI-réfutation pour B de longueur (r+1) qui est telle que la solution calculée $\theta_r\theta_{r-1}...\theta_1\theta_0$ vérifie $\theta_r\theta_{r-1}...\theta_1\theta_0 \leq^{i}_{SP} \sigma$ [vars(B)].

(b) A_m est de la forme $P(t_1,...,t_n)$:

Par hypothèse $\sigma(B)$ admet une SLDEI-réfutation et $\sigma(A_m)$ est le premier atome sélectionné. Donc il existe une variante de clause dans C de la forme "$P(s_1,...,s_n) \Leftarrow Q_1(s^1_1,...,s^1_{n_1})...Q_u(s^u_1,...,s^u_{n_u})$" et une substitution fermée α avec $D(\alpha) = vars((s_1,...,s_n))$ telles que $\alpha(s_i) \equiv \sigma(t_i)$. Etant donné $D(\alpha) \cap D(\sigma) = \emptyset$, la composition $\alpha\sigma$ est un EI-unificateur (fermé) de $P(t_1,...,t_n)$ et $P(s_1,...,s_n)$. D'où l'existence d'un EI-unificateur θ_0 élément de $U(P(t_1,...,t_n), P(s_1,...,s_n), SP)$ tel que $\theta_0 \leq^{i}_{SP} \alpha\ \sigma$ [vars$((s_1,...,s_n)) \cup$ vars $((t_1,...,t_n))$]. Il existe donc une substitution fermée β telle que $\beta\theta_0 \equiv^{i}_{SP} \alpha\sigma$[vars $((s_1,...,s_n)) \cup$ vars $((t_1,...,t_n))$]. Soit B_1 = "$\theta_0(\Leftarrow A_1...A_{m-1}\ Q_1(s^1_1,...,s^1_{n_1})...Q_u(s^u_1,...,s^u_{n_u})A_{m+1}...A_q)$". B_1 est une EI-résolvante du but B. Comme $\beta(B_1)$ admet une SLDEI-réfutation de longueur r, on déduit par hypothèse d'induction qu'il existe une SLDEI-réfutation pour B_1 telle que $\theta_r\theta_{r-1}...\theta_1 \leq^{i}_{SP} \beta$ [vars(B1)]. D'autre part, on a $\alpha\sigma \equiv^{i}_{SP} \sigma$ [vars(B)]. D'où la SLDEI-réfutation pour B, de longueur (r+1), qui est telle que la solution calculée $\theta_r\theta_{r-1}...\theta_1\theta_0$ vérifie $\theta_r\theta_{r-1}...\theta_1\theta_0 \leq^{i}_{SP} \sigma$ [vars(B)].

Nous pouvons maintenant énoncer un théorème de complétude pour le calcul de solutions par SLDEI-résolution.

Théorème 2.5 : (Complétude de $Sol_{SLDEI\text{-résolution}}$ (SP, B))

Soient SP=(Σ, C) une spécification, B = $A_1 \ldots A_n$ un but et W un ensemble de variables contenant V=vars(B). Soit $Sol_{SLDEI\text{-résolution}}$ (SP, B) l'ensemble des solutions calculées par SLDEI-résolution à partir de la clause de but "$\Leftarrow A_1 \ldots A_n$". Alors, $Sol_{SLDEI\text{-résolution}}$ (SP, B) est un ensemble complet pour le but B en dehors de W.

Preuve : conséquence directe des théorèmes 2.3 (cohérence) et 2.4 (complétude).

2.3.5. Complétude forte de la SLDEI-résolution

La complétude de la SLDEI-résolution telle qu'elle est énoncée dans le théorème2.5 ne donne aucune indication quant au choix de l'atome sélectionné pour effectuer une étape de déduction. Aussi, tout algorithme de recherche de solutions basé sur la SLDEI-résolution doit-il considérer toutes les sélections possibles des atomes d'un but afin d'assurer la complétude du calcul. L'objectif de ce paragraphe est de montrer que ce non déterminisme dans le calcul peut être évité et que l'algorithme obtenu produit toujours un ensemble complet de solutions. C'est ce que l'on appelle la complétude "forte" [Llo87]. Ceci est réalisé en introduisant la notion de stratégie dans les SLDEI-dérivations.

Définition 2.20 : **(stratégie de résolution)**

On définit une stratégie de Résolution comme étant une fonction (partielle) de l'ensemble des buts BUT vers l'ensemble des naturels non nuls $\mathbb{N}_+$; SR : BUT $\rightarrow \mathbb{N}_+$ telle que pour tout but B = $A_1 \ldots A_n$, SR(B) = i, avec $1 \leq i \leq n$. i correspond à la position de l'atome, A_i, sélectionné dans le "but" pour la déduction de EI-résolvantes. SR(B) n'est pas défini si aucune EI-résolvante n'est déductible de B, notamment lorsque B = [].

Définition 2.21 : **(SLDEI-dérivation avec une stratégie)**

Soient SP une spécification, B= $\Leftarrow A_1 \ldots A_n$ une clause de but et SR une stratégie de résolution. Une SLDEI-dérivation (resp. SLDEI-réfutation) avec la stratégie SR est une SLDEI-dérivation (resp. SLDEI-réfutation) dans laquelle l'atome sélectionné, lors de chaque calcul de nouvelle EI-résolvante B_{j+1} à partir de B_j, est désigné par la stratégie SR.

La SLDEI-résolution avec une stratégie définit un calcul complet. Pour montrer ce résultat, nous énonçons et prouvons le lemme technique suivant. Intuitivement, ce lemme 2.3 montre que l'on peut toujours permuter les choix des atomes sélectionnés, lors d'une SLDEI-réfutation.

Lemme 2.3 : Soient SP=(Σ, C) une spécification, B une clause de but et α une solution fermée pour B. Pour toute SLDEI-réfutation issue de B de longueur r :

(1) $B_0=B \xrightarrow{SLDEI} [i_0,cl_0,\theta_0]\ B_1 \ldots B_{j-1} \xrightarrow{SLDEI} [m,cl_{j-1},\theta_{j-1}]\ B_j \xrightarrow{SLDEI} [n,cl_j,\theta_j]\ B_{j+1} \ldots []$

telle que :

$B_{j-1} = \Leftarrow A_1 \ldots A_m \ldots A_n \ldots A_q$

$\theta_{r-1} \ldots \theta_j \theta_{j-1} \ldots \theta_0 \leq^i_{SP} \alpha\ [vars(B)]$

Alors, en sélectionnant l'atome A_n au j-ème pas de dérivation et l'atome A_m au (j+1)-ième pas de déduction, on peut obtenir une SLDEI-réfutation issue de B dont la solution engendrée est plus petite (selon $\leq^i_{SP}$) que α.

Preuve : Par cas sur les formes des atomes A_m et A_n.

- Cas où ni A_n ni A_m ne sont des équations. Soient cl_{j-1} = "$T_{j-1} \Leftarrow Cp_{j-1}$" et cl_j = "$T_j \Leftarrow Cp_j$" avec T et Cp désignent respectivement la tête et le corps de la clause cl. On a alors B_j = "$\Leftarrow \theta_{j-1}(A_1 \ldots Cp_{j-1} \ldots A_n \ldots A_q)$" et B_{j+1} = "$\Leftarrow \theta_j\theta_{j-1}(A_1 \ldots Cp_{j-1} \ldots Cp_j \ldots A_q)$". Posons $\theta'_{j-1} = \theta'_j = \theta_j\theta_{j-1}$ et considérons la SLDEI-réfutation suivante:

(2) $B_0=B \xrightarrow{SLDEI} [i_0,cl_0,\theta_0]\ B_1 \ldots B_{j-1} \xrightarrow{SLDEI} [m,cl_j,\theta'_{j-1}]\ B'_j \xrightarrow{SLDEI} [n,cl_{j-1},\theta'_j]\ B_{j+1} \ldots []$

où B'_j = "$\Leftarrow \theta_j\theta_{j-1}(A_1 \ldots A_m \ldots Cp_j \ldots A_q)$".

Vérifions d'abord que la dérivation (2) a bien un sens. D'abord θ'_{j-1} est un EI-unificateur de A_n et T_j. En effet, d'après la SLDEI-réfutation (1) on a θ_j est élément de $U(\theta_{j-1}(A_n), T_j, SP)$, $D(\theta_j) \subseteq I(\theta_{j-1}) \cup vars(T_j)$ et $vars(T_j) \cap D(\theta_{j-1}) = \emptyset$. D'où θ'_{j-1} est élément de $EIU(A_n, T_j, SP)$. D'autre part θ'_j est un EI-unificateur de $\theta'_{j-1}(A_m)$ et T_{j-1}. En effet, on sait d'après (1) que θ_{j-1} est un EI-unificateur de A_m et T_{j-1}, et donc θ'_{j-1} n'est qu'une instance de θ_{j-1}. Le but B_{j+1} est obtenu à partir de B'_j grâce à l'idempotence des substitutions. Pour terminer cette partie de la preuve, on doit tenir compte de la définition de la SLDEI-résolution dont les substitutions utilisées appartiennent à un ensemble complet par rapport aux solutions fermées. On ne peut donc pas exiger que θ'_{j-1} soit élément d'un tel ensemble. Or par hypothèse $\theta_{r-1} \ldots \theta_j\theta_{j-1} \ldots \theta_0 \leq^i_{SP} \alpha\ [vars(B)]$, il existe donc une substitution fermée β telle que $\beta\theta_{r-1} \ldots \theta_j\theta_{j-1} \ldots \theta_0 \equiv^i_{SP} \alpha\ [vars(B)]$. On peut facilement dériver de la SLDEI-réfutation (2) une nouvelle SLDEI-réfutation en prenant comme substitutions $\theta''_{j-1} = \beta\theta_{r-1} \ldots \theta_j\theta_{j-1}$ pour j=1...r.

- Autres cas : A_m ou A_n sont des équations. Ces cas ne posent pas de problèmes supplémentaires. Il suffit de ne pas considérer les (corps des) clauses cl_{j-1} ou cl_j.cqfd.

Nous sommes maintenant à même d'énoncer les deux résultats de complétude suivants.

Théorème 2.6 : **(Complétude forte de la SLDEI-résolution)**

Soient SP=(Σ, C) une spécification, B= $\Leftarrow A_1 \ldots A_q$ une clause de but, σ une solution fermée pour B et SR une stratégie de résolution. Alors, il existe une solution θ calculée par SLDEI-résolution avec la stratégie SR telle que $\theta \leq^{i}_{SP} \sigma$ [vars(B)].

Preuve: Conséquence de la complétude de la SLDEI-résolution (théorème 2.4) et du lemme 2.3.

Théorème 2.7 : (Complétude de $Sol^{SR}_{SLDEI\text{-résolution}}$ (SP, B))

Soient SP=(Σ, C) une spécification, B= $A_1 \ldots A_n$ un but, W un ensemble de variables contenant V=vars(B) et SR une stratégie de Résolution. Soit $Sol^{SR}_{SLDEI\text{-résolution}}$ (SP, B) l'ensemble des solutions calculée par SLDEI-résolution à partir de la clause de but "$\Leftarrow A_1 \ldots A_n$" avec la stratégie SR. Alors $Sol^{SR}_{SLDEI\text{-résolution}}$ (SP, B) est un ensemble complet pour le but B en dehors de W.

Preuve : conséquence du théorème 2.3 (cohérence de la SLDEI-résolution) et du théorème 2.6 (complétude de la SLDEI-résolution avec une stratégie).

2.4. CONCLUSION

La règle de Résolution a été introduite par Robinson pour le calcul dans la logique du premier ordre sans égalité. La SLD-résolution de Kowalski et Kuehner est un raffinement de la Résolution au cas de la logique des clauses de Horn sans égalité. Dans ce cas, l'unification dans la Résolution reflète la congruence utilisée sur les termes. En effet, les spécifications ne définissant pas le prédicat d'égalité, la congruence utilisée est réduite alors à la diagonale {(t, t) | t ∈ T(Σ,X)}. Cette congruence représente en fait à la fois $=_{SP}$ et $=^{i}_{SP}$.

En présence de spécifications dans la logique des clauses de Horn avec égalité, les deux congruences $=_{SP}$ et $=^{i}_{SP}$ sont différentes dans le cas général ($=_{SP} \subseteq =^{i}_{SP}$, cf. chapitre 1). Dans ce mémoire nous désignons par "algorithme de E-unification" (respectivement "algorithme de EI-unification") les algorithmes de résolution "d'équations" prenant comme sémantique d'égalité la congruence $=_{SP}$ (respectivement la congruence $=^{i}_{SP}$). La substitution de l'unification (syntaxique) dans la règle de SLD-résolution par des algorithmes de E-unification ou des algorithmes de EI-unification donne lieu à deux règles différentes : la SLDE-résolution qui prend en compte la congruence $=_{SP}$, voir par exemple, Gallier et Raatz [GR89], Jaffar et al.[JLM84], Hölldobler [Höl88]. L'autre règle est celle que nous avons présentée dans ce chapitre, à savoir la SLDEI-résolution. Evidemment, les définitions de complétude et de cohérence sont différentes pour les deux calculs. Cette dernière règle est certainement plus appropriée que la SLDE-résolution, pour les calculs dans les langages de programmation (à sémantique initiale). Cette extension de la SLD-résolution paraît actuellement comme l'une des voies intéressantes pour réussir l'intégration des langages à la fois fonctionnels et prédicatifs [BE86] [GM84] [BBBLM86] [BR90].

Le calcul dans la logique du premier ordre avec égalité a fait l'objet de nombreuses études, voir par exemple [RW69] [Mor69] [Plo72] [CL73] [HR78] [Sti85]. Parmi ces propositions, la Paramodulation de Robinson et Wos [RW69] est certainement le calcul le plus étudié. La paramodulation a été améliorer par rapport à sa version originelle par Brand [Bra75] et Peterson [Pet83] en évitant l'utilisation de certains axiomes (de réflexivité fonctionnelle). Des raffinement de ce calcul se trouvent dans [Lov78] et [Rus89]. Nous n'avons pas considéré la Paramodulation dans notre étude à cause de sa généralité. La SLDEI-résolution étudiée dans ce mémoire en est un cas particulier qui s'applique aux clauses de Horn avec égalité.

Chapitre 3

Surréduction

Nous avons vu dans le deuxième chapitre que la déduction à l'aide de la SLDEI-résolution nécessite des algorithmes de résolution d'équations. Il y a plusieurs façons d'aborder ce problème; nous citons les deux cas extrêmes: (i) on s'intéresse à des algorithmes de résolution d'équations dans des théories particulières comme dans [Plo72], (ii) on utilise un algorithme général de résolution comme dans [GS88]. Dans ce chapitre, nous étudions des algorithmes de résolution d'équations qui font partie de la deuxième approche et qui sont valables en présence de systèmes de réécriture canoniques. Ces algorithmes sont basés sur la relation appelée surréduction.

Nous commençons ce chapitre par quelques rappels préliminaires sur la résolution d'équations (on dit aussi : E-unification, unification universelle, unification sémantique) suivis de résultats classiques sur la surréduction, notamment la complétude des algorithmes de résolution d'équations basés sur la surréduction et la surréduction normale. Nous améliorons ensuite ces algorithmes en utilisant des stratégies de surréduction. L'utilisation de ces stratégies ne donnant pas toujours des algorithmes complets, nous présentons des conditions suffisantes sur ces stratégies afin d'assurer la complétude des algorithmes. Nous étudierons enfin une amélioration de ces algorithmes en utilisant un raffinement de la surréduction que nous appelons surréduction inductivement normale.

3.1. PRELIMINAIRES

Dans ce chapitre, et sauf mention contraire, nous considérons des spécifications SP=(Σ, C) où l'ensemble d'axiomes C est scindé en deux sous-ensembles E et CH. L'ensemble E est constitué d'équations et l'ensemble CH est constitué de clauses de Horn dont les têtes de clauses ne sont pas des équations. La raison de ce partage est technique et correspond au champ d'application des algorithmes présentés dans ce chapitre. Ainsi, les clauses dans CH n'interviennent pas dans la définition des congruences $=_{SP}$ et $=\!\!\!\!\!\!\dot{}_{SP}$. Ces congruences sont donc complètement définies par la seule donnée de E (nous les noterons parfois $=_E$ et $=\!\!\!\!\!\!\dot{}_E$).

3.1.1. E-unification

Nous rappelons ci-après les notions de base concernant la E-unification. Pour plus de détails, voir par exemple [Sie89, Kir85, Kni89, JK90]. La E-unification peut être informellement définie comme la résolution de systèmes d'équations $\{t_i == t_i', 1 \leq i \leq n\}$ dans une théorie équationnelle donnée. Les solutions des équations sont alors exprimées à l'aide des substitutions.

Définition 3.1: (substitution)

Soient Σ une signature et X un ensemble de variables. Une **substitution** σ est un endomorphisme sur $T(\Sigma, X)$. On note **SUB** l'ensemble des substitutions. L'ensemble $\mathbf{D}(\sigma) = \{x \in X \mid \sigma(x) \neq x\}$ est appelé **domaine** de la substitution σ et $\mathbf{I}(\sigma) = \cup_{x \in D(\sigma)} vars(\sigma(x))$ est l'ensemble des **variables introduites** par σ. La composition des substitutions est notée $\sigma\sigma'$ avec $\sigma\sigma'(t) = \sigma(\sigma'(t))$. Dans la suite, on confondra une substitution σ avec l'application $\sigma'': X \rightarrow T(\Sigma, X)$ dont σ est l'unique extension ($T(\Sigma,X)$ étant libre sur X). Une substitution est dite **fermée** si $I(\sigma)$ est vide. Soient V un ensemble de variables et σ une substitution, on note par $\sigma|_V$ la restriction de la substitution σ aux variables de V, définie par : $\sigma|_V(x) = \sigma(x)$ si $x \in V$, $\sigma|_V(x) = x$ si $x \notin V$. Nous supposons dans la suite que toutes les substitutions sont idempotentes, c'est-à-dire $\sigma\sigma = \sigma$, pour toute substitution σ.

Soient σ une substitution et $D(\sigma) = \{x_i , i = 1...m\}$ son domaine tel que $\sigma(x_i) = t_i$ pour $i = 1...m$. On notera une telle substitution par $\sigma = \{x_1 \rightarrow t_1, ..., x_m \rightarrow t_m\}$.

On définit deux préordres sur SUB, $\leq_{SP}$ et $\leq^{i}_{SP}$. Intuitivement, $\sigma \leq_{SP} \beta$ (respectivement $\sigma \leq^{i}_{SP} \beta$) signifie que la substitution σ est "moins instanciée" modulo la congruence $=_{SP}$ (respectivement la congruence $=^{i}_{SP}$) que la substitution β. Nous définissons ces deux préordres en fonction de deux relations d'équivalence sur l'ensemble des substitutions, $\equiv_{SP}$ et $\equiv^{i}_{SP}$, définies comme suit : Soit V un ensemble de variables.

$\forall \sigma, \theta \in SUB, \sigma \equiv_{SP} \theta\ [V] \Leftrightarrow \forall x \in V, \sigma(x) =_{SP} \theta(x)$
$\forall \sigma, \theta \in SUB, \sigma \equiv^{i}_{SP} \theta\ [V] \Leftrightarrow \forall x \in V, \sigma(x) =^{i}_{SP} \theta(x)$
$\forall \sigma, \theta \in SUB, \sigma \leq_{SP} \theta\ [V] \Leftrightarrow \exists \delta \in SUB \mid \delta\sigma \equiv_{SP} \theta\ [V]$
$\forall \sigma, \theta \in SUB, \sigma \leq^{i}_{SP} \theta\ [V] \Leftrightarrow \exists \delta \in SUB \mid \delta\sigma \equiv^{i}_{SP} \theta\ [V]$

Définition 3.2 : (E-unification)

Soit $SP = (\Sigma, E \cup CH)$ une spécification. On dit que deux termes t et t' sont E-unifiables si et seulement si il existe une substitution θ telle que $\theta(t) == \theta(t')$ soit une conséquence logique de SP, c'est-à-dire $\theta(t) =_{SP} \theta(t')$. Une telle substitution θ, qui résout l'équation $t == t'$ par rapport à $E\text{-}Mod_{SP}$, est dite **E-unificateur** de t et t'. On notera par **EU(SP, t, t')** l'ensemble des E-unificateurs de t et t'. Par extension, nous noterons $EU((t_1,...,t_n), (t'_1,...,t'_n), SP)$ l'ensemble des E-unificateurs pour le système d'équations $\{t_i == t'_i , 1 \leq i$

$\leq n\}$. Autrement dit $EU(SP, (t_1,...,t_n), (t'_1,...,t'_n)) = \{\sigma \in SUB \mid \forall\ i=1...n,\ \sigma(t_i) =_{SP} \sigma(t'_i)\}$.

Exemple 3.1[FH86]: Soit $SP=(\Sigma, E \cup CH)$ telle que $E = \{f(0, x) == x,\ h(f(x, y)) == h(y)\}$.

Considérons l'équation $h(0) == h(z)$. L'ensemble des substitutions

$$D = \{\{z \rightarrow 0\}\} \cup \{\{z \rightarrow f(v_i, f(v_{i-1}, \ldots f(v_0, 0)\ldots))\} \mid i \geq 0\}$$

est un ensemble de E-unificateurs des termes h(0) et h(z).

L'ensemble des E-unificateurs de t et t', EU(SP, t, t'), est récursivement énumérable. Souvent on cherche à calculer un sous-ensemble ES de EU(SP, t, t') à partir duquel on peut déduire par instanciation, toutes les substitutions dans EU(SP, t, t') [Plo72]. Un tel ensemble est appelé ensemble complet de E-unificateurs.

Définition 3.3 : (ensemble complet de E-unificateurs)

Soient $SP=(\Sigma,E \cup CH)$ une spécification, t et t' deux termes et W un ensemble de variables contenant $V = vars(t) \cup vars(t')$. Un ensemble de substitutions ES est un ensemble complet de E-unificateurs de t et t' en dehors de W ssi:

(1) $\forall\ \sigma \in ES,\ D(\sigma) \subseteq V$ et $I(\sigma) \cap W = \emptyset$

(2) $\forall\ \sigma \in ES,\ \sigma \in EU(SP, t, t')$

(3) $\forall\ \sigma \in EU(SP, t, t'),\ \exists\ \sigma' \in ES \mid \sigma' \leq_{SP} \sigma\ [V]$

L'introduction de l'ensemble W relève de considérations techniques. W représente un contexte de variables déjà utilisées. La première condition peut toujours être vérifiée, il suffit de renommer les variables de façon adéquate. La deuxième condition exprime la cohérence de l'ensemble ES et la troisième exprime sa complétude.

Exemple 3.2 : L'ensemble D dans l'exemple 3.1.1 est un ensemble complet de E-unificateurs pour les termes h(0) et h(z).

La définition d'un ensemble complet de E-unificateurs peut être raffinée de manière à ce que les E-unificateurs soient tous distincts. On parle alors dans ce cas d'ensemble minimal de E-unificateurs. Cette notion généralise celle de l'unificateur minimal (m.g.u.) [Rob65].

Définition 3.4 : (ensemble minimal de E-unificateurs)

Soient $SP=(\Sigma,E \cup CH)$ une spécification, t et t' deux termes, W un ensemble de variables contenant $V = vars(t) \cup vars(t')$ et ES un ensemble complet de E-unificateurs de t et t' en dehors de W. ES est dit minimal ssi il satisfait de plus la condition :

$\forall \sigma, \sigma' \in ES, \sigma \leq_{SP} \sigma' [V] \Rightarrow \sigma = \sigma'$.

Selon la spécification et les équations à résoudre, un ensemble minimal de E-unificateurs peut ne pas exister, même lorsque le système d'équations considéré admet des solutions. En effet les deux termes de l'exemple 3.1.1 sont E-unifiables mais n'admettent pas d'ensemble minimal de E-unificateurs comme cela est montré dans [FH86]. D'autre part, un ensemble minimal de E-unificateurs, quand il existe, est unique (au renommage près et modulo la congruence sur les termes $=_{SP}$), et son cardinal n'est pas toujours fini. Nous renvoyons le lecteur intéressé par une discussion sur la cardinalité des ensembles minimaux de E-unificateurs à [Sie89] où l'on trouve une hiérarchisation des problèmes de E-unification suivant ces cardinalités.

Les définitions de E-unificateur et d'ensemble complet de E-unificateurs ont été motivées par des préoccupations dans le domaine de la démonstration automatique. C'est pour cela que les E-unificateurs sont définis par rapport à tous les E-modèles possibles, à savoir la classe E-Mod_{SP}. Dans ce mémoire, nous nous intéressons aux déductions dans le E-modèle standard, qui est la sémantique que nous avons retenue pour les spécifications. Nous modifions donc les définitions ci-dessus de la E-unification pour l'adapter à nos besoins. Nous parlerons alors de la EI-unification "E-unification inductive" et d'ensemble complet de EI-unificateurs.

Définition 3.5 : (EI-unification)

Soit $SP=(\Sigma, E \cup CH)$ une spécification. On dit que deux termes t et t' sont EI-unifiables si et seulement si il existe une substitution θ telle que $\theta(t) == \theta(t')$ soit valide dans E-MS_{SP}, c'est-à-dire $\theta(t) =\dot{S}_P \theta(t')$. Une telle substitution θ est appelée **EI-unificateur** de t et t'. On notera par **EIU(t,t',SP)** l'ensemble des EI-unificateurs de t et t'. Par extension, nous noterons $EIU(SP, (t_1,...,t_n), (t'_1,...,t'_n))$ l'ensemble des EI-unificateurs pour le système d'équations $\{t_i == t'_i , 1\leq i \leq n\}$. Autrement dit, $EIU((t_1,...,t_n), (t'_1,...,t'_n), SP) = \{\sigma \in SUB \mid \forall\ i=1...n, \sigma(t_i) =\dot{S}_P \sigma(t'_i)\}$. Nous noterons **EIUF(SP, t, t')** le sous ensemble de EIU(SP, t, t') constitué de EI-unificateurs fermés.

Par analogie avec les définitions sur la E-unification, nous introduisons les notions d'ensemble complet de EI-unificateurs et d'ensemble minimal de EI-unificateurs.

Définition 3.6 : (ensemble complet de EI-unificateurs)

Soient $SP=(\Sigma,E \cup CH)$ une spécification, t et t' deux termes et W un ensemble de variables contenant $V = vars(t) \cup vars(t')$. Un ensemble de substitutions ES est un ensemble complet de EI-unificateurs de t et t' en dehors de W ssi:

(a) $\forall \sigma \in ES, D(\sigma) \subseteq V$ et $I(\sigma) \cap W = \emptyset$

(b) $\forall \sigma \in ES, \sigma \in EIU(SP, t, t')$

(c) $\forall \sigma \in EIUF(SP, t, t'), \exists \sigma' \in ES \mid \sigma' \leq^{i}_{SP} \sigma [V]$

L'ensemble ES est dit **minimal** ssi il satisfait de plus la condition

(d) $\forall \sigma, \sigma' \in ES, \sigma \leq^{i}_{SP} \sigma' [V] \Rightarrow \sigma = \sigma'$.

Exemple 3.3 : Soit SP=(Σ, E $\cup$ CH) telle que E = {0 + x == x, s(x)+ y == s(x+y)}. Considérons l'équation à résoudre "z + 0 == z". Les ensembles F et G suivants sont des ensembles complets de EI-unificateurs pour les termes z + 0 et z.

$F = \{\{z \rightarrow 0\}\} \cup \{\{z \rightarrow s^i(0)\} \mid i \geq 1\}$

$G = \{id\}$

Notons que pour les termes z+0 et z, la substitution id est un EI-unificateur et non pas un E-unificateur.

La définition d'un ensemble complet de EI-unificateurs est donnée par rapport aux EI-unificateurs fermés. Evidemment, ceci n'implique pas qu'un ensemble complet de EI-unificateurs soit constitué exclusivement de substitutions fermées. Un ensemble minimal de EI-unificateurs existe toujours quand les termes considérés sont EI-unifiables, mais il n'est pas forcément unique, contrairement au cas de la E-unification. Notons enfin que tout E-unificateur (respectivement ensemble complet de E-unificateurs) de deux termes est un EI-unificateur (respectivement un ensemble complet de EI-unificateurs) de ces termes ; la réciproque est fausse.

3.1.2. Système de réécriture

Etant donné une spécification SP = (Σ, E $\cup$ CH) et une équation t == t', montrer que t $=_{SP}$ t, revient à déduire l'équation t == t' à partir des équations dans E, en utilisant le raisonnement équationnel. Une telle déduction équationnelle peut être simplifiée lorsqu'il existe un système de réécriture "défini" par E. Nous rappelons ci-dessous les principales définitions et propriétés des systèmes de réécriture utiles pour le reste du mémoire. Pour plus de détails, voir par exemple [DJ89] [HO80].

Afin de définir la relation de réduction induite par un système de réécriture, nous introduisons d'abord quelques outils syntaxiques, à savoir les positions de sous-termes d'un terme donné et le remplacement dans un terme.

Définition 3.7 : (positions)

Soit t un terme ($t \in T(\Sigma, X)$). On désigne par **Pos(t)** l'ensemble des positions de tous ses sous-termes. Pos(t) est un sous ensemble de $\mathbb{N}_+^*$, où $\mathbb{N}_+^*$ désigne l'ensemble des

chaînes constituées d'entiers naturels non nuls. La chaîne vide est notée ε. Pos(t) est défini inductivement par les deux règles suivantes :

(1) $\varepsilon \in Pos(t)$ (t est sous-terme de lui-même)

(2) $u \in Pos(t_i) \Rightarrow i.u \in Pos(f(t_1,\ldots,t_i,\ldots,t_n))$ pour $i = 1\ldots n$, $f \in \Sigma$.

Nous utiliserons dans la suite l'ordre lexicographique sur les éléments de $\mathbb{N}^*$, noté $<$, induit par l'ordre classique sur $\mathbb{N}$, pour comparer les positions dans un terme.

Définition 3.8 : (**sous-terme** et **remplacement**)

Soient t un terme et u une position dans Pos(t). On définit le sous-terme de t à la position u, noté **t/u**, par

(1) $t/\varepsilon = t$

(2) $f(t_1,\ldots,t_i,\ldots,t_n)/i.u = t_i/u$ pour $i = 1\ldots n$, $f \in \Sigma$.

On définit le terme obtenu à partir de t en remplaçant son sous-terme à la position u par le terme t', noté **t[u← t']**, par

(3) $t[\varepsilon \leftarrow t'] = t'$

(4) $f(t_1,\ldots,t_i,\ldots,t_n)\ [i.u \leftarrow t'] = f(t_1,\ldots,t_i[u \leftarrow t'],\ldots,t_n)$ pour $i = 1\ldots n$, $f \in \Sigma$

Nous désignons par **Pos*(t)** les positions dans t qui ne sont pas des variables, c'est-à-dire: $Pos^*(t) = Pos(t) - \{u \in Pos(t) \mid t/u \text{ est une variable}\}$.

Définition 3.9 : (**règle de réécriture, système de réécriture**)

Une règle de réécriture est un couple de termes de même sorte, notée **g -----> d**, avec g et d éléments de $T(\Sigma, X)$ tels que $vars(d) \subseteq vars(g)$. g et d sont dits respectivement **membre gauche** et **membre droit** de la règle. Un système de réécriture SR est un couple $SR = (\Sigma, R)$ où Σ est une signature et R un ensemble fini de règles de réécriture.

Définition 3.10 : (**relation de réduction**)

Soit $SR=(\Sigma, R)$ un système de réécriture. SR définit une relation binaire sur les termes $(T(\Sigma,X))$ appelée relation de réduction, que nous noterons ----->_{SR} , ----->_{R} ou simplement ----->. La relation ----->_{SR} est la plus petite relation contenant les couples (g, d) tels que g----->d appartienne à R et est fermée par substitution et remplacement, c'est-à-dire :

$\forall\ t, t' \in T(\Sigma, X)$, t ----->t' $\Rightarrow \forall\ \sigma \in SUB$, $\sigma(t)$ ----->$\sigma(t')$ (stable par substitution)

$\forall\ t, t', t'' \in T(\Sigma, X)$, t ----->t' $\Rightarrow$ t''[u← t] -----> t''[u← t'] (monotonie de ----->)

On dit qu'un terme t se réduit en t' (ou t se réécrit en t') si il existe une position u dans Pos(t), une règle g_i ----->d_i dans R et une substitution σ tels que $\sigma(g_i) = t/u$ et $t' = t[u \leftarrow \sigma(d_i)]$. On écrit **t ------>[u, i] t'** ou simplement t -----> t'. On note la fermeture réflexive et transitive de la relation de réduction par ----->*. Par $=_R$ on note la fermeture

symétrique de ----->*. $=_R$ est en fait la congruence $=_{SP}$ engendrée sur les termes par la spécification SP= (Σ, E_R) où E_R est défini par $E_R = \{g == d \mid g\text{----}> d \in R\}$.

Définition 3.11 : (Propriétés des systèmes de réécriture)
Soit SR=(Σ, R) un système de réécriture. On dit que

(1) SR est **nœthérien** ssi il n'existe aucun terme t duquel part une dérivation infinie de réduction (c'est-à-dire t----->t_1 -----> ...).

(2) SR est **confluent** ssi

$\forall$ t,t',t'' $\in$ T(Σ, X), t----->*t' et t----->*t'' $\Rightarrow \exists$ T$\in$T(Σ,X) | t'----->*T et t''----->*T

(3) SR est **canonique** (ou convergent) ssi SR est nœthérien et confluent.

Les propriétés des systèmes de réécriture citées ci-dessus sont en général non décidable. Néanmoins, la procédure de complétion [KB70] aide dans de nombreux exemples à décider de ces propriétés. A ce sujet voir entre autres [Der87],[Hul80][BDH86] [Hue79] [Buc85].

Définition 3.12 : (forme normale, substitution normalisée)
Soit SR=(Σ, R) un système de réécriture. Un terme t est dit sous forme normale ssi t n'est pas réductible (c'est-à-dire $\nexists$ t' $\in$ T(Σ,X) | t -----> t'). Lorsque t ----->*t'' et t'' est sous forme normale, on dit que t'' est une forme normale de t. Si SR est nœthérien, tout terme a au moins une forme normale. Si SR est confluent, tout terme a au plus une seule forme normale notée **t!**. Si SR est canonique, tout terme a une et une seule forme normale.
Une **substitution normalisée** σ est une substitution telle que pour toute variable x dans son domaine D(σ), σ(x) est sous forme normale.

Lorsqu'un système de réécriture SR = (Σ, R) est canonique, la congruence $=_R$ est décidable et on a l'équivalence t $=_R$ t' $\Leftrightarrow$ t! = t'!. On dira dans la suite du mémoire qu'un ensemble d'équations E (ou une spécification SP=(Σ, E)) définit un système de réécriture canonique R si l'équivalence t $=_{SP}$ t' $\Leftrightarrow$ t! = t'! est vraie.

L'utilisation des systèmes de réécriture comme sémantique opérationnelle des spécifications équationnelles est largement admise aujourd'hui. Cette utilisation consiste entre autres en des interpréteurs pour des langages équationnels [O'Do85] (normalisation, E-unification) ou des démonstrateurs automatiques [HH80] [Rus89].

3.2. SURREDUCTION

Dans ce paragraphe nous présentons des algorithmes de E-unification dans le cas où l'ensemble des équations E définit un système de réécriture canonique. Ces algorithmes se

basent sur la relation dite surréduction (on dit aussi narrowing [Sla74] [Hul80a], superposition [Fri85]). La surréduction est une relation binaire sur les termes ($T(\Sigma, X)$). Elle a été introduite par Slagles dans [Sla74]. Son utilisation dans un algorithme de E-unification a été illustrée pour la première fois par Fay [Fay79]. Les définitions que nous utilisons sont conformes avec [Hul80b] et [DJ89].

Définition 3.13: **(surréduction)**

Soit $SR=(\Sigma, R)$ un système de réécriture. Un terme t est surréductible en t' à la position u, en utilisant la règle g_i ----->d_i ssi

- $u \in Pos^*(t)$
- t/u et g_i sont unifiables
- $t' = \sigma(t[u \leftarrow d_i]) = \sigma(t)[u \leftarrow \sigma(d_i)])$, avec σ l'unificateur minimal de t/u et g_i

Nous noterons cette relation par $t \text{ --N-->}_{[u,i,\sigma]} t'$.("N" dans --N--> veut dire *Narrowing*)

Exemple 3.4 : Soit $SR=(\Sigma, R)$ avec

R = { 1: f(0, x) -----> x
2: h(f(x, y)) -----> h(y) }.

On a : $h(z) \text{ --N-->}_{[\varepsilon,2,\{z \rightarrow f(x,y)\}]} h(y)$

$h(f(0,0)) \text{ --N-->}_{["1",2,\{x \rightarrow 0,\ y \rightarrow 0\}]} h(0)$ la réduction est un cas particulier de la surréduction.

f(h(0), 0) n'est pas surréductible.

Définition 3.14 : **(dérivation de surréduction)**

Soit $SR=(\Sigma, R)$ un système de réécriture. Une dérivation de surréduction issue d'un terme t est la donnée

- d'une suite de terme $t_0, \ldots, t_n$ (où $t=t_0$)
- d'une suite de variantes de règles g_{i_j} ------>d_{i_j}, j=0...n-1
- d'une suite d'unificateurs minimaux σ_j, j=0...n-1

telles que : pour j=0...n-1, $t_j \text{ --N-->}_{[u_j,i_j,\sigma_j]} t_{j+1}$. On note une telle dérivation par

$t \text{ --N-->}_{[u_0,i_0,\sigma_0]} t_1 \text{ --N--> } \ldots \text{ --N-->}_{[u_{n-1},i_{n-1},\sigma_{n-1}]} t_n$

3.2.1. E-unification par surréduction

Lorsqu'un ensemble d'équations E (ou une spécification $SP=(\Sigma, E)$) définit un système de réécriture canonique SR=(S, R), on sait que décider $t =_{SP} t'$ est équivalent au test t! = t'!. Par conséquent, pour tout E-unificateur ρ de deux termes t et t', on a une dérivation de

réduction[1] $\rho(t) == \rho(t')$ ------>* T == T, où T représente la forme normale $\rho(t)!$ et $\rho(t')!$. D'autre part, comme nous le rappelons dans le lemme3.1 ci-dessous, on sait que pour toute dérivation de réduction $\rho(t)$ ------>* t' dans un système de réécriture canonique, avec la substitution ρ normalisée, on peut associer une certaine dérivation de surréduction issue de l'équation t == t'. Ce lemme constitue en fait la remarque clef qui a donné naissance à différents algorithmes de E-unification fondés sur la relation de surréduction.

Lemme 3.1: (relation entre réduction et surréduction)

Soient SR=(Σ, R) un système de réécriture canonique, t un terme, V un ensemble de variables contenant les variables de t et μ une substitution normalisée. Alors pour toute dérivation de réduction

$\mu(t)$ ------>$_{[u_0,i_0]}$ t_1 ------> ... ------>$_{[u_{n-1},i_{n-1}]}$ t_n

Il existe une dérivation de surréduction

t --N-->$_{[u_0,i_0,\sigma_0]}$ t'_1 --N--> ... --N-->$_{[u_{n-1},i_{n-1},\sigma_{n-1}]}$ t'_n

telle que : pour tout j =1...n, il existe une substitution normalisée μ_j avec $\mu_j(t'_j) = t_j$ et $\mu = \mu_j\sigma_{j-1}...\sigma_0$ [V].

Preuve : voir [Hul80b]

Nous donnons maintenant une première définition de ce qu'est une solution calculée par surréduction résolvant une équation t == t'. Le lemme 3.2 et le théorème 3.1 ci-dessous montrent respectivement la cohérence des solutions calculées par surréduction et la complétude d'un tel calcul.

Définition 3.15 : (solution calculée par surréduction)

Soit SP=(Σ, E $\cup$ CH) une spécification telle que l'ensemble des équations E définit un système de réécriture canonique SR=(Σ, R). Soient t et t' deux termes et V= vars(t) $\cup$ vars(t'). Une substitution θ est une solution calculée par surréduction de l'équation t == t' si il existe une dérivation de surréduction

t==t' --N-->$_{[u_0,i_0,\sigma_0]}$ ... --N-->$_{[u_{n-1},i_{n-1},\sigma_{n-1}]}$ $t_n == t'_n$

avec t_n et t'_n unifiables et $\theta = \rho\sigma_{n-1}...\sigma_0|_V$, où ρ est l'unificateur minimal de t_n et t'_n.

Lemme 3.2 : (cohérence des solutions)

Soit SP=(Σ, E $\cup$ CH) une spécification telle que l'ensemble des équations E définit un système de réécriture canonique SR=(Σ, R). Soient t et t' deux termes. Toute solution de l'équation t == t' calculée par surréduction est un E-unificateur de t et t'.

Preuve : évident, il suffit de normaliser $\rho\sigma_{n-1}...\sigma_0$(t==t').

---

[1]Dans ce cas, une équation est vue comme un terme de racine l' "opérateur binaire" ==.

Définition 3.16 : ($Sol_{surréduction}(R, t, t')$)

Soit SR = (Σ, R) un système de réécriture, on note $Sol_{surréduction}(R, t, t')$ l'ensemble de toutes les solutions calculées par surréduction.

Nous avons vu que les substitutions $\rho\sigma_{n-1}\ldots\sigma_0|_V$, telles qu'elles sont définies ci-dessus, sont des E-unificateurs des deux termes de l'équation initiale (t == t'). D'autre part, on sait, pour tout E-unificateur normalisé θ de t et t', en présence d'un système de réécriture canonique, qu'il existe une dérivation de réduction $\theta(t==t')$ ------>* T == T. Or d'après le lemme 3.1, il existe aussi une dérivation de surréduction issue de t == t' telle que la solution calculée par surréduction est plus générale (modulo E) que la solution θ. Ceci établit alors la complétude de $Sol_{surréduction}(R, t, t')$. D'où

Théorème 3.1 : (complétude de $Sol_{surréduction}(R, t, t')$)

Soit SP=(Σ, E $\cup$ CH) une spécification telle que l'ensemble des équations E définit un système de réécriture canonique SR=(Σ, R). Soient t et t' deux termes et W un ensemble de variables contenant vars(t) $\cup$ vars(t'). $Sol_{surréduction}(R, t, t')$ est un ensemble complet de E-unificateurs en dehors de W.

Preuve : voir [Hul80b]

Exemple 3.5 : Considérons l'exemple classique de l'addition des naturels définie par les deux règles suivantes :

R = { 0 + x -----> x

s(x) + y -----> s(x+y) }

Soit à résoudre l'équation x+0 == s(0)+0

On a : $Sol_{surréduction}$ (R, x+0, s(0)+0) = {{x→ s(0)}}. Le lecteur peut vérifier qu'il y a 16 dérivations de surréduction différentes pour calculer cette unique solution.

Dans[Fay79], Fay normalise les termes après chaque pas de surréduction. Il évite ainsi certaines dérivations redondantes. Nous appelons cette relation "surréduction normale" (on dit aussi surréduction normalisante [Rét88], reduced narrowing [Pad88], normal narrowing [DJ89], superréduction [JL86]).

Définition 3.17 : (surréduction normale[1])

---

[1]Une autre définition de la surréduction normale est donnée par t--NN-->[u,i,σ]t' $\Leftrightarrow$ t! --N-->[u,i,σ] t'. voir [Rét88]

Soit SR=(Σ, R) un système de réécriture. On définit la relation de surréduction normale entre deux termes t et t', notée $t \text{ --NN-->}_{[u,i,\sigma]} t'$, par :

$t \text{ --NN-->}_{[u,i,\sigma]} t' \Leftrightarrow t \text{ --N-->}_{[u,i,\sigma]} t''$ et t'=t''!

("NN" dans --NN--> veut dire *Normal Narrowing*)

On définit la notion de dérivation de surréduction normale de façon analogue que pour la surréduction. On note une telle dérivation par

$$t \text{ --NN-->}_{[u_0,i_0,\sigma_0]} t_1 \text{ --NN--> } \ldots \text{ --NN-->}_{[u_{n-1},i_{n-1},\sigma_{n-1}]} t_n$$

Une substitution θ est une solution calculée par surréduction normale pour t et t' si il existe une dérivation

$$t! == t'! \text{ --NN-->}_{[u_0,i_0,\sigma_0]} t_1 == t'_1 \text{ --NN--> } \ldots \text{ --NN-->}_{[u_{n-1},i_{n-1},\sigma_{n-1}]} t_n == t'_n$$

telle que les termes t_n et t'_n soient unifiables, $\theta = \rho\sigma_{n-1}\ldots\sigma_0|_V$ avec ρ l'unificateur minimal de t_n et t'_n et V = vars(t) ∪ vars(t').

Théorème 3.2 : Soit SP=(Σ, E ∪ CH) une spécification telle que l'ensemble des équations E définit un système de réécriture canonique SR=(Σ, R). Soient t et t' deux termes et W un ensemble de variables contenant vars(t) ∪ vars(t'). L'ensemble $Sol^{normale}_{surréduction}$ (R, t, t') constitué des solutions calculées par surréduction normale pour t et t' est un ensemble complet de E-unificateurs en dehors de W.

Preuve: voir [Fay79], se déduit du théorème 3.1 et du fait : σ(t) == σ(t') ⇔ σ(t!) == σ(t'!)

Exemple 3.6 : En considérant les mêmes données que l'exemple 3.5, il existe une seule dérivation de surréduction normale qui calcule la solution unique {x→ s(0)} dans $Sol^{normale}_{surréduction}$ (R, x+0, s(0)+0).

Notons que $Sol^{normale}_{surréduction}$ (R, t, t') est toujours inclus dans $Sol_{surréduction}$(R, t, t').

Remarque 3.1 : Si E définit un système de réécriture SR=(Σ, R) non canonique, $Sol_{surréduction}$(R, t, t') (et a fortiori $Sol^{normale}_{surréduction}$ (R, t, t')) n'est pas complet en général. en effet, soit R = {c---->b, c---->a}où a,b et c sont trois constantes. R n'est pas confluent. $Sol_{surréduction}$(R, a, b) est vide alors que a == b est une conséquence équationnelle de c == b et c == a. D'autre part, si R est confluent mais non nœthérien, par exemple R ={a -----> f(a)}, $Sol_{surréduction}$(R, f(x), x) est vide car aucune dérivation de surréduction n'est possible à partir de f(x) == x. Cependant, il est clair que les substitutions {x→ f^n(a), n ∈ ℕ } avec f^0(a)=a, sont solutions de l'équation f(x) == x.

D'autres améliorations de la E-unification par surréduction ont été proposées. Nous citons par exemple la surréduction basique de Hullot [Hul80], voir aussi [NRS87]. Toutes ces améliorations n'évitent pas entièrement les calculs redondants en général. Nous étudions dans le paragraphe suivant une autre amélioration de la surréduction qui tente d'éliminer cette redondance.

3.2.2. Stratégies de surréduction

Nous avons vu dans le paragraphe précédent des algorithmes basés sur la surréduction (théorèmes 3.1 et 3.2). Ces algorithmes ressemblent dans leur comportement aux calculs par la SLDEI-résolution, c'est-à-dire que leur implantation est non déterministe à la fois sur les positions de surréduction (l'équivalent du choix d'atome pour la SLDEI-résolution) et sur les règles de réécriture candidates à la surréduction (l'équivalent du choix des clauses pour la SLDEI-résolution). Dans le présent paragraphe, nous tâcherons de réduire ce nombre considérable de dérivations de surréduction. L'idée consiste à éliminer totalement le non-déterminisme sur les positions de surréduction. Ceci sera réalisé en introduisant des stratégies de choix de position de surréduction.

Définition 3.18 : (stratégie de surréduction)

Soient SP=(Σ, R $\cup$ CH) une spécification et X un ensemble de variables. Une stratégie de surréduction SS est une fonction partielle de l'ensemble des termes vers l'ensemble des positions $\mathbb{N}+^*$. SS: $T(\Sigma, X) \rightarrow \mathbb{N}+^*.\mathbb{N}+^*$ étant l'ensemble des chaînes construites à partir de l'alphabet $\mathbb{N}+$ (ensemble des naturels non nuls). Si t est surréductible, SS(t) est défini et t est surréductible à la position SS(t). SS(t) n'est pas défini seulement si t n'est pas surréductible. Pour une stratégie SS et un terme t donnés, la position SS(t) dépend évidemment du système de réécriture considéré.

Dans la suite on notera par abus d'écriture SS(t == t') une position de surréduction dans t == t' où l'équation t == t' est vue comme un terme et == comme un opérateur binaire. Nous dirons indifféremment dans la suite, stratégie ou stratégie de surréduction.

Exemple 3.7 : Considérons le système de réécriture R défini dans l'exemple 3.5. Soient les termes s(x) et s((x+(y+0)) + (y+z)). Nous illustrons sur ces deux termes le résultat de deux stratégies très souvent étudiées, à savoir les stratégies left-most-outer-most et left-most-inner-most. Nous définissons d'abord ces deux stratégies. Pour cela, nous introduisons trois ensembles de positions de surréduction d'un terme : Surr(t) ensemble de toutes les positions de surréduction dans t, Inner(t) ensemble des positions de surréduction inner-most et Outer(t) ensemble des positions de surréduction outer-most :

Surr(t) = {p $\in$ Pos(t) | t est surréductible en p}

Inner (t) = {p ∈ Surr(t) | ∄ q ∈ Surr(t), q = p.u}

Outer (t) = {p ∈ Surr(t) | ∄ q ∈ Surr(t), p = q.u}

Les définitions des deux stratégies peuvent être données en cherchant la position minimale[1] dans Inner(t) et Outer(t).

left-most-outer-most (t) = min (Outer(t))

left-most-inner-most (t) = min (Inner(t))

avec min (Ø) = indéfini.

On a alors :

left-most-outer-most(s(x)) n'est pas défini

left-most-outer-most(s((x+(y+0)) + (y+z))) = "11"

left-most-inner-most(s(x)) n'est pas défini

left-most-inner-most(s((x+(y+0)) + (y+z))) = "112"

Définition 3.19 : (dérivation de surréduction avec une stratégie)

Soient SR=(Σ,R) un système de réécriture, X un ensemble de variables et SS une stratégie de surréduction. Une dérivation de surréduction avec la stratégie SS issue d'un terme t (t ∈ T(Σ, X)) est une dérivation de surréduction

$t \text{ --N-->}_{[u_0,i_0,\sigma_0]} t_1 \text{ --N--> } \ldots \text{ --N-->}_{[u_{n-1},i_{n-1},\sigma_{n-1}]} t_n$

telle que $u_i = SS(t_i)$ pour i = 0...n-1

Nous sommes en mesure maintenant de définir l'ensemble des solutions calculées par surréduction avec une stratégie. On notera que l'axiome de la réflexivité n'est utilisé que si les deux termes d'une équation ne sont pas surréductibles.

Définition 3.20 : ($Sol^{SS}_{surréduction}(R, t, t')$)

Soient SR=(Σ, R) un système de réécriture, X un ensemble de variables, SS une stratégie de surréduction et t et t' deux termes de même sorte dans T(Σ, X). On notera par $Sol^{SS}_{surréduction}(R, t, t')$, l'ensemble des substitutions σ telles que il existe une dérivation de surréduction avec SS issue de l'équation t == t'

$t == t' \text{ --N-->}_{[u_0,i_0,\sigma_0]} t_1 == t'_1 \text{ --N--> } \ldots \text{ --N-->}_{[u_{n-1},i_{n-1},\sigma_{n-1}]} t_n == t'_n$

avec :

t_n et t'_n unifiables

$SS(t_n == t'_n)$ n'est pas défini

---

[1]L'ordre sous-jacent est l'ordre lexicographique sur les chaînes d'entiers naturels, induit par l'ordre classique sur les entiers naturels.

$\sigma = \rho\sigma_{n-1}\sigma_{n-2}\ldots\sigma_0|_{vars(t) \cup vars(t')}$

ρ est l'unificateur minimal de t_n et t'_n.

Les substitutions dans $Sol^{SS}_{surréduction}(R, t, t')$ sont les **solutions** calculées par surréduction avec la stratégie SS.

Le lecteur a certainement fait le rapprochement entre le calcul de $Sol^{SS}_{surréduction}$ (R, t, t') et celui de $Sol^{SR}_{SLDEI\text{-}résolution}$ (SP, B) où SR est une stratégie qui sélectionne l'atome candidat à la SLDEI-résolution. Contrairement à la SLDEI-résolution (cf.§2.3.5), la surréduction n'est pas fortement complète. En effet

Proposition 3.1 : Soient SP=(Σ, E $\cup$ CH) une spécification telle que l'ensemble E définit un système de réécriture canonique SR=(Σ, R), X un ensemble de variables et SS une stratégie de surréduction. l'ensemble $Sol^{SS}_{surréduction}$ (R, t, t') n'est pas un ensemble complet de EI-unificateurs dans le cas général.

Contre-exemple : Soit SP = (Σ, E) une spécification avec E défini par

E = { 1 : f(0, 0) == 0
2 : f(s(x), 0) == s(0)
3 : f(x, s(y)) == s(s(0)) }

Le système de réécriture R, déduit de E par orientation des équations de la gauche vers la droite, est canonique. Résolvons l'équation f(f(i, j), k) == 0 avec la stratégie left-most-outer-most. Il y a une seule dérivation de surréduction avec la stratégie considérée, issue de l'équation f(f(i, j), k) == 0 : (left-most-outer-most (f(f(i, j), k) == 0) = "1").

f(f(i, j), k) == 0 --N-->["1",3, {<x, f(i,j)>, <k, s(y)>}] s(s(0)) == 0

On a alors $Sol^{SS}_{surréduction}$ (R, f(f(i, j), k), 0) = Ø. D'où l'incomplétude du calcul puisque la substitution $\sigma = \{i \rightarrow 0, j \rightarrow 0, k \rightarrow 0\}$ est solution de l'équation considérée.

Le but de ce qui suit est de donner des conditions suffisantes garantissant la complétude de l'ensemble $Sol^{SS}_{surréduction}$ (R, t, t'). Nous commençons par introduire une propriété sur les stratégies de surréduction .

Définition 3.21 : (**stratégie uniforme**[1])

Soient SR=(Σ,R) un système de réécriture canonique, X un ensemble de variables et SS une stratégie de surréduction. La stratégie SS est **uniforme** ssi pour tout terme t (t$\in$

[1]Dans [Pad88], Padawitz donne une définition différente de l'uniformité qui n'a pas d'équivalent ici.

$T(\Sigma,X)$), si SS(t) est défini alors, pour toute substitution fermée normalisée μ, $\mu(t)$ est réductible à la position SS(t).

L'utilisation des stratégies de surréduction uniformes garantit la complétude de $Sol^{SS}_{surréduction}$ (R, t, t'). En effet

Théorème 3.3 : (complétude de $Sol^{SS}_{surréduction}$ (R, t, t'))

Soient SP=(Σ, E $\cup$ CH) une spécification telle que l'ensemble E définit un système de réécriture canonique SR=(Σ, R), t et t' deux termes, W un ensemble de variables contenant V = vars(t) $\cup$ vars(t') et SS une stratégie de surréduction uniforme. Alors, l'ensemble $Sol^{SS}_{surréduction}$ (R, t, t') est un ensemble complet de EI-unificateurs en dehors de W.

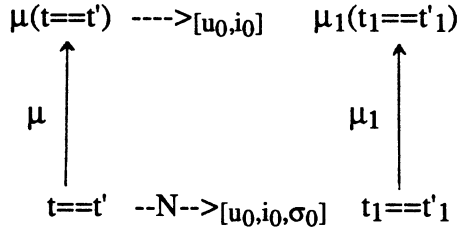
Preuve: Pour montrer la complétude de $Sol^{SS}_{surréduction}$ (R, t, t'), il suffit de considérer un EI-unificateur fermé et normalisé de t et t', soit μ, et de montrer qu'il existe une substitution θ dans $Sol^{SS}_{surréduction}$ (R, t, t') telle que $\theta \leq^{i}_{SP} \mu$ [V]. Les conditions (a) et (b) d'un ensemble complet de EI-unificateurs (définition3.6) sont clairement satisfaites pour toute substitution dans $Sol^{SS}_{surréduction}$ (R, t, t').

Par hypothèse (μ est un EI-unificateur fermé de t et t', et R est canonique), μ(t==t') se réduit par la relation $\rightarrow^*$ en une équation T == T, où T est la forme normale de $\mu(t)$ et $\mu(t')$. Soit L la longueur maximale des dérivations μ(t==t') $\rightarrow^*$ T==T. Montrons par induction sur L l'existence de θ.

Cas de base: L = 0, t et t' sont unifiables (via ρ) et non surréductibles. On pose $\theta = \rho$. θ est bien un élément de $Sol^{SS}_{surréduction}$ (R, t, t'). $\theta \leq^{i}_{SP} \mu$ [V] est immédiat.

Hypothèse d'induction : soit η un EI-unificateur fermé et normalisé de t et t' tel que la longueur maximale des dérivations de η(t==t') ---->* T==T est inférieure ou égale à L. Alors, il existe un EI-unificateur ξ dans $Sol^{SS}_{surréduction}$ (R, t, t') tel que $\xi \leq^{i}_{SP} \eta$ [V].

Supposons maintenant que la longueur maximale de μ(t==t') ---->* T==T soit L+1. D'après le lemme 3.1, t == t' est surréductible et en particulier à la position u_0 = SS(t==t'). SS étant uniforme, μ(t==t') est réductible en u_0. On a alors d'après le lemme3.1 la correspondance suivante.



avec μ(t==t') = $\mu_1\sigma_0$(t==t') et μ_1 est une substitution fermée et normalisée. D'après

l'hypothèse d'induction, il existe ξ dans $Sol^{SS}_{surréduction}$ (R, t_1, t'_1), c'est-à-dire qu'il existe une dérivation de surréduction avec la stratégie SS

$t_1 == t'_1$ --N-->$[u_1,i_1,\sigma_1]$ ... --N-->$[u_{n-1},i_{n-1},\sigma_{n-1}]$ $t_n == t'_n$

avec :

t_n et t'_n unifiables

$SS(t_n == t'_n)$ n'est pas défini

$\xi = \rho\sigma_{n-1}\sigma_{n-2}\ldots\sigma_1$

ρ est l'unificateur minimal de t_n et t'_n.

telle que $\xi \leq^{i}_{SP} \mu_1$ [vars(t_1) $\cup$ vars(t'_1)]. Autrement dit, il existe une substitution β telle que $\beta\xi \equiv^{i}_{SP}\mu_1$[vars($t_1$) $\cup$ vars(t'_1)]. D'autre part, on sait que $\mu = \mu_1\sigma_0$[vars(t) $\cup$ vars(t')]. D'où $\beta\xi\sigma_0 \equiv^{i}_{SP}\mu_1\sigma_0$ [vars(t) $\cup$ vars(t')] car $D(\sigma_0)$ est disjoint avec les domaines de toutes les autres substitutions et $I(\sigma_0)$ est inclus dans [vars(t_1) $\cup$ vars(t'_1)]. c.q.f.d

Remarque 3.2 : L'uniformité des stratégies n'est pas une condition nécessaire pour la complétude du calcul. En effet, soit à résoudre l'équation f(f(i,j), k) == s(s(0)) en considérant la spécification donnée dans le contre-exemple de la proposition 1 et la stratégie SS = left-most-outer-most. SS n'est pas uniforme car SS(f(f(i,j), k) == s(s(0))) = "1" mais σ (f(f(i,j), k) == s(s(0))), avec σ={i→0, j→0, k→0}, n'est pas réductible à la position "1". Par contre, $Sol^{SS}_{surréduction}$ (R, f(f(i,j), k), s(s(0))) = {{k→ s(y)}} est complet.

La décidabilité de l'uniformité est montrée dans le chapitre suivant.

3.2.3. Raffinement du calcul

Nous avons introduit dans le paragraphe précédent l'utilisation des stratégies de surréduction dans le calcul, par la relation de surréduction, d'un ensemble complet de EI-unificateurs. Un premier raffinement immédiat consiste à utiliser la relation de surréduction normale. Le résultat suivant est une conséquence du théorème 3.3.

Corollaire 3.1 : (complétude de $Sol^{SS\ normale}_{surréduction}$ (R, t, t'))

Soient SP=(Σ, E $\cup$ CH) une spécification telle que l'ensemble E soit défini par un système de réécriture canonique SR=(Σ, R), t et t' deux termes, W un ensemble de variables contenant V = vars(t) $\cup$ vars(t') et SS une stratégie de surréduction uniforme. Alors, l'ensemble $Sol^{SS\ normale}_{surréduction}$ (R, t, t') constitué de substitutions θ telles que il existe une dérivation de surréduction normale

$t! == t'!$ --NN-->$[u_0,i_0,\sigma_0]$ $t_1 == t'_1$ --NN-->$[u_1,i_1,\sigma_1]$ ... --NN-->$[u_{n-1},i_{n-1},\sigma_{n-1}]$ $t_n == t'_n$

avec :

t_n et t'_n unifiables

$u_i = SS(t_i == t'_i)$

$SS(t_n == t'_n)$ n'est pas défini

$\theta = \rho\sigma_{n-1}\sigma_{n-2}\ldots\sigma_0|_V$

ρ est l'unificateur minimal de t_n et t'_n.

est un ensemble complet de EI-unificateurs en dehors de W.

Preuve : Soit SS' une stratégie de surréduction telle que

SS'(t) = SS(t) si t est sous forme normale et

t/SS'(t) est réductible si t n'est pas sous forme normale.

On a alors

$Sol^{SS\ normale}_{surréduction}(R, t, t') = Sol^{SS'}_{surréduction}(R, t, t')$

SS' est par définition une stratégie de surréduction uniforme (puisque SS l'est). D'après le théorème3.3, $Sol^{SS'}_{surréduction}(R, t, t')$ est un ensemble complet de EI-unificateurs. D'où $Sol^{SS\ normale}_{surréduction}(R, t, t')$ est un ensemble complet de EI-unificateurs. c.q.f.d.

Un raffinement important que nous présentons maintenant consiste à utiliser des théorèmes inductifs (orientés) lors de la normalisation dans le calcul des substitutions par surréduction normale avec une stratégie uniforme. Nous appelons cette nouvelle relation "surréduction inductivement normale". Dans [Fri85], Fribourg a proposé cette amélioration dans le seul cas de la stratégie inner-most avec des conditions syntaxiques sur les spécifications, et en utilisant l'axiome de réflexivité autant de fois que possible. Dans ce qui suit nous généralisons ce résultat aux stratégies uniformes en imposant une forte restriction sur l'utilisation de l'axiome de réflexivité.

Avant de donner la définition de la relation de "surréduction inductivement normale", nous introduisons quelques hypothèses sur les spécifications utilisées. Nous supposons dans le reste de ce paragraphe que l'ensemble des axiomes des spécifications est partagé en trois parties disjointes E, EI et CH. Ainsi, une spécification SP sera présentée par SP = (Σ, E $\cup$ EI $\cup$ CH) où E et CH sont définis comme précédemment et EI[1] est un ensemble d'équations valides dans le E-modèle standard de la spécification SP' = (Σ, E $\cup$ CH). Nous supposons aussi que toute spécification SP = (Σ, E $\cup$ EI $\cup$ CH) est telle que

(1) le système de réécriture SR' = (Σ, R), constitué des règles de réécriture obtenues en orientant les équations dans E, est canonique, et

(2) le système de réécriture SR = (Σ, R $\cup$ RI), RI étant constitué des règles obtenues par simple orientation des équations dans EI, est nœthérien.

Remarquons que le système SR n'est pas forcément canonique, par contre il est canonique sur les termes fermés. Nous appelons forme normale inductive d'un terme, une forme normale de ce terme par rapport au système de réécriture SR.

---

[1]Il n'existe pas de lien direct entre l'ensemble EI dans SP = (Σ, E $\cup$ EI $\cup$ CH) et la notation "EI" dans *EI*-unificateur.

Définition 3.22 : (forme normale inductive)

Soit SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification telle que les équations dans EI soient valides dans E-MS_{SP}. Soit SR = (Σ, R $\cup$ RI) le système de réécriture tel que R respectivement RI soient constitués des règles de réécriture obtenues par orientation des équations dans E, respectivement dans EI. Nous appelons **forme normale inductive** d'un terme t, notée $\mathbf{t!^i}$, une forme normale de t par rapport au système de réécriture SR = (Σ, R $\cup$ RI).

Nous sommes prêt maintenent pour introduire la surréduction inductivement normale.

Définition 3.23: (surréduction inductivement normale)

Soit SR=(Σ, R $\cup$ RI) un système de réécriture noethérien avec R canonique. On définit la relation de surréduction inductivement normale entre deux termes t et t', notée t --INN--$>_{[u,i,\sigma]}$ t', par :

$$t \text{ --INN-->}_{[u,i,\sigma]} t' \Leftrightarrow t \text{ --N-->}_{[u,i,\sigma]} t'' \text{ et } t'=t''!^i$$

où la règle utilisée g_i ------> d_i est élément de R.

Nous rappelons que par $t!^i$ nous désignons la forme normale du terme t par rapport au système de réécriture SR = (Σ, R $\cup$ RI).

(INN dans --INN--> veut dire *Inductively Normal Narrowing*)

Exemple 3.8: Soient

$$R = \{\ 0 + x \text{ ------> } x$$
$$s(x) + y \text{ -----> } s(x+y)\ \} \text{ et}$$
$$RI = \{\ x + 0 \text{ -----> } x$$
$$x + s(y) \text{ -----> } s(x+y)\ \}$$

Considérons le terme 0+(x+0), on a :

$$0+(x+0) \text{ --N-->}_{[\varepsilon,1,\ldots]} x+0$$
$$(x+0)!^i = x, \text{ d'où}$$
$$0+(x+0) \text{ --INN-->}_{[\varepsilon,1,\ldots]} x$$

Définition 3.24 : ($\mathbf{Sol_{INN}}$ (R $\cup$ RI, t, t'))

Soit SR=(Σ, R $\cup$ RI) un système de réécriture noethérien avec R canonique. Soient t et t' deux termes, nous notons par Sol_{INN} (R $\cup$ RI, t, t') l'ensemble des substitutions θ telles que il existe une dérivation

$$t!^i == t'!^i \text{ --INN-->}_{[u_0,i_0,\sigma_0]} t_1 == t'_1 \text{ --INN--> } \ldots \text{ --INN-->}_{[u_{n-1},i_{n-1},\sigma_{n-1}]} t_n == t'_n$$

où les termes t_n et t'_n sont unifiables, $\theta = \rho\sigma_{n-1}\ldots\sigma_0|_V$ avec ρ l'unificateur minimal de t_n et t'_n et V = vars(t) $\cup$ vars(t').

Une substitution dans $Sol_{INN}(R \cup RI, t, t')$ est dite solution pour t et t' calculée par surréduction inductivement normale.

Exemple 3.9 : Considérons à nouveau le système de réécriture SR=(Σ, R ∪ RI) avec R et RI définit dans l'exemple 3.8. Soient les équations à résoudre x+ 0 == s(0) + 0 et x+ 0 == x. On a :

$$Sol_{INN}(R \cup RI, x+0, s(0) + 0) = \{\{x \rightarrow s(0)\}\}$$
$$Sol_{surréduction}(R, x+0, s(0) + 0) = \{\{x \rightarrow s(0)\}\}$$
$$Sol^{normale}_{surréduction}(R, x+0, s(0) + 0) = \{\{x \rightarrow s(0)\}\}$$

$$Sol_{INN}(R \cup RI, x+0, x) = \{id\}$$
$$Sol_{surréduction}(R, x+0, x) = \{\{x \rightarrow 0\}, \ldots, \{x \rightarrow s^i(0)\}, \ldots\}$$
$$Sol^{normale}_{surréduction}(R, x+0, x) = \{\{x \rightarrow 0\}, \ldots, \{x \rightarrow s^i(0)\}, \ldots\}$$

On remarquera que pour ces deux équations, il existe un nombre fini (ici égal à un) de dérivation de --INN-->, alors qu'il existe une infinité de dérivations de --N--> et --NN-->.

Nous montrons maintenant la complétude de l'ensemble $Sol_{INN}(R \cup RI, t, t')$. Pour ce faire, nous énonçons les deux lemmes suivants.

Lemme 3.3: (cohérence des solutions calculées par surréduction inductivement normale) Soient SP = (Σ, E ∪ EI ∪ CH) une spécification telle que l'ensemble E soit défini par un système de réécriture canonique SR'=(Σ, R) et que EI soit défini par un système de réécriture SR"=(Σ, RI) tel que le système SR=(Σ, R ∪ RI) soit noethérien. Alors, toute solution calculée pour t et t' par surréduction inductivement normale est un EI-unificateur de t et t'.

Preuve : Soit θ une solution calculée par surréduction inductivement normale pour les deux termes t et t'. Cela veut dire qu'il existe une dérivation

$t!^i == t'!^i$ --INN-->$_{[u_0,i_0,\sigma_0]}$ $t_1 == t'_1$ --INN--> ... --INN-->$_{[u_{n-1},i_{n-1},\sigma_{n-1}]}$ $t_n == t'_n$

où les termes t_n et t'_n sont unifiables, $\theta = \rho\sigma_{n-1}\ldots\sigma_0|_V$ avec ρ l'unificateur minimal de t_n et t'_n et V = vars(t) ∪ vars(t'). Il est immédiat que (1) θ(t == t')------->*$_{R \cup RI}$ T == T. D'autre part, θ est un EI-unificateur de t et t' ssi E-$MS_{(\Sigma, E \cup CH)} \models \theta(t) == \theta(t')$. Autrement dit, si pour toute instance fermée de θ, soit μθ, (2) μθ(t) == μθ(t') ---------->*$_R$ H == H. Or puisque RI représente des règles inductives, (2) est immédiat sachant (1). cqfd

Lemme 3.4 :(complétude de la surréduction inductivement normale)

Soient SP = (Σ, E ∪ EI ∪ CH) une spécification telle que l'ensemble E soit défini par un système de réécriture canonique SR'=(Σ, R) et que EI soit défini par un système de

réécriture SR"=(Σ, RI) tel que le système SR=(Σ, R ∪ RI) soit noethérien. Soit ψ un EI-unificateur fermé et normalisé de deux termes t et t'. Alors il existe une solution ϕ, calculée par surréduction inductivement normale, telle que $\phi \leq^{i}_{SP} \psi$ [vars(t) ∪ vars(t')].

Preuve : ψ étant un EI-unificateur fermé de t et t', on a ψ(t) == ψ(t') ------->*R T==T où T est la forme normale de ψ (t) et de ψ(t') par rapport à R. Par définitions de R et RI on a aussi $\psi(t!^{i}) == \psi(t'!^{i})$ ------->*R T==T. Soit L la longueur maximale de telles dérivations issues de $\psi(t!^{i}) == \psi(t'!^{i})$. Prouvons par induction sur L l'existence de ϕ.

cas de base : L = 0. Dans ce cas on a $\psi(t!^{i}) == \psi(t'!^{i})$ ------->0R T==T. C'est à dire $\psi(t!^{i}) = \psi(t'!^{i}) = T$. Ceci implique que $t!^{i}$ et $t'!^{i}$ sont unifiables. Posons ϕ égal à l'unificateur minimal de $t!^{i}$ et $t'!^{i}$. ϕ est donc un EI-unificateur de t et t'. De plus, $\phi \leq^{i}_{SP} \psi$ [vars(t) ∪ vars(t')].

Hypothèse d'induction : Soit α une substitution fermée et normalisée telle $\alpha(t!^{i}) == \alpha(t'!^{i})$ ----->*R T == T avec la longueur maximale d'une telle dérivation soit L. Alors il existe une solution calculée par surréduction inductivement normale θ telle que $\theta \leq^{i}_{SP} \alpha$ [vars(t) ∪ vars(t')].

Soit ψ une solution fermée et normalisée pour t == t' telle que la longueur maximale des dérivations de normalisation $\psi(t!^{i}) == \psi(t'!^{i})$ ------->*R T==T soit L+1. On sait par le théorème3.2 qu'il existe une dérivation de surréduction normale

$t!^{i}$==$t'!^{i}$ --NN-->$_{[u_0,i_0,\sigma_0]}$ t_1==t'_1 --NN-->$_{[u_1,i_1,\sigma_1]}$ ... --NN-->$_{[u_{n-1},i_{n-1},\sigma_{n-1}]}$ $t_n == t'_n$ telle que t_n et t'_n sont unifiables via l'unificateur minimal ρ et $\rho\sigma_{n-1}\ldots\sigma_0 \leq^{i}_{SP} \psi$ [vars($t!^{i}$) ∪ vars($t'!^{i}$)]. Par définition de $\leq^{i}_{SP}$ il existe une substitution fermée β telle que $\beta\rho\sigma_{n-1}\ldots\sigma_0 \equiv^{i}_{SP} \psi$[vars($t!^{i}$) ∪ vars($t'!^{i}$)]. Soit ψ' la forme normale de $\beta\rho\sigma_{n-1}\ldots\sigma_1$, ψ' est solution de t_1==t'_1. ψ' est donc aussi solution de $t_1!^{i}$==$t'_1!^{i}$ et la longueur maximale de $\psi(t_1!^{i}) == \psi(t_1'!^{i})$ ------->*R T==T est forcément inférieure ou égale à L (car sinon la dérivation $\psi(t!^{i}) == \psi(t'!^{i})$ ------->*R T==T n'aurait pas comme longueur maximale L+1). Par hypothèse d'induction, il existe alors une substitution calculée par surréduction inductivement normale θ telle que $\theta \leq^{i}_{SP} \psi'$ [vars(t_1) ∪ vars(t'_1)]. Posons $\phi = \theta\sigma_0$. ϕ est bien une solution calculée par surréduction inductivement normale et $\phi \leq^{i}_{SP} \psi$ [vars(t) ∪ vars(t')].cqfd

Des deux lemmes précédents, nous déduisons

Théorème 3.4 : Soient SP = (Σ, E ∪ EI ∪ CH) une spécification telle que l'ensemble E soit défini par un système de réécriture canonique SR'=(Σ, R) et que EI soit défini par un système de réécriture SR"=(Σ, RI) tel que le système SR=(Σ, R ∪ RI) soit noethérien. Soient t et t' deux termes et W un ensemble de variables contenant vars(t) ∪ vars(t'). Alors, Sol_{INN}(R ∪ RI, t, t') est un ensemble complet de EI-unificateurs en dehors de W.

Remarque 3.3 : Les éléments de $Sol_{INN}(R \cup RI, t, t')$ ne sont pas forcément des E-unificateurs.

Nous terminons ce paragraphe en montrant la complétude forte du calcul de EI-unificateurs par la surréduction inductivement normale. Autrement dit, nous raffinons le calcul du théorème 3.4 en introduisant des stratégies de surréduction.

Corollaire 3.2 : Soit SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification telle que l'ensemble E soit défini par un système de réécriture canonique SR'=(Σ, R) et que EI soit défini par un système de réécriture SR"=(Σ, RI) avec SR=(Σ, R $\cup$ RI) noethérien. Soient SS une stratégie de surréduction uniforme, t et t' deux termes et W un ensemble de variables contenant V=vars(t) $\cup$ vars(t'). Alors, l'ensemble $Sol^{SS}_{INN}(R \cup RI, t, t')$ constitué de substitution θ telle que il existe une dérivation de surréduction inductivement normale

$t!^{i}==t'!^{i}$ --INN-->$_{[u_0,i_0,\sigma_0]}$ $t_1==t'_1$ --INN-->$_{[u_1,i_1,\sigma_1]}$ ... --INN-->$_{[u_{n-1},i_{n-1},\sigma_{n-1}]}$ $t_n == t'_n$

avec: t_n et t'_n unifiables

$u_i = SS(t_i==t'_i)$

$SS(t_n == t'_n)$ n'est pas défini

$\theta = \rho\sigma_{n-1}\sigma_{n-2}\ldots\sigma_0|_V$

ρ est l'unificateur minimal de t_n et t'_n.

est un ensemble complet de EI-unificateurs en dehors de W.

Preuve : la preuve est similaire à celle du théorème 3.3. L'utilisation de règles dans RI, est justifiée par le théorème 3.4.

3.3. CONCLUSION

Nous avons présenté dans ce chapitre des algorithmes de résolution d'équations fondés sur la surréduction. Nous avons essentiellement étudié la surréduction avec stratégies et avons donné des conditions sur les stratégies pour assurer la complétude des calculs. L'utilisation de la surréduction dans un algorithme de résolution d'équations a été proposée pour la première fois par Fay dans [Fay79], où il étudie la surréduction normale. Hullot et Herold ont proposé la surréduction basique, respectivement dans [Hol80] et [Her86]. On dit qu'une dérivation de surréduction, t --N-->$_{[u_0,i_0,\sigma_0]}$ t_1 --N--> ... --N-->$_{[u_{n-1},i_{n-1},\sigma_{n-1}]}$ t_n, est basique si et seulement si chaque position u_j est choisie dans un ensemble de positions U_j défini par

-(selon Hullot)

$U_0 = Pos^*(t)$ et $U_{j+1} = (U_j - \{w \in U_j \mid u_j \leq w\}) \cup \{u_j.v \mid v \in Pos^*(d_{i_j})\}$

-(selon Herold)

$U_0 = Pos^*(t)$ et $U_{j+1} = (U_j - \{w \in U_j \mid u_j \leq w$ ou $(w < u_j$ et $u_j \neq ww')\}) \cup \{u_j.v \mid v \in Pos^*(d_{i_j})\}$, la condition ($w < u_j$ et $u_j \neq ww'$) veut dire que la position w est "à gauche" de

la position u_j dans le terme t_j (La définition duale "à droite" est possible).
L'ensemble des solutions calculées par les dérivations de surréduction basique constitue un ensemble complet de EI-unificateurs. Les deux relations de surréduction normale et de surréduction basique améliorent l'algorithme de résolution par surréduction (théorème3.1) en évitant certains calculs redondants. Les dérivations redondantes évitées par chacune de ces deux relations ne sont pas comparables en général. La combinaison de la surréduction basique et de la surréduction normale a été étudiée par Nutt et al. [NRS87] et Réty [Rét88]. La combinaison de la surréduction basique avec les stratégies n'est pas complète.

Dinkbas et Van Hentenryck [DH87] et Reddy [Red86] ont informellement introduit la surréduction paresseuse "Lazy Narrowing". Une formalisation de la surréduction paresseuse peut être consultée par exemple dans [MR89] ou [Pad88]. You a proposé dans [You88] les dérivations appelées "outer narrowing". C'est une sorte de surréduction paresseuse pour laquelle You a montré la complétude dans le cas des systèmes de réécriture à base de constructeurs.

L'introduction des stratégies de surréduction a été proposée pour la première fois par Fribourg. Dans [Fri85], Fribourg a donné des conditions syntaxiques assurant la complétude de la seule stratégie "innermost". Dans [Pad88], Padawitz a donné des conditions analogues aux nôtres pour montrer la complétude de la surréduction en présence des stratégies. La définition de stratégie de surréduction utilisée par Padawitz est moins générale que celle utilisée dans ce mémoire. Il impose de plus sur toute stratégie SS la condition : $\forall\, t \in T(\Sigma, X)$, $\forall\, \sigma \in SUB$, $\sigma(t)/SS(\sigma(t)) = \sigma(t/SS(t))$ (cf [Pad88], page 195). Une telle définition écarte d'emblée les stratégies classiques telles que la stratégie "inner-most" ou la stratégie "outer-most" (voir [Pad88], exemple8.4.1, page 199). Padawitz montre la complétude du calcul par surréduction avec de telles stratégies lorsque la propriété dite "locally redex stable[1]" est vérifiée.

Bosco et al [BGM87] ont proposé une transformation de termes et d'équations en clauses de Horn, la surréduction est ensuite "simulée" par la SLD-résolution sur les clauses obtenues, augmentées de la réflexivité. C'est une idée intéressantes qui consiste intuitivement à "déplier" les termes, par exemple le terme f(f(i, j), k) est transformé en "*f(z1, k) == z0, f(i, j) == z1*", une règle de réécriture g -----> d est transformée en "*g = z ⇐ transformée(d)*" et un but f(f(i, j), k) == 0 est transformé en "*f(z1, k) == z0, f(i, j) == z1, 0 == z0* ". Cependant la surréduction d'un terme, soit f(f(i, j), k), à une position de surréduction, par exemple ε, n'est pas toujours équivalente à la résolution de l'atome correspondant au sous-terme considéré, ici *f(z1, k) == z0*. L'utilisation des stratégies de

[1]Cette propriété porte sur les spécifications et non sur les stratégies. Elle est très proche de l'uniformité des spécifications que nous définissons dans le chapitre 4.

résolution ne permet donc pas d'éliminer autant de calculs redondants (ou inutiles) que l'utilisation des stratégies de surréduction.

Nous pensons enfin que l'utilisation des stratégies uniformes de surréduction en présence de la surréduction conditionnelle [Pad88],[Höl89], [Kap87], [Hus85] [Rus89] ou de la surréduction équationnelle [JKK82] donne toujours des algorithmes complets. Ces résultats ne doivent pas posés de difficultés particulières.

Chapitre 4

Spécifications Uniformes

Nous avons vu dans le chapitre 3 (cf. théorème 3.3) que l'uniformité d'une stratégie de surréduction est une condition suffisante pour garantir la complétude de la EI-unification par surréduction. Dans ce chapitre, nous définissons une classe de spécifications dites *uniformes*. En présence de ces spécifications, toute stratégie de surréduction est, par définition, uniforme. Ceci a l'avantage d'induire une totale liberté de choix de la position de surréduction lors des calculs de EI-unificateurs. Nous montrons la décidabilité de l'uniformité des spécifications et donnons ensuite des conditions suffisantes pour qu'une spécification soit uniforme. Ces conditions nous permettent en particulier de retrouver un résultat de You sur le E-filtrage [You88]. Enfin, nous proposons et prouvons un algorithme de transformation de spécifications basées sur les constructeurs, satisfaisant le principe de définition de Huet et Hullot [HH80], en des spécifications satisfaisant les conditions que nous proposons.

4.1. SPECIFICATIONS UNIFORMES

Nous avons vu dans le §3.2.2 que l'uniformité des stratégies est une condition suffisante pour établir la complétude de la EI-unification en utilisant des stratégies de surréduction. Il s'agit maintenant de voir dans quelle mesure les résultats précédents sont applicables. Pour cela nous allons considérer une classe particulière de spécifications. C'est ce que nous appelons la classe des spécifications uniformes.

Dans le reste de ce chapitre nous ne considérons que des spécifications purement équationnelles, de la forme SP =(Σ, E) avec Σ=(S, Ω), au lieu des spécifications considérées jusqu'à présent, de la forme SP =(Σ, E ∪ CH), où l'on trouve des équations et des clauses de Horn dont la tête n'est pas une équation. Cette supposition est suggérée, d'une part par le fait que la congruence $=\!\!\!\downarrow_{SP}$ engendrée par la spécification est entièrement définie par l'ensemble des équations E, et d'autre part par le fait que nous ne considérons dans ce chapitre, que des problèmes équationnels, plus précisément la résolution d'équations. Tous les résultats établis dans ce chapitre restent valables pour les spécifications dont la présentation est de la forme SP =(Σ, E ∪ CH).

Définition 4.1 : (spécification uniforme)

Soit SP = (Σ, E) une spécification telle que le système de réécriture associé à E, SR=(Σ, R), soit canonique. La spécification SP est dite uniforme si et seulement si toute stratégie de surréduction est uniforme.

Rappelons qu'une stratégie de surréduction SS est dite uniforme (cf. définition 3.21) si et seulement si quel que soit un terme surréductible t, pour toute substitution fermée et normalisée μ, $\mu(t)$ est réductible à la position SS(t). Il est clair que dans le cas d'une spécification uniforme, les résultats de complétude des différents calculs par surréduction du §3.2 restent valides.

L'uniformité d'une spécification peut être considérée comme une condition "sémantique" sur les spécifications. Nous montrons maintenant que cette condition est vérifiable de manière syntaxique. Nous introduisons d'abord quelques résultats préliminaires.

Définition 4.2 : (terme uniforme, ensemble uniforme)

Soient SP = (Σ, E) une spécification telle que le système de réécriture associé à E, SR=(Σ, R), soit canonique. Un terme t est dit uniforme à la position ε (ou uniforme) si et seulement si, si t est surréductible à la position ε alors pour toute substitution fermée et normalisée μ, $\mu(t)$ est réductible à la position ε. Un ensemble de termes A est dit uniforme si tous ses termes sont uniformes.

Proposition 4.1 : Soient SP = (Σ, E) une spécification telle que le système de réécriture associé à E, SR=(Σ, R), soit canonique. SP est uniforme si et seulement si pour tout ensemble de variables X, $T(\Sigma, X)$ est uniforme.

Preuve : SP est uniforme $\Leftrightarrow \forall$ SS : stratégie de surréduction, SS est uniforme $\Leftrightarrow$ (1) $\forall$ SS : stratégie de surréduction, $\forall$ X, $\forall$ t $\in$ $T(\Sigma, X)$, SS(t) est définie $\Rightarrow \forall \mu \in$ SUBFN, $\mu(t)$ est réductible à la position SS(t) .

Montrons l'équivalence : (1) $\Leftrightarrow \forall$ X, $\forall$ t $\in$ $T(\Sigma, X)$, t est uniforme.

($\Leftarrow$) Soient SS une stratégie de surréduction et t un terme. Si SS(t) est défini, cela veut dire que t/SS(t) est surréductible à la position ε. t/SS(t) étant un terme, par hypothèse t/SS(t) est uniforme. D'où (1).

($\Rightarrow$) par contraposée. Soient X un ensemble de variables et t un terme dans $T(\Sigma, X)$ tels que t ne soit pas uniforme. C'est-à-dire que t est surréductible à la position ε, et il existe une substitution ϕ dans SUBFN telle que $\phi(t)$ n'est pas réductible à la position ε. Soit SS une stratégie de surréduction telle que SS(t) = ε (dans ce cas, ce serait une stratégie *outermost*) $\phi(t)$ n'est pas réductible à la position SS(t). D'où la négation de (1). cqfd.

Compte tenu de la proposition ci-dessus, pour vérifier l'uniformité d'une spécification, nous introduisons deux lemmes. Le premier (lemme 4.1) concerne la décidabilité de l'uniformité d'un seul terme et le second (lemme 4.2) exhibe un ensemble fini de termes, dit ensemble de tests pour l'uniformité, tel que l'uniformité de cet ensemble est équivalente à celle de $T(\Sigma, X)$, pour tout X.

Lemme 4.1 : Soit $SP = (\Sigma, E)$ une spécification telle que le système de réécriture associé à E, $SR=(\Sigma, R)$, soit canonique. Soient X un ensemble de variables et t un terme ($t \in T(\Sigma, X)$). Vérifier que t est uniforme est décidable.

Preuve : Il suffit pour cela de pouvoir caractériser de façon finie les termes fermés normalisés (et les substitutions fermées normalisées). Cette méthode a été utilisée de manière explicite ou non dans les preuves de la propriété de réductibilité inductive (voir par exemple [Pla85] [KNZ87] [Com88] [JK86]). Nous utilisons dans cette preuve la méthode de Kounalis parue récemment dans [Kou90] où pour un système de réécriture R, il associe un ensemble de termes S(R) ayant les propriétés suivantes :

- S(R) est fini
- Pour tout terme t fermé et normalisé, il existe un terme unique t' dans S(R) et une substitution fermée σ tels que $\sigma(t') = t$.
- Pour tout terme t' dans S(R), il existe un terme fermé normalisé t et une substitution σ tels que $\sigma(t') = t$
- Tous les termes non fermés de S(R) sont transnormaux (un terme t est dit transnormal ssi il admet une infinité d'instances sous forme normale).
- Tous les termes non fermés de S(R) sont de hauteur supérieure ou égale à H, H étant la hauteur du système de réécriture R (cf. définition 4.3 ci-dessous).

Un algorithme construisant un tel ensemble peut être consulté dans [Kou90]. A partir de S(R), il est possible de décrire de façon finie toutes les substitutions fermées et normalisées. La vérification de l'uniformité d'un terme est dès lors immédiate.cqfd

Nous introduisons maintenant quelques définitions utiles pour exprimer l'ensemble de tests pour l'uniformité d'une spécification.

Définition 4.3 : (**hauteur** d'un terme, d'une règle de réécriture et d'un système de réécriture)

Soient $SR=(\Sigma, R)$ un système de réécriture et t un terme ($t \in T(\Sigma, X)$). On définit la hauteur d'un terme par **hauteur (t)** = $\max_{u \in Pos(t)}(|u|)$, où |u| désigne la longueur de la chaîne u. La hauteur d'une règle de réécriture est la hauteur de son membre gauche; hauteur(g----->d) = hauteur(g). On définit la hauteur d'un système de réécriture SR, noté H_{SR} (ou simplement H), par **H** = $\max_{r \in R}$ (hauteur(r)).

Définition 4.4 : ($T(\Sigma, X)_{\leq h}$)

Soient Σ une signature, h un entier naturel et X un ensemble de variables. Nous désignons par $T(\Sigma, X)_{\leq h}$ l'ensemble des termes de hauteur inférieure ou égale à h.

Proposition 4.2 : Soient $\Sigma = (S, \Omega)$ une signature algébrique et h un entier naturel. Il existe un entier naturel, noté $n(\Sigma, h)$, tel que pour tout ensembles de variables X et Y avec card(X) quelconque et $card(Y) \geq n(\Sigma, h)$. Alors, $\forall\, t \in T(\Sigma, X), \exists\, t' \in T(\Sigma, Y)_{\leq h}, \exists\, \sigma \in SUB \mid \sigma(t') = t$.

Preuve : Il suffit de prendre $n(\Sigma, h) = k^h$ avec $k = \max_{f_{us} \in \Omega}(|u|)$. Informellement, k représente le plus grand nombre d'arguments qu'un opérateur de la signature puisse avoir. k^h représente le plus grand nombre de variables pouvant apparaître dans un terme de hauteur inférieure ou égale à h. Soient X et Y deux ensembles dénombrables de variables avec $card(Y) \geq k^h$. Soit t un terme de $T(\Sigma, X)$, Cherchons t' dans $T(\Sigma, Y)_{\leq h}$ et une substitution σ tels que $\sigma(t') = t$. On distingue deux cas pour t.

1) $hauteur(t) \leq h$. Dans ce cas $card(vars(t)) \leq k^h$. Il existe donc une substitution σ qui est un renommage et t' dans $T(\Sigma, Y)$ avec $\sigma(t') = t$.

2) $hauteur(t) \geq h + 1$. Soit t" un terme dans $T(\Sigma, X)$ obtenu à partir de t en élaguant le terme t à la hauteur h. C'est-à-dire : $\forall\, u \in Pos^*(t)$, $[t(u) = t"(u)]$[1] ou $[(|u|=h)$ et $(t"/u = x_u \in X)$ et $(t/u \notin X)]$ avec x_u n'apparaissant qu'une seule fois dans t". Soit la substitution $\rho = \{x_u \rightarrow t/u \mid x_u \notin vars(t)$ et $|u| = h\}$. Il est clair que $\rho(t") = t$. D'autre part, on a $hauteur(t") = h$ par construction. D'après le premier cas, il existe un terme t' dans $T(\Sigma, Y)_{\leq h}$ et un renommage σ tel que $\sigma(t') = t"$. Alors $\rho\sigma(t') = \rho(t") = t$. cqfd.

Nous définissons ci-dessous un ensemble fini de termes qui servira dans la vérification de l'uniformité des spécifications.

Définition 4.5 : (ensemble de tests pour l'uniformité)

Soit SP = (Σ, E) une spécification telle que le système de réécriture associé à E, SR=(Σ, R), soit canonique. Un ensemble de tests pour l'uniformité est tout ensemble $T(\Sigma,X)_{\leq H+1}$ où H est la hauteur du système de réécriture SR, X est un ensemble de variables de cardinalité finie mais supérieur à $n(\Sigma,H+1)$ (c'est-à-dire $n(\Sigma, H+1) \leq card(X) < \omega$).

---

[1]La notation t(u) désigne le symbole d'opération à la racine du terme t/u.

Lemme 4.2 : Soit SP = (Σ, E) une spécification telle que le système de réécriture associé à E, SR=(Σ, R), soit canonique. Soit A un ensemble de tests pour l'uniformité. A est uniforme si et seulement si T(Σ, X) est uniforme pour tout ensemble de variables X.

Preuve :

($\Leftarrow$) $\forall$ X, T(Σ, X) uniforme $\Rightarrow$ A uniforme. évident ($\exists$ Y, A $\subset$ T(Σ, Y)).

($\Rightarrow$) par contraposée. $\exists$ Z, T(Σ, Z) non uniforme $\Rightarrow$ A non uniforme. T(Σ, Z) étant non uniforme, il existe donc un terme t dans T(Σ, Z) tel que t n'est pas uniforme. Alors : (i) t est surréductible à la position ε, soient g_i----->d_i et σ la règle et la substitution utilisées pour la surréduction de t : t--N-->$_{[\varepsilon,i,\sigma]}$ $\sigma(d_i)$, et (ii) il existe une substitution fermée et normalisée μ telle que $\mu(t)$ n'est pas réductible à la position ε. D'après la proposition 4.2, il existe un terme t' dans A et une substitution ϕ telle que $t = \phi(t')$. On déduit alors que t' est surréductible à la position ε car $\sigma(t) = \sigma(\phi(t')) = \sigma(g_i)$ et $\mu(\phi(t'))$ n'est pas réductible à la position ε. Or $\mu\phi$ n'est pas forcement normalisée. Soit ψ la forme normale de $\mu\phi$, du fait que $\mu(\phi(t'))$ n'est pas réductible à la position ε, on déduit que $\psi(t')$ n'est pas réductible à la position ε. En effet, quelle que soit une variable x élément du domaine D(ϕ), x se trouve à la hauteur H+1. D'où la normalisation de $\mu\phi(x)$ n'affecte en rien la possibilité de réécriture du terme $\psi(t)$ puisque H est la hauteur du système de réécriture considéré. A est donc non uniforme. cqfd.

Nous pouvons maintenant énoncer le

Théorème 4.1 : Soient SP = (Σ, E) une spécification telle que le système de réécriture associé à E, SR=(Σ, R), soit canonique. Vérifier que SP est uniforme est décidable.

Preuve : se déduit des lemmes 4.1 et 4.2, ainsi que de la proposition 4.1.

Remarque 4.1 : Si l'on définit la notion de "dérivation de surréduction uniforme issue de t" comme étant toute dérivation de surréduction issue de t :

$$t \text{ --N-->}_{[u_0,i_0,\sigma_0]} t_1 \text{ --N--> } \ldots \text{ --N-->}_{[u_{n-1},i_{n-1},\sigma_{n-1}]} t_n$$

telle que, pour tout i=0...n-1, t_i/u_i est uniforme (avec $t_0 = t$). Alors, le théorème 3.3 et le corollaire3.2 du chapitre 3 restent valables lorsqu'on considère les "dérivations de surréduction uniformes" issues de l'équation à résoudre. C'est-à-dire qu'au lieu d'exiger l'uniformité de la stratégie de surréduction utilisée, il suffit, en fait, pour assurer la complétude des ensembles de EI-unificateurs cherchés, que les dérivations de surréduction issues de l'équation à résoudre avec la stratégie SS

$$t == t' \text{ --N-->}_{[u_0,i_0,\sigma_0]} t_1 == t'_1 \text{ --N--> } \ldots \text{ --N-->}_{[u_{n-1},i_{n-1},\sigma_{n-1}]} t_n == t'_n$$

avec $u_i = SS(t_i == t'_i)$

soient uniformes (c'est-à-dire $t_i == t'_i / u_0$ est uniforme). Grâce à cette remarque et le test de

l'uniformité d'un terme, il est facile de concevoir des stratégies de surréduction qui, à chaque pas de surréduction, choisissent *à la volée* une position de surréduction pertinente pour la complétude du calcul.

4.2. CONDITIONS POUR L'UNIFORMITE DES SPECIFICATIONS

Nous introduisons dans ce paragraphe des conditions syntaxiques sur les spécifications qui garantissent l'uniformité de toute stratégie de surréduction possible. Afin de donner ces conditions, nous définissons d'abord la notion de *sous-unification.*

Définition 4.6 : (sous-unification)

Deux termes de même sorte sont dits sous-unifiables ssi il existe une position u dans $Pos(t) \cap Pos(t')$ telle que

(i) t/u et t'/u sont unifiables (via σ_u)

(ii) Pour tout préfixe strict w de u ($w < u$), racine(t/w) = racine(t'/w).

Si σ_u n'est pas un renommage et $D(\sigma_u)$ n'est pas vide, on dira que t et t' sont **strictement sous-unifiables.**

Exemple 4.1 : Les termes f(0,0) et f(x, s(y)) sont sous-unifiables à la position "1", $\sigma_{"1"} = \{x \rightarrow 0\}$.

Les termes f(0,0) et f(x, s(y)) sont strictement sous-unifiables puisque $\sigma_{"1"} = \{x \rightarrow 0\}$.

Les termes f(0, s(x)) et f(s(x), s(y)) sont sous-unifiables à la position"2.1" mais ne sont pas strictement sous-unifiables parce que $\sigma_{"2.1"} = \{x \rightarrow y\}$ est un renommage.

Notons que deux termes sous-unifiables peuvent l'être en plusieurs positions différentes. Les termes considérés étant tous finis, décider de la sous-unification est immédiat.

Les conditions syntaxiques sur les spécifications que nous définissons sont des cas particuliers des spécifications à base de constructeurs [HH80]. De manière générale, une spécification SP=(Σ, E) avec Σ = (S, Ω) est dite à base de constructeurs si l'ensemble des opérateurs Ω est scindé en deux ensembles disjoints C et D. Les éléments de C sont dits constructeurs et ceux de D opérateurs définis. On notera par T(C,X) l'ensemble des termes construits à partir des opérateurs dans C et les variables dans X. T(C) désignera l'ensemble T(C, Ø).

Définition 4.7 : (Conditions Syntaxiques pour l'Uniformité (CSU))

Une spécification SP=(Σ, E) avec Σ = (S, Ω) vérifie les conditions **CSU** ssi

(1) Ω est scindé en deux ensembles C et D. $\Omega = C \cup D$ et $C \cap D = \emptyset$.

(2) Les opérateurs dans D sont complètement définis[1]

(3) Les membres gauches des équations dans E sont de la forme $f(t_1,...,t_n)$ où le symbole f est élément de D et les t_i sont éléments de T(C, X).

(4) Les membres gauches des équations dans E sont deux à deux non-strictement sous-unifiables.

(5) Le système de réécriture (Σ, R) obtenu à partir de E en orientant les équations de gauche à droite est un système de réécriture canonique.

Exemple 4.2 : Soient SP=(Σ, E) et SP'=(Σ, E') les deux spécifications suivantes dont nous ne donnons que les équations.

E = { 1 : f(0, 0) == 0
2 : f(s(x), 0) == s(0)
3 : f(x, s(y)) == s(s(0)) }

E' = { 1 : f(0,0) == 0
2 : f(s(x), 0) == s(0)
3 : f(0, s(y)) == s(s(0))
4 : f(s(x), s(y)) == s(s(0)) }

La spécification SP ne satisfait pas la condition (4) dans CSU. SP' vérifie par contre les conditions CSU. Notons que ces deux spécifications définissent le même E-modèle standard.

La vérification des conditions CSU est indécidable dans le cas général à cause de la condition (5). Il existe toute fois des outils qui permettent de vérifier cette condition dans certains cas, voir par exemple [Der85]. Les conditions (1), (3) et (4) sont facilement vérifiables. Pour la vérification de la condition (2), voir par exemple [NW83][Der87] [Thi84], [KNZ87] [Com86].

La classe de spécifications satisfaisant CSU peut paraître trop restrictive au premier abord. En fait, nous montrons dans le paragraphe suivant (§4.3) que cela n'est pas le cas en exhibant un algorithme de transformation de spécifications à base de constructeurs en des spécifications vérifiant les conditions CSU. A notre connaissance, tous les langages actuels utilisant des stratégies de surréduction ne vont pas au delà des spécifications avec constructeurs, par exemple [Hus85][Fri85][BE86][BBLM86] [MR89].

---

[1]Un opérateur f : $s_1...s_n$->s dans D est complètement défini ssi $\forall t_i \in T_{s_i}(C)$ i =1...n, $\exists t \in T_s(C)$ | $f(t_1,...,t_n) =_{SP} t$.

Comme nous l'avons annoncé, l'intérêt des spécifications satisfaisant les conditions CSU est de garantir l'uniformité des stratégies de surréduction, et par suite la "complétude forte" des algorithmes de surréduction.

Lemme 4.3 : Soit SP=(Σ, E) une spécification vérifiant les conditions CSU. Alors SP est uniforme.

Preuve : Par l'absurde. Soit SR =(Σ, R) le système de réécriture défini par SP. Soit t un terme. Supposons que :

(1) t est surréductible à la position ε. C'est-à-dire qu'il existe k et σ_k tels que $\sigma_k(t) = \sigma_k(l_k)$ avec l_k----->r_k dans R.

(2) il existe une substitution fermée normalisée μ telle que $\mu(t)$ n'est pas réductible à la position ε.

De (1), (2) et des conditions (2) et (3) de CSU, nous déduisons que le terme t contient au moins un opérateur défini (appartenant à D) ailleurs qu'à la racine de t.

Soit SI ={ i | l_i----->$r_i \in$ R et l_i ne filtre[1] pas $\mu(t)$ à cause d'un "clash" entre un opérateur défini dans t et un constructeur dans l_i}.

SI n'est pas vide (SI $\neq \emptyset$), D'après (2) et les conditions (2) et (3) de CSU.

Soit U ={u $\in$ Pos(t) | la racine de t/u est un opérateur défini appartenant à D}.

U $\neq \emptyset$, conséquence de SI $\neq \emptyset$.

Nous associons maintenant à U l'ensemble U' défini comme suit:

U' ={u' $\in$ Pos(l_k), k est le même que dans (1) telle que l_k/u' est une variable, soit x, et $\sigma_k(x)$ contient t/u avec u in U}.

Remarquons que u'$\leq$ u.

U' $\neq \emptyset$, D'après U $\neq \emptyset$ et (1).

(3) Soit j $\in$ SI, soit u $\in$ U la position où le clash a eu lieu. C'est-à-dire la racine de l_j/u est un constructeur et la racine de t/u est un opérateur défini.

Soit u' dans U' la position correspondante à u dans U.

l_j/u' ne filtre pas $\mu(t)$/u', D'après (3) et u' $\leq$ u.

l_k/u' est une variable, D'après la définition de U'.

l_j/u' n'est pas une variable,D'après (3).

Nous déduisons alors que l_k/u' et l_j/u' sont unifiables et leur unificateur minimal n'est ni un renommage ni l'identité. De plus, pour toute position w inférieure à u' (w < u'), l_k/w et l_j/w ont la même racine. Ceci contredit la condition (4) de CSU. cqfd

On a donc

---

[1]Un terme t filtre un autre terme s si et seulement si il existe une substitution σ telle que $\sigma(t) = s$

Théorème 4.2 : Soient SP=(Σ, E) une spécification vérifiant les conditions CSU, SR=(Σ, R) le système de réécriture défini par E et SS une stratégie de surréduction. Soient t et t' deux termes et W un ensemble de variables contenant vars(t) ∪ vars(t'). Alors, $Sol^{SS}_{surréduction}$ (R, t, t') est un ensemble complet de EI-unificateurs en dehors de W.

Preuve : se déduit du lemme 4.3 et du théorème 3.3

Le résultat suivant exprime une condition supplémentaire sur les spécifications permettant le calcul d'un ensemble minimal de EI-unificateurs.

Théorème 4.3 : Soient SP=(Σ, E) une spécification vérifiant les conditions CSU, SR=(Σ, R) le système de réécriture défini par E et SS une stratégie de surréduction. Soient t et t' deux termes et W un ensemble de variables contenant vars(t) ∪ vars(t'). Si de plus, tous les membres gauches des règles dans R ne sont pas unifiables deux à deux, alors $Sol^{SS}_{surréduction}$ (R, t, t') est un ensemble minimal de EI-unificateurs en dehors de W.

Preuve : $Sol^{SS}_{surréduction}$ (R, t, t') est clairement un ensemble complet de EI-unificateurs (voir théorème ci-dessus). Prouvons maintenant, par l'absurde, que sous l'hypothèse supplémentaire, toutes les substitutions de $Sol^{SS}_{surréduction}$ (R, t, t') sont incomparables.

Soient deux substitutions s et s' dans $Sol^{SS}_{surréduction}$ (R, t, t'). Supposons $s \leq^{i}_{SP} s'$ [V] (V étant l'ensemble de variables de t et t'), et $s \neq s'$.

Puisque $s \neq s'$, il existe deux dérivations différentes de surréduction issues de l'équation $t_0 == t'_0$,

$$t_0 == t'_0 \text{ --N->}_{[u_0, i_0, \sigma_0]} t_1 == t'_1 \text{ --N-> ... --N->}_{[u_{n-1}, i_{n-1}, \sigma_{n-1}]} t_n == t'_n$$
$$t_0 == t'_0 \text{ --N->}_{[u'_0, i'_0, \sigma'_0]} t_1 == t'_1 \text{ --N-> ... --N->}_{[u'_{m-1}, i'_{m-1}, \sigma'_{m-1}]} t_m == t'_m$$

Avec $s = \sigma\theta_n$ et $s' = \sigma'\theta'_m$. Les deux dérivations ci-dessus peuvent avoir un même début. Nous supposons, sans perte de généralité, que les deux dérivations sont différentes dès la première dérivation de surréduction, c'est-à-dire $i_0 \neq i'_0$ (notons que $u_0 = u'_0 = SS(t_0 == t'_0)$). Alors, $s'(t_0 == t'_0)$ se réduit à la position u_0 en utilisant la règle $l_{i'_0} \text{----->} r_{i'_0}$. Il existe donc une substitution f_1 telle que (A) $f_1(l_{i'_0}) = s'(t_0 == t'_0)/u_0$. Or, puisque $s \leq^{i}_{SP} s'$, il existe une substitution ρ (normalisée) telle que $\rho s \equiv^{i}_{SP} s'$. s et s' étant normalisées (coséquence des conditions CSU et en particulier de la condition (3)), alors $s'(t_0 == t'_0)$ qui est égal à $\rho s(t_0 == t'_0)$ se réduit aussi à la position u_0 en utilisant la règle $l_{i_0} \text{----->} r_{i_0}$, c'est-à-dire qu'il existe une substitution f_2 telle que (B) $s'(t_0 == t'_0)/u_0 = f_2(l_{i_0})$. Enfin, de (A) et (B) nous concluons que l_{i_0} et $l_{i'_0}$, avec $i_0 \neq i'_0$, sont unifiables ($vars(l_{i_0}) \cap vars(l_{i'_0}) = \emptyset$). Ceci contredit la condition supplémentaire. cqfd.

Dans [You88], en proposant les dérivations "outer narrowing", You a donné un résultat concernant le E(I)-filtrage dans le cadre des systèmes de réécriture à base de constructeurs. On dit qu'un terme t E-filtre (respectivement EI-filtre) un terme t' ssi il existe une substitutions σ telle que $\sigma(t) =_{SP} t'$ (respectivement $\sigma(t) =^{i}_{SP} t'$). σ est alors appelé E-filtre (respectivement EI-filtre). Comme dans le cas de la E(I)-unification, on définit les notions d'ensemble complet de E(I)-filtres et d'ensemble minimal de E(I)-filtres. Nous obtenons un résultat analogue à celui de You comme corollaire du théorème 4.3 ci-dessus.

Corollaire 4.1 : Soient SP=(Σ, E) une spécification vérifiant les conditions CSU[1], SR=(Σ, R) le système de réécriture défini par E et SS une stratégie de surréduction. Soient t et t' deux termes avec t' dans T(C, X) et vars (t) $\cap$ vars(t') = $\emptyset$ et W un ensemble de variables contenant vars(t) $\cup$ vars(t'). Si de plus, tous les membres gauches des règles dans R ne sont pas unifiables deux à deux, alors $Sol^{SS}_{surréduction}$ (R, t, t') est un ensemble minimal de EI-filtres en dehors de W.

Preuve : se déduit immédiatement du théorème 4.3 ci-dessus puisque t' ne contient aucune position de surréduction.

4.3. TRANSFORMATION DE SPECIFICATIONS

Dans ce paragraphe, nous donnons un algorithme de transformation de spécifications satisfaisant le principe de Huet-Hullot [HH80] en des spécifications satisfaisant les conditions CSU. Notre but dans ce paragraphe est de montrer que la condition (4) dans CSU peut être ignorée dans une première étape de spécification comme c'est le cas des spécifications dans les langages tels que SLOG, LPG ou RAP. Rappelons d'abord le principe de définition donné par Huet et Hullot.

Définition 4.8 : **(principe de définition de Huet et Hullot)**

Une spécification SP=(Σ, E) avec Σ=(S, Ω) satisfait le principe de définition de Huet et Hullot ssi

(1) Ω est scindé en deux ensembles disjoints C et D.

(2) $\forall\, t \in T(\Sigma), \exists\, t' \in T(C) \mid t =_E t'$.

(3) $\forall\, t, t' \in T(C), t =_E t' \Leftrightarrow t = t'$

Nous ajoutons la condition supplémentaire suivante

(4) Le système de réécriture défini par E est canonique

---

[1]On peut ne pas considérer la condition (4) de CSU mais en utilisant une stratégie uniforme, comme par exemple la stratégie "innermost".

Dans [HH80], des conditions suffisantes pour vérifier les propriétés (2) et (3) du principe de définition ont été données. Ces conditions sont basées sur la notion "d'ensemble complet pour C". Intuitivement, un ensemble de k-uplets $P = \{P_i \mid P_i = (P_1^i, ..., P_k^i), i = 1..q$ avec $P_l^m \in T(\Sigma, X)\}$ est un ensemble complet pour C si tous les k-uplets P_i sont linéaires et pour tout k-uplet $(t_1,...,t_k)$ avec les t_i fermés et normalisés, il existe m avec $1 \leq m \leq q$ et une substitution σ tels que pour tout $j = 1...k$, $\sigma(P_j^m) = t_j$. Dans ce qui suit, nous utilisons une définition légèrement différente de celle de donnée dans [HH80] où la condition de linéarité a été levée.

Définition 4.9 : (ensemble complet pour C)

Un ensemble de k-uplets $P = \{P_i \mid P_i = (P_1^i, ..., P_k^i), i = 1...q$ avec $P_l^m \in T(\Sigma, X)\}$ est un ensemble complet pour C si et seulement si pour tout k-uplet $(t_1,...,t_k)$ avec les t_i fermés et normalisés, il existe m avec $1 \leq m \leq q$ et une substitution σ tels que pour tout $j = 1...k$, $\sigma(P_j^m) = t_j$.

Le lemme ci-dessous montre une condition nécessaire sur les spécifications satisfaisant les conditions de la définition 4.8.

Lemme 4.4 : Soit SP = (S, C∪D, E) une spécification satisfaisant les conditions de la définition 4.8. Alors, il existe un sous-ensemble E' de E tel que les membres gauches (notés lhs pour *left hand side*) dans E' sont de la forme $f(t_1,...,t_n)$ avec les t_i dans T(C,X) et la spécification SP' = (S, C∪D, E') définit les mêmes opérateurs que SP, c'est-à-dire $=^i_{SP}$ et $=^i_{SP'}$ sont les mêmes congruences.

Preuve: Par l'absurde. Soit E' l'ensemble des équations l = r dans E telles que le membre gauche l est de la forme $f(t_1,...,t_n)$ avec les t_i' dans T(C,X). Soit $g:s_1,...,s_n \to s$ un élément de D. Supposons qu'il existe $(\tau_1,...,\tau_n)$ dans $T_{s_1}(C)X...XT_{s_n}(C)$ tel qu'il n'y ait pas d'équation (règle) dans E' dont le membre gauche filtre $g(\tau_1,...,\tau_n)$. Ceci est contradictoire avec la condition (2) de la définition 4.8. D'où g est complètement défini par E' et ensuite $=^i_{SP}$ est identique à $=^i_{SP'}$. cqfd.

Soit SP = (S, Ω, E) une spécification satisfaisant les conditions de la définition 4.8. Soient f un opérateur défini ($f \in D$) d'arité $s_1...s_k$, et E_f l'ensemble des équations définissant f. Alors, les membres gauches dans E_f sont de la forme $f(P_1^i,..., P_k^i)$ avec $1 \leq i \leq q$ et l'ensemble $P = \{P_i \mid P_i = (P_1^i, ..., P_k^i)$, $1 \leq i \leq q\}$ est complet pour C. D'après le lemme 4.4 nous supposons que tous les P_j^i appartiennent à T(C, X), par contre les P_i ne sont pas nécessairement deux à deux non strictement sous-unifiables. Notre objectif est de calculer un nouvel ensemble pour f, soit $P' = \{P'_i \mid P'_i = (P'_1{}^i, ..., P'_k{}^i)$, $1 \leq i \leq q'\}$, tel que P' soit complet pour C et que les P'_i soient deux à deux non strictement sous-unifiables. Pour cela,

nous définissons ci-dessous une procédure de transformation "TP" qui prend comme entrées un ensemble de "patterns" tel que P et produit en sortie un ensemble tel que P': (TP(P)=P').

Soit $P = \{P_i \mid P_i = (P_1^i, ..., P_k^i)\ ,\ 1 \le i \le q\}$ un ensemble complet pour C tel que les P_j^i appartiennent à T(C, X). Dans la suite, la notation $P_j^{i=1..q}$ dénote l'ensemble des $j^{ème}$ composantes dans les P_i.

TP(P) = Transform $(P_1^{i=1..q})$ X ... X Transform $(P_k^{i=1..q})$
-- X dénote le produit cartésien.

Transform $(P_j^{i=1..q})$ =
if pour tout i =1..q, P_j^i est une variable **then**
if T(C)s_j est fini **then** $T_{s_j}(C)$
--$T_{s_j}(C)$ dénote les termes sur C de sorte s_j.
else $\{x_j\}$ -- x_j est une nouvelle variable (bien "sortée").
else {constantes de sorte s_j s'il y en a} $\cup$ $\cup_{c \in C}$ { c ◊ TP (R)}
-- Où R = $\{Q_r \mid c(Q_r) \in P_j^{i=1..q}\} \cup$
--{$(x_1,...,x_m) \mid \exists$ i, P_j^i est une variable et c est d'arité $s_1,...,s_m \rightarrow s_j$}

f ◊ $\{t_1,..., t_k\} = \{f(t_1), ..., f(t_n)\}$ -- Nous utilisons la notation infixée pour l'opération ◊ .

Informellement, Transform $(P_j^{i=1..q})$ décompose la $j^{ème}$ composante le plus possible par rapport aux structures des P_j^i. Quant à l'opérateur "c ◊ $\{t_1,..., t_n\}$", il "applique" le constructeur c à chaque terme t_i produisant l'ensemble $\{c(t_1),..., c(t_n)\}$. Dans ce qui suit nous établissons quelques propriétés de l'ensemble résultant TP(P), où P est un ensemble complet pour C.

Lemme 4.5 : Soit SP=(S, Ω, E) une spécification satisfaisant les conditions de la définition 4.8. Soient $P = \{P_i \mid P_i = (P_1^i, ..., P_k^i)\ ,\ 1 \le i \le q\}$ un ensemble complet pour C avec les P_j^i dans T(C, X) et P'= TP(P). Alors :

(1) TP termine toujours.

(2) $P' = \{P'_i \mid P'_i = (P'_1{}^i, ..., P'_k{}^i)\ ,\ 1 \le i \le q'\}$ est un ensemble complet pour C et pour tout k-uplets $(t_1,...,t_k)$ constitué de termes fermés et normalisés, il existe un unique i avec $1 \le i \le q'$ et une substitution σ telle que pour tout j=1..k, $\sigma(P'_j{}^i) = t_j$.

(3) les éléments de P' sont deux à deux non strictement sous-unifiables.

(4) $\forall P'_i \in P', \exists j \in \{1..q\}, \exists \sigma \mid \sigma(P_j)=P'_i$.

Preuve :

(1) Par induction sur la hauteur maximale des P_i (preuve directe).

(2) Par induction sur la hauteur maximale des P_i (notée ci-dessous : max-h).

Nous devons montré que : (A) pour tout t dans $T(C)_{s_j}$, il existe un unique terme t' dans Transform($P_j^{i=1..q}$) qui filtre t.

(cas de base): max-h = 0. Dans ce cas si l'ensemble $P_j^{i=1..q}$ est constitué de variables (respectivement de constantes) seulement, L'ensemble résultat Transform($P_j^{i=1..q}$) est soit $\{t \mid t \in T_{s_j}(C)\}$ ou $\{x_j\}$ (respectivement {constantes de sorte s_j}). dans les deux cas la propriété (A) est vraie. Si l'ensemble $P_j^{i=1..q}$ contient des constantes et des variables, Transform($P_j^{i=1..q}$) sera égal à {constantes de sorte s_j} $\cup$ $\{c \Diamond TP(\{(x_1,...,x_n) \mid c: s_1,...,s_n\text{-->}s_j \in C\})\}$. Dans ce cas, la propriété (A) est vraie aussi.

(Hypothèse d'induction) : max-h $\leq$ n => $\forall (t_1,...,t_k) \in T_{s_1}(C)X...XT_{s_k}(C), \exists!\ p' \in P' \mid p'$ filtre $(t_1,...,t_k)$.

Supposons que max-h = n+1. Soit $P_j^{i=1..q}$ tel que la hauteur maximum de ses termes soit n+1.

Transform($P_j^{i=1..q}$) = {constantes de sorte s_j s'il y en a} $\cup$ $\cup_{c \in C}$ $\{ c \Diamond TP(R)\}$; où $R=\{Q_r \mid c(Q_r) \in P_j^{i=1..q}\} \cup \{(x_1,...,x_m) \mid \exists i, P_j^i$ est une variable et c est de profil $s_1,...,s_m\text{->}\ s_j\}$

L'ensemble R est complet sur $(s_1,...,s_m)$ (d'après la complétude de P) et la hauteur maximum des termes dans R est inférieure ou égale à n. En utilisant l'hypothèse d'induction, nous avons : $\forall t \in T(C)_{s_j}, \exists!\ t' \in$ Transform($P_j^{i=1..q}$) tel que t' filtre t. D'où la propriété (A) est vraie.

(3) Directe d'après la définition de TP.

(4) Par absurde. Supposons qu'il existe un "pattern" p' dans P' tel qu'il n'y ait pas de p dans P qui filtre p'. Alors, soit t une instance fermée et normalisée de p'. p' est le seul pattern dans P' qui filtre t (d'après (2)). Or, puisque l'ensemble P est complet, il existe p dans P tel que p filtre t. Ceci veut dire que p et p' sont unifiables, c'est-à-dire, il existe σ_1 et σ_2 telles que σ_1 (p) = σ_2 (p') où σ_2 n'est pas un renommage. Ceci contredit la définition de TP. En effet, Transform($P_j^{i=1..q}$) est égal à $\{x_j\}$ seulement si tous les $j^{èmes}$ composantes sont des variables. cqfd.

Maintenant nous définissons la procédure qui transforme les spécifications, soit TSP. Pour cela nous considérons seulement un seul opérateur défini, la généralisation à toute une spécification est immédiate. Soit E_f l'ensemble des équations définissant f (avec $f \in D$).

$E_f=\{f(P_i)== r_i$,avec $P=\{P_i, i =1..p\}$ un ensemble complet pour C et les $P_j^i \in T(C,X)\}$.

Nous définissons TSP (E_f) comme étant l'ensemble E'_f suivant :

E'_f={f(P'$_j$) == forme normale de $\sigma(r_i)$ par rapport à R (défini par E), σ est telle que $\sigma(P_i) = P'_j$}

L'existence de σ est garantie par le lemme 4.5.

Théorème 4.4 : Soit SP = (Σ, E) une spécification satisfaisant le principe de définition de Huet et Hullot. Alors, il existe une spécification SP'=(Σ,E') équivalente à SP (définissant les même fonctions) satisfaisant les conditions CSU.

Preuve : Se déduit du lemme 4.5 et de la définition de TSP[]

Exemple 4.3 : Nous considérons la spécification de l'opération de l'égalité sur les naturels à valeur booléenne, eq: nat, nat --> bool, défini comme suit :

E_{eq}={1 : eq(i_0, i_0) == true
2 : eq(s(i_1), 0) == false
3 : eq(0, s(i_2)) == false
4 : eq (s(i_3), s(i_4)) == eq(i_3, i_4)}

E_{eq} ne satisfait pas toutes les conditions de CSU puisque le membre gauche de la première équation est strictement sous-unifiable avec les autres membres gauches.

C ={0, s, true, false} and D ={eq}

L'ensemble P={(i_0, i_0), (s(i_1), 0), (0, s(i_2)), (s(i_3), s(i_4))} est complet pour C (évident).

TP(P) = Transform({i_0,s(i_1), 0, s(i_3)}) X Transform({i_0, 0, s(i_2), s(i_4)})

Transform({i_0,s(i_1), 0, s(i_3)}) = {0} $\cup$ s $\Diamond$ TP({i_1, i_3, x_1})

TP({i_1, i_3, x_1}) ={x_2}

Alors, Transform({i_0,s(i_1), 0, s(i_3)}) ={0, s(x_2)}

Transform({i_0, 0, s(i_2), s(i_4)}) = {0} $\cup$ s $\Diamond$ TP({i_2, i_4, x_3})

TP({i_2, i_4, x_3}) = {x_4}

Alors, Transform({i_2, i_4, x_3}) = {0, s(x_4)}

TP(P) = {(0,0),(0, s(x_4),(s(x_2),0),(s(x_2), s(x_4))}

Enfin, on trouve l'ensemble E'_{eq}= TSP (E_f) :

E'_{eq} = {1 : eq(0,0) == true
2 : eq(0, s(x_4)) == false

3 : eq(s(x_2), 0) == false

4 : eq(s(x_2), s(x_4)) == eq (x_2, x_4)}

Un autre exemple de transformation est celui donné dans l'exemple 4.2. Il est en effet facile de vérifier que E' = TSP(E).

4.4. CONCLUSION

Nous avons présenté dans ce chapitre les spécifications uniformes. D'après les résultats du chapitre 3, la résolution d'équations par surréduction avec n'importe quelle stratégie est complète en présence de telles spécifications. Nous avons montré la décidabilité de ces spécifications et avons proposé des conditions suffisantes pour définir des spécifications uniformes. Dans [Fri85], Fribourg donne des conditions similaires aux conditions CSU, excepté la condition (4), permettant de montrer la complétude de la stratégie "innermost". Une autre preuve de la complétude de cette stratégie est obtenue en montrant l'uniformité de la stratégie "innermost", ce qui est immédiat d'après la forme des membres gauches et la complétude des définitions des fonctions.

Lorsqu'une spécification n'est pas uniforme, on peut utiliser soit des stratégies uniformes (innermost dans le cas [Fri85]) ou bien on peut choisir "dynamiquement", à chaque pas de surréduction, une position de surréduction de telle sorte que le sous-terme à la position choisie soit uniforme (cf. remarque4.1). Cependant, ce choix dynamique de position de surréduction n'est pas toujours possible puisqu'il existe des spécifications et des équations pour lesquelles aucune stratégie de surréduction n'est uniforme. En effet soit la spécification SP=(Σ, E) avec :

E = { 1 : f(s(x), 0) == s(0)

2 : f(x, s(y)) == s(0)

3 : g(0) == s(0) },

et "f(g(v), v) == s(0)" l'équation à résoudre. Les dérivations de surréduction possibles issues de l'équation à résoudre sont :

f(g(v), v) == s(0) --N-->["1", 2, {v→ s(z),...}] s(0)==s(0)

f(g(v), v) == s(0) --N-->["11", 3, {v→ 0}] f(s(0), 0) == s(0) -----> s(0) == s(0)

Dans l'équation f(g(v), v) == s(0), il y a deux positions de surréduction "1" et "11". Les solutions calculées par les deux dérivations sont distinctes et "disjointes"(c'est-à-dire qu'aucune solution ne généralise l'autre). Par conséquent, aucune stratégie de surréduction ne peut être uniforme dans ce cas.

D'autres auteurs ont utilisé la condition (4) de CSU pour prouver la complétude des stratégies de surréduction, nous citons par exemple Padawitz dans [Pad88] et Frustos-Escrig et Fernandez-Camacho dans [FF90].

Chapitre 5

Application à un Langage de Programmation

Dans ce chapitre, nous présentons les principales caractéristiques d'un langage fonctionnel et logique. Ce langage est LPG1.8 [BE86]. Il est le résultat de la fusion de deux langages : LPG1.2 [Ber83a], un langage équationnel, et Prolog-pur, le langage de la logique des clauses de Horn. Notre présentation de LPG1.8 est informelle ; nous renvoyons le lecteur intéressé par des détails techniques à [BDE87a] [Be et al90] et pour la sémantique d'un sous-ensemble de LPG1.8 à [Rey87]. Ce chapitre illustre une application des résultats des chapitres précédents. Nous montrons à travers quelques exemples les différents traits de LPG1.8. Aussi, discutons-nous des caractéristiques suivantes: les définitions des types abstraits de données (§5.1), la programmation équationnelle (une forme de programmation fonctionnelle) (§5.2), la programmation "logique" (§5.3), la programmation logico-fonctionnelle (§5.4), la modularité (§5.5) et la généricité (§5.6). Nous discutons aussi de l'interprète que nous avons réalisé pour ce langage, dans le paragraphe (§5.7). Nous terminons ce chapitre par un bref survol des langages voisins de LPG1.8 (§5.8).

5.1. TYPES ABSTRAITS

L'abstraction est l'une des principales caractéristiques d'un langage de très haut niveau. Cette notion permet en effet aux utilisateurs de décrire les objets, indépendamment de leur représentation en machine. L'utilisateur évite ainsi tout codage prématuré des objets. La mise en œuvre de cette méthode de conception de structures de données et de programmes se trouve plus aisée lorsque le langage utilisé offre des facilités d'abstraction.

Dans LPG1.8, la notion d'abstraction se manifeste par le biais des types abstraits de données. Informellement, un type abstrait de données (dénoté TAD dans la suite) définit une classe d'objets, qui est caractérisée par les opérations possibles sur ces objets (opérations de création et de manipulation des objets). L'utilisation de TADs est plutôt une discipline qu'un programmeur peut suivre, même en utilisant des langages classiques comme PL/1, PASCAL ou C, ou des langages où la notion de TAD existe déjà, comme ADA, CLU ou SIMULA. Toutefois, dans ces langages classiques, la définition des TADs se fait en deux étapes : une première phase de spécification des opérations caractérisant le TAD à définir, et une deuxième

phase d'implantation (on dit aussi *représentation*) des opérations du TAD. Pour pallier cette deuxième phase qui peut être une source d'erreurs, de nouveaux langages, dits algébriques, sont apparus à la suite des travaux de Liskov et Zilles [LZ74], de Guttag [Gut74] et du groupe ADJ [GTWW75] ; ces langages permettent une définition algébrique des TADs en s'affranchissant des détails d'implantation. Des travaux antérieurs faisaient déjà apparaître l'aspect algébrique des types de données [Bur69]. Le fruit de ces travaux est la méthode de spécification algébrique des TADs, c'est-à-dire la donnée d'une signature, soit $\Sigma = (S, \Omega, \Pi)$ (définitions 1.1 et 1.2), puis des axiomes définissant les opérations et éventuellement les prédicats de la signature (définition 1.7)). LPG1.8 se base sur cette méthode de définition de types abstraits. De plus, pour des raisons d'analyse de spécifications que l'on verra au paragraphe §5.5, ont été introduites des conditions syntaxiques de définition de TADs ; il s'agit, dans LPG1.8 qui suit en cela [HH80] et [BQS80] de partager l'ensemble des opérateurs Ω de la signature Σ en deux sous-ensembles C et D avec $\Omega = C \cup D$ et $C \cap D = \emptyset$. L'ensemble C est constitué d'opérateurs dits constructeurs (ensembles des opérateurs à partir desquels il est possible d'engendrer toutes les valeurs (termes) des TADs à définir) et D est l'ensemble des opérateurs dits définis (opérateurs spécifiques aux applications et non nécessaires pour définir les types de données). Dans LPG1.8, les TADs sont définis dans des "modules" désignés syntaxiquement par le mot-clé *type* comme illustré dans l'exemple suivant. La présentation d'un tel module est de la forme $SP=(\Sigma, E \cup Ch)$ où l'ensemble des opérations Ω de signature $\Sigma = (S, \Omega, \Pi)$ est scindé en deux parties C, ensemble des constructeurs et D, ensemble des opérateurs définis équationnellement par l'ensemble E. L'ensemble Ch est l'ensemble de clauses définissant les prédicats dans Π.

Exemple 5.1 : Nous nous proposons de définir trois types abstraits :

- le type abstrait des naturels : *nat*
- le type abstrait des points sur $\mathbb{N}^2$: *dot*
- le type abstrait des lignes droites sur $\mathbb{N}^2$: *bee-line*

Nous définissons par la même occasion l'opération d'addition sur les naturels (notée +) et l'opération d'égalité sur les naturels à valeur booléenne (notée =) ; ainsi que deux prédicats sur les droites spécifiant respectivement qu'une ligne est horizontale (`horizontal`) ou qu'une ligne est verticale (`vertical`). Nous supposons que le type abstrait des booléens est défini par ailleurs[1].

```
type Nat_Dot
   sorts
         nat, dot, bee_line
   constructors
         0 : -> nat
         succ : nat -> nat
         d : (nat, nat) -> dot
```

[1]Dans LPG1.8, les importations de spécifications se font de manière implicite.

```
        bee_l : (dot, dot) -> bee_line
    operators
        + : (nat, nat) -> nat
        = : (nat, nat) -> bool
    predicates
        vertical, horizontal : bee_line
    variables
        i, j, k : nat
    equations
        1: 0 + i == i
        2: succ(i) + j == succ(i + j)
        3: 0 = 0 == true
        4: 0 = succ(i) == false
        5: succ(i) = 0 == false
        6: succ(i) = succ(j) == x = y
    clauses
        7: vertical (bee_l(d(i,j), d(i,k))) <== k=j == false
        8: horizontal (bee_l(d(i,j), d(k,j)) <== k=i == false
end Nat_Dot
```

La sémantique[1] de la présentation (ou spécification) SP = (Σ, E, C) d'un module "type" est le E-modèle standard défini par la présentation (voir chapitre 1).

Pour définir un TAD, la donnée seule de l'ensemble des constructeurs suffit. Or les TADs ne sont utiles que lorsqu'on les utilise, c'est-à-dire s'ils figurent dans les domaines des fonctions ou des prédicats. Les définitions des fonctions et des prédicats sont abordées dans les deux paragraphes suivants.

5.2. PROGRAMMATION FONCTIONNELLE

L'idée principale de la programmation fonctionnelle est d'exprimer les algorithmes à l'aide de fonctions. Les langages qui permettent ce style de programmation sont souvent appelés langages fonctionnels ou applicatifs - *applicatif* faisant allusion à l'application des fonctions à leurs arguments-. Les avantages que l'on peut apprécier dans cette méthode de programmation sont multiples. Citons par exemple :

- la clarté des programmes. La syntaxe de certains langages étant relativement proche des notations mathématiques usuelles.
- la concision des programmes.
- la réutilisation de programmes est facilitée grâce aux fonctions d'ordre supérieur (fonctionnelles).

---

[1] Le langage LPG1.8 étant générique et modulaire, la sémantique des différents modules ne peut être définie de façon aussi directe. Nous ne donnons dans ce mémoire que l'idée intuitive de cette sémantique. Pour une sémantique plus conséquente, voir [EM85][EM90] [GM87][Rey87]. Pour le moment il n'existe pas de sémantique établie pour le langage LPG1.8 dans sa généralité.

- l'absence de l'affectation qui fait tant de tort aux langages "classiques" à la PASCAL.
- l'absence des instructions de "saut" comme GOTO.
- l'existence de méthodes de preuves de programmes .
- l'existence d'algorithmes de transformation de programmes vers d'autres programmes plus efficaces [BD69].
- l'existence de machines effectives orientées langages fonctionnels (par exemple machine lisp).
- l'existence de technique de compilation [Pey87].
- etc

Les langages fonctionnels actuels sont basés sur différents formalismes. Le λ-calcul a inspiré beaucoup d'auteurs comme par exemple Mc Carthy dans LISP [McC60], Landin dans ISWIM [Lan66], Gorden et al. dans ML [GMW79], Burstall et al. dans HOPE [BQS80]. On trouve aussi d'autres langages qui utilisent la notion de *combinateur* , FP [Bac77] "et APL [Ive62]" en sont des exemples. Signalons que ces deux derniers langages permettent d'écrire des programmes "trop" concis et cela aux dépens de leur lisibilité. Le formalisme algébrique, quant à lui, a fait naître un autre style de langage où les fonctions sont définies par des équations conditionnelles récursives sur les types de données. A titre d'exemples de langages algébriques, nous citons CLEAR [BG77], HOPE [BQS80], OBJ2 [FGJM84], ACT ONE [EM85]. Les langages fonctionnels peuvent aussi se distinguer selon qu'ils sont typés ou non, par exemple LISP et KRC [Tur81] ne sont pas typés alors que ML et MIRANDA [Tur85] sont respectivement les "extensions" typées de LISP et de KRC. D'autre part, la sémantique opérationnelle peut être aussi un point de différence entre les langages fonctionnels. Citons dans ce registre les langages KRC et MIRANDA qui admettent des structures infinies grâce à leur stratégie d'évaluation dite paresseuse (lazy evaluation). Cette stratégie est justifiée du moment que les fonctions ne sont pas considérées comme strictes. Cependant la majorité des langages fonctionnels suppose que les fonctions sont strictes (informellement, une fonction est stricte ssi elle est définie seulement si tous ses arguments sont définis).

LPG1.8, se basant sur la logique des clauses de Horn avec égalité, est un langage fonctionnel. Les fonctions y sont définies par des équations conditionnelles récursives, souvent par induction structurelle [Bur69] [Aub79] sur les constructeurs des TADs. Les équations doivent vérifier les conditions de CSU (cf.§4.2) hormis la condition (4). La condition (4) de CSU n'est pas requise puisque nous utilisons la stratégie de Surréduction inner-most qui est uniforme (voir §4.4 et §5.7). Dans LPG1.8 et à l'instar des autres langages algébriques, les fonctionnelles ne sont pas spécifiables. Ceci n'est point surprenant puisque les logiques sous-jacentes à ces langages sont du premier ordre (logique

équationnelle, logique conditionnelle, logique des clauses de Horn avec égalité). Cependant, comme nous le verrons dans le §5.6, la généricité nous permettra, sans parler explicitement de fonctionnelle, d'approcher l'expressivité des langages fonctionnels d'ordre supérieur.

Exemple 5.2 : Nous définissons trois opérations sur les naturels : la multiplication `*`, la factorielle `fact` et une fonction `dist` (pour distance) qui calcule la valeur absolue de la différence entre deux naturels. Cette dernière fonction utilise la fonction - (x - y coincide avec dist(x, y) lorsque x est supérieur ou égal à y ; et vaut 0 sinon) . Nous profitons de cette exemple pour signaler la possibilité dans LPG1.8 de déclarer certaines équations en tant que théorèmes inductifs et ce sous la rubrique `theorems`. Ces théorèmes doivent donc être valides dans le E-modèle standard, défini par l'axiomatisation des opérations figurant dans les théorèmes déclarés.

```
enrichment   On_Nat
      operators
            *, -, dist: (nat, nat) -> nat
            fact : nat -> nat
      variables
            i, j, k : nat
      equations
            1: 0 * i == 0
            2: succ(i) * j == j + (i * j)
            3: fact(0) == 1
            4: fact (succ(i)) == succ(i) * fact (i)
            5: i - 0 == i
            6: 0 - succ(i) == 0
            7: succ(i) - succ(j) == i - j
            8: dist(i, j) == (i - j) + (j - i)
      theorems
            9: i * 0 == 0
            10: i * succ(j) == i + (i * j)
            11: i * (j + k) == (i * j) + (i * k)
end On_Nat
```

5.3. PROGRAMMATION "LOGIQUE"

La programmation "logique" ou PROLOG (pour PROgrammation en LOGique) est le nom générique qui sous-entend la programmation à l'aide du langage de la logique des clauses de Horn [Kow74]. Les algorithmes y sont donc exprimés par des prédicats au lieu de fonctions, comme c'était le cas ci-dessus ; c'est pour cela que nous préférons qualifier ce genre de langage de *langage prédicatif*. Son aspect déclaratif, le rapprochant de l'utilisateur, lui a conféré un succès certain depuis son apparition au début des années 1970. De plus, PROLOG garde une certaine similitude avec les langages classiques, comme l'a souligné Kowalski dans [Kow74] où il donne une interprétation procédurale de ce langage. La

programmation prédicative trouve ses applications dans des domaines de plus en plus nombreux ; citons par exemple les Bases de Données (ou de connaissances) ou l'Intelligence Artificielle (programmation de systèmes experts). Programmer en PROLOG consiste à déclarer des implications comme :

```
"fait1 et fait2 et ... et faitn => fait0" ou
"fait0"
```

Vu l'historique du langage PROLOG, les objets du langage ne sont pas typés. Ceci prive l'utilisateur d'un outil précieux qu'est le "contrôleur" du typage, et ouvre ainsi le champ à certaines erreurs difficilement détectables et aussi à des utilisations "non recommandées".

Les implantations de PROLOG fidèles à la théorie se basent sur la règle d'inférence dite SLD-résolution[KK71] (voir définition 2.9). Malheureusement, de telles implantations s'avèrent inadéquates pour résoudre des problèmes non scholastiques. Il a "fallu" donc modifier la syntaxe de PROLOG en lui ajoutant des structures de contrôle ou d'autres "impuretés" (impureté d'un point de vue logique, s'entend) pour soit faciliter la programmation, soit améliorer les performances de l'interprèteur. Les "impuretés" les plus célèbres sont le "cut" et l'absence de vérification de l' "occur-check" dans l'algorithme d'unification. L'interprétation des programmes engendre en général des arbres de recherche assez grands. Pour remédier à ce problème, des techniques d'implantation de parcours d'arbre ont été utilisées. Parmi les stratégies utilisées, on trouve les parcours en profondeur (depth-first), parallèleOU et parallèleET. Les deux dernières stratégies font encore l'objet de nombreuses activités de recherche et par conséquent n'ont pas été très expérimentées. D'autre part, les interprètes classiques -parcours en profondeur- sont relativement lents. L'une des causes de ce manque d'efficacité découle du fait que les opérations de base, à savoir l'unification et la substitution, sont assez différentes des opérations des processeurs classiques, MC-68000 par exemple. Warren, soucieux de ce problème, a proposé une machine virtuelle dont le langage sert comme langage cible de son compilateur [War77]. Depuis son travail, d'autres compilateurs ont vu le jour ; citons à titre d'exemple les travaux de Nilsson [Nil83].

LPG1.8, se basant sur la logique des clauses de Horn avec égalité, est aussi un langage prédicatif. Les prédicats y sont typés, c'est-à-dire chaque prédicat a un profil et ne peut pas donc s'utiliser avec des arguments qui ne correspondent pas aux profils déclarés (la surcharge des symboles est autorisée). Remarquons que cette façon de typer les prédicats, qui est immédiate, ne correspond pas à ce que l'on appelle *type* dans "la communauté programmation logique", voir par exemple [Mis84], où par *type* d'un prédicat est sous-entendu l'extension du prédicat (c'est-à-dire les valeurs pour lesquelles le prédicat est vrai).

Les prédicats, excepté les prédicats d'égalité, sont donc définis à l'aide de clauses de Horn dont les corps peuvent contenir des équations. La négation par l'échec introduite par Clark [Cla78] n'est pas présente dans LPG1.8 à cause de la présence des prédicats d'égalité. Mais les fonctions peuvent s'en charger dans une certaine mesure [Fri85].

Exemple 5.3 : Nous donnons dans cet exemple une spécification du problème classique des huit reines. Cet exemple illustre le côté "prédicatif" de LPG1.8 (voir les clauses 1 à 12 pour des définitions à la PROLOG pur). La clause 13 donne une première idée sur la combinaison des langages fonctionnels et prédicatifs que nous discutons dans le paragraphe suivant. Nous utilisons la notation [i, j, k, l] pour désigner une séquence. Ce genre de notation est possible dans LPG1.8 [BDE87a].

```
type   Queens
       sorts
              seq_nat, seq_dot
       constructors
              nil_n : -> seq_nat
              nil_d : -> seq_dot
              cons_n : (nat, seq_nat) -> seq_nat
              cons_d : (dot, seq_dot) -> seq_dot
       predicates
              eight_Queens : seq_dot
              Queens : (seq_nat, seq_dot)
              Permut : (seq_nat, seq_nat)
              Insert : (nat, seq_nat, seq_nat)
              Pairs :  (seq_nat, seq_nat, seq_dot)
              Safe : seq_dot
              Check : (dot, seq_dot)
              Diag : (dot, dot)
       variables
              i, j, k, l, dx, dy, z : nat
              dt, pt : dot
              s, u, v : seq_nat
              Pos, sd : seq_dot
       clauses
              1: eight_Queens(Pos)<==Queens([8,7,6,5,4,3,2,1],Pos)
              2: Queens(s,Pos) <== Permut(s,u), Pairs(s,u,Pos), Safe(Pos)
              3: Permut(nil_n, nil_n)
              4: Permut(cons_n(i,s),u)<== Permut(s,v),Insert(i,v,u)
              5: Insert(i,s,cons_n(i,s))
              6: Insert(i,cons_n(j,s),cons_n(j,u))<== Insert(i,s,u)
              7: Pairs(nil_n, nil_n, nil_d)
              8: Pairs(cons_n(i,s),cons_n(j,u),cons_d(d(i,j),sd))<==
                                                          Pairs (s,u,sd)
              9: Safe (nil_d)
              10: Safe (cons_d(dt, sd)) <== Check (dt, sd),Safe(sd)
              11: Check(dt, nil_d)
              12: Check(dt,cons_d(pt,sd)) <== Diag(dt,pt), Check(dt,sd)
              13: Diag(d(i,j),d(k,l)) <==  dx == dist(i,k),
                                           dy == dist(j,l),
                                           dist(dx,dy) == succ(z)
end Queens
```

5.4. PROGRAMMATION FONCTIONNELLE ET PREDICATIVE

Nous avons vu dans les deux paragraphes précédents que nous pouvions aussi bien définir des fonctions que des prédicats dans LPG1.8. Evidemment cela ne veut pas dire que nous avons deux langages distincts dans un seul "système", et que nous traitons chaque partie à part. La combinaison est bien réelle, c'est-à-dire les fonctions apparaissant dans les atomes peuvent être définies à l'aide d'équations.

L'idée de combiner les langages fonctionnels et prédicatifs n'est pas récente, bien que un effort particulier ait été voué (et continue à l'être) à ce genre de langage durant ces dernières années. Déjà en 1981, LOGLISP [RS81], un langage visant la combinaison de PROLOG et LISP, est apparu (voir §5.8). En démonstration automatique, Robinson et Wos ont proposé en 1969 une règle de déduction complète, à savoir la paramodulation [RW69], qui permet de calculer dans la logique des clauses de Horn avec égalité. D'autres langages, dont la finalité est la combinaison des aspects fonctionnels et prédicatifs, ont été proposés, voir §5.8. En LPG1.8, aussi bien des fonctions que des prédicats peuvent être définis. Les fonctions sont définies comme nous l'avons présenté dans §5.2. Par contre, pour les prédicats, outre les définitions à la PROLOG classique, il est possible dans LPG1.8 d'inclure des égalités dans les corps des clauses. Evidemment les symboles de fonctions apparaissant dans les atomes ne sont pas forcément des constructeurs de types comme c'est le cas dans PROLOG, mais peuvent être définis par des équations conditionnelles. Notons enfin, que nous n'autorisons pas l'utilisation des prédicats dans les définitions de fonctions.

Exemple 5.4 : Nous définissons cinq prédicats différents. `cutting(d,d')` (respectivement `parallel(d,d')`) est vrai lorsque les droites d et d' sont sécantes (respectivement parallèles). `ordered(s)` est vrai lorsque la séquence s est ordonnée. `Cats-Birds(i,j,k,l)` est vrai lorsque k vaut la somme de i et j (k est le nombre de têtes si l'on interprète i et j comme étant respectivement le nombre de chats et le nombre d'oiseaux), et l vaut 4*i+2*j (le nombre de pattes). Enfin le prédicat `n_Queens(n,Pos)` est vrai lorsque les positions dans Pos sont conformes aux règles de placement de n reines sur un échiquier nXn. Dans chacune de ces définitions, nous utilisons des fonctions définies équationnellement. Nous supposons que les fonctions `sort` et `seqn`, et le prédicat `same-sign`, décrits ci-dessous sont définis par ailleurs:

- `sort : seq_nat -> seq_nat`. Le résultat de l'application `sort(s)` est la séquence triée selon un ordre total que nous supposons aussi défini.

- `seqn : nat -> seq_nat`. Le résultat de l'application `seqn(n)` est la séquence [n, n-1,...,1].

- `Same_sign : (bee_l, bee_l). Same_sign(d1, d2)` est vrai lorsque les pentes des droites d1 et d2 ont le même signe.

les autres fonctions utilisées ont déjà été définies dans les exemples précédents.

```
enrichment   Fonc_Pred
     predicates
          cutting, parallel : (bee_line, bee_line)
          ordered : seq_nat
          Cats_Birds : (nat, nat, nat, nat)
          n_Queens : (nat, seq_dot)
     variables
          i1, i2, j1, j2, n1, n2, m1, m2, c, b, z : nat
          s : seq_nat
          Pos : seq_dot
     clauses
          1: parallel(bee_l(d(i1,j1),d(n1,m1)),beel(d(i2,j2),d(n2,m2)))
               <==  dist(i1,n1)*dist(j2,m2) == dist(i2,n2)*dist(j1,m1),
                         Same_sign(bee_l(d(i1,j1),d(n1,m1)),
                                   bee_l(d(i2,j2),d(n2,m2)))

          2: cutting(bee_l(d(i1,j1),d(n1,m1)),beel(d(i2,j2),d(n2,m2)))
                         <== dist(dist(i1,n1)*dist(j2,m2),
                                  dist(i2,n2)*dist(j1,m1)) == succ(z)
                                  Same_sign(bee_l(d(i1,j1),d(n1,m1)),
                                            bee_l(d(i2,j2),d(n2,m2)))

          3: ordered (s) <== sort(s) == s
          4: Cats_Birds(c, b, c+b, 4*c+2*b)
          5: n_Queens(n1, Pos) <== Queens(seqn(n1),Pos)
end Fonc_Pred
```

5.5. PROGRAMMATION MODULAIRE

La conception de logiciel de façon modulaire est une nécessité [Par72]. Tous les langages modernes en offrent la possibilité, et le langage LPG 1.8 ne fait pas exception ; il permet la conception de programmes modulaires, aussi bien en largeur qu'en profondeur. Il existe trois sortes de modules en LPG1.8 : les *Types*, les *Enrichissements* et les *Propriétés*. Nous illustrons dans ce paragraphe les deux premiers modules. Les modules de *Propriétés* sont introduits dans le paragraphe suivant.

Rappelons d'abord les définitions d'une spécification *extension* d'une autre spécification, et d'une spécification *enrichissement* d'une autre spécification [EM85].

Définition 5.1 : (extension, enrichissement)

Soit SP= (Σ, E $\cup$ CH) une spécification avec Σ = (S, Ω, Π) . SP est appelée **extension** de la spécification SP'= (Σ', E'$\cup$ CH') avec Σ' = (S', Ω', Π') ssi S'$\subseteq$ S, Ω'$\subseteq$ Ω, Π' $\subseteq$ Π, E'$\subseteq$ E, et CH'$\subseteq$ CH. La spécification SP est appelée **enrichissement** de SP' lorsque S=S' et SP est une extension de SP'.

Syntaxiquement, la notion d'extension dans LPG1.8 correspond à l'utilisation d'un module *Type* alors que la notion d'enrichissement correspond aux modules *Enrichissement.*

Exemple 5.5 : Nous donnons dans cette exemple la spécification de deux types abstraits : celui des *bits* (binary digit) avec l'addition des bits ainsi que le calcul des retenues. Nous définissons ensuite le type des *mots* -nous spécifierons les mots de quatre bits seulement- avec le prédicat de parité qui utilise l'addition des bits. Nous enrichissons ensuite ces deux types abstraits par la définition de l'addition des mots.

```
type   Bit
       sorts
              bit
       constructors
              I : -> bit
              O : -> bit
       operators
              add_bit : (bit, bit, bit) -> bit
              carry_bit : (bit, bit, bit) -> bit
       variables
              i, j : bit
       equations
              1: add_bit (i, i, j) == j
              2: add_bit (i, j, i) == j
              3: add_bit (j, i, i) == j
              4: carry_bit (i, i, j) == i
              5: carry_bit (i, j, i) == i
              6: carry_bit (j, i, i) == i
end Bit
```

```
type   Word
       sorts
              word
       constructors
              wd : (bit, bit, bit, bit) -> word
       operators
              carry_wd : (word, word, bit) -> bit
       predicates
              parity : word
       variables
              i0, i1, i2, i3 : bit
              j0, j1, j2, j3 : bit
              c0, c1, c2, c3 : bit
              w : word
       equations
              1: carry_wd (wd(i0, i1, i2, i3),wd(j0, j1, j2, j3), c0) ==
                     let   c1 = carry_bit(i0, j0, c0)
                           c2 = carry_bit(i1, j1, c1)
                           c3 = carry_bit(i2, j2, c2)
                     in    carry_bit (i3, j3, c3) endlet
       clauses
              2: Parity (wd(i0, i1, i2, i3)) <==
```

```
                                    add_bit(i0,add_bit(i1,add_bit(i2, i3))) == O
end Word
```

```
enrichment   Add_words
      operators
            add_wd : (word, word, bit) -> word
      variables
            i0, i1, i2, i3 : bit
            j0, j1, j2, j3 : bit
            c0, c1, c2, c3 : bit
            r0, r1, r2, r3 : bit
      equations
            1: add_wd (wd(i0, i1, i2, i3), wd(j0, j1, j2, j3), c0) ==
                  let   r0 = add_bit(i0, j0, c0)
                        c1 = carry_bit(i0, j0, c0)
                        r1 = add_bit(i1, j1, c1)
                        c2 = carry_bit(i1, j1, c1)
                        r2 = add_bit(i2, j2, c2)
                        c3 = carry_bit(i2, j2, c2)
                        r3 = add_bit(i3, j3, c3)
                  in   wd(r0, r1, r2, r3) endlet
end Add_words
```

Dans les langages classiques (Pascal, Ada, etc), les définitions de types de données, de procédures ou de fonctions d'un module donné ne sont pas modifiées par les déclarations des autres modules. Ceci n'est malheureusement pas le cas des spécifications écrites en LPG1.8. En effet, de nouveaux axiomes ou déclarations d'opérateurs peuvent modifier les définitions originelles des fonctions ou des prédicats, comme il est possible d'ajouter de nouvelles structures (valeurs) aux Types de données existants. Par exemple, si on ajoute l'axiome "O == I" dans le module Add_words, le TAD bit, composé initialement de deux éléments, se trouvera réduit à un seul élément ; de même l'extension du prédicat Parity se trouve aussi modifiée (Parity deviendrait vrai pour n'importe quel mot). De telles modifications de TAD, de dénotations de fonctions et des prédicats sont dans la plupart des cas non souhaités. Nous donnons ci-dessous trois cas possibles de modification de spécifications étendues ou enrichies (voir par exemple [EM85] qui les donne sans considérer les prédicats). Nous aurons besoin pour définir ces différents cas de la notion de "Σ-réduction" d'un modèle.

Définition 5.2 : (Σ-réduction)

Soient $\Sigma = (S, \Omega, \Pi)$ et $\Sigma' = (S', \Omega', \Pi')$ deux signatures telles que $S' \subseteq S$, $\Omega' \subseteq \Omega$ et $\Pi' \subseteq \Pi$. Soit I une Σ-interprétation. On appelle **Σ'-réduction** de la Σ-interprétation I l'interprétation, notée $I_{|\Sigma'}$, obtenue à partir de I en oubliant les interprétations des sortes, fonctions et prédicats ne figurant pas dans la signature Σ'. $I_{|\Sigma'}$ est donc une Σ'-interprétation.

Etant donné une spécification SP=(Σ, E ∪ CH) enrichissement (ou extension) d'une spécification SP'=(Σ', E' ∪ CH'), pour tout E-modèle M de SP, $M_{|\Sigma'}$ est aussi un E-modèle de SP' ($\equiv_{SP} \subseteq \equiv_{SP'}$). En particulier $E\text{-}MS_{SP|\Sigma'}$ est un E-modèle de SP'. L'initialité de $E\text{-}MS_{SP'}$ dans la catégorie $E\text{-}Mod_{SP'}$ assure l'existence d'un homomorphisme unique h : $E\text{-}MS_{SP'} \rightarrow E\text{-}MS_{SP|\Sigma'}$. Nous projetons les propriétés des enrichissements (et extensions) sur cet homomorphisme.

La première propriété que nous présentons est la complétude. Intuitivement, la complétude d'une spécification SP=(Σ, E ∪ CH) par rapport à une autre spécification SP'=(Σ', E' ∪ CH') veut dire que les ensembles supports de $E\text{-}MS_{SP|\Sigma'}$ ne contiennent pas de nouvelles valeurs par rapport à celles de $E\text{-}MS_{SP'}$, et de même pour les extensions des prédicats. Cette propriété est souvent connue par "no junk". Par exemple, la spécification Add_words ne serait pas complète si l'on ajoutait la clause "Parity (wd(O,I,O,O))". Cette propriété de complétude peut être exprimée par :

Définition 5.3 : **(extension -enrichissement- complète)**
Soient SP=(Σ, E ∪ CH) et SP'=(Σ', E' ∪ CH') deux spécifications telles que SP soit une extension (ou enrichissement) de SP'. SP est une extension (ou enrichissement) **complète** de SP' ssi l'homomorphisme unique h : $E\text{-}MS_{SP'} \rightarrow E\text{-}MS_{SP|\Sigma'}$ est un épimorphisme.

La propriété duale à la complétude est la consistance. Elle correspond au fait qu'il n'y ait pas de perte d'éléments (par fusion de classes d'équivalence) dans les ensembles supports (et par conséquent dans les extensions des prédicats). Cette propriété est souvent connue par "no confusion".

Définition 5.4 : **(extension -enrichissement- consistante)**
Soient SP=(Σ, E ∪ CH) et SP'=(Σ', E' ∪ CH') deux spécifications telles que SP soit une extension (ou enrichissement) de SP'. SP est une extension (ou enrichissement) **consistante** (cohérente) avec SP' ssi l'homomorphisme unique h : $E\text{-}MS_{SP'} \rightarrow E\text{-}MS_{SP|\Sigma'}$ est un monomorphisme.

Une spécification qui est à la fois consistante et complète est dite conservative. Ce qui donne la définition suivante.

Définition 5.5 : **(extension -enrichissement- conservative)**
Soient SP=(Σ, E ∪ CH) et SP'=(Σ', E' ∪ CH') deux spécifications telles que SP soit une extension (ou enrichissement) de SP'. SP est une extension (ou enrichissement)

conservative de SP' ssi l'homomorphisme unique h : E-$MS_{SP'}$ → E-$MS_{SP|\Sigma'}$ est un isomorphisme.

Souvent ces propriétés sont caractérisées par les conditions suivantes :

Proposition 5.1 : Soient SP et SP' deux spécifications telles que SP est une extension (ou enrichissement) de SP'. SP est une extension (ou enrichissement) complète de SP' ssi

(i) $\forall\ s \in S', \forall\ t \in T_s(\Sigma), \exists\ t' \in T_s(\Sigma'), t =_{SP} t'$

(ii) $\forall\ t_i \in T_{s_i}(\Sigma'), i=1..n, P(t_1,\ldots, t_n)$ est vrai dans E-$MS_{SP} \Rightarrow P(t_1,\ldots, t_n)$ est vrai dans E-$MS_{SP'}$.

SP est est une extension (ou enrichissement) consistante avec SP' ssi

(iii) $\forall\ s \in S', \forall\ t, t' \in T_s(\Sigma'), t =_{SP} t' \Rightarrow t =_{SP'} t'$.

Preuve : (indication) voir [EM85] dans le cas des algèbres (fonctions). Concernant les extensions des prédicats, l'utilisateur ne peut que rajouter des éléments aux extensions des prédicats. En effet ceci est possible d'après la forme des clauses constituant les programmes (la négation n'étant pas permise). La non consistance ne peut donc être provoquée par l'axiomatisation des prédicats, mais elle peut l'être par les équations. Par contre, la complétude peut être violée par l'axiomatisation des prédicats. Ce qui est exprimé par la condition (ii).

Dans LPG1.8 les extensions et les enrichissements sont implicitement supposés conservatifs. D'autres langages, comme OBJ3 ou EQLOG, peuvent permettre les extensions de spécifications de façon non consistante ou non conservative. L'utilisateur signale alors dans chaque spécification le statut de celle-ci par rapport aux spécifications primitives utilisées. La vérification de telles propriétés (complétude, consistance) est en général indécidable. Dans [Gut85], Guttag fut le premier a signaler ce genre de problème dans le cadre d'un langage de programmation et a proposé des conditions syntaxiques pour vérifier la complétude. D'autres méthodes ont été proposées depuis, voir par exemple [Thi84] [KNZ87] [Com86]. Enfin, pour clore ce paragraphe sur les modules, signalons les travaux actuels en cours de développement [EM90] [Wir89] [Rey90] [ST90] dont l'objectif est de formaliser la notion de module de façon aussi indépendante que possible du langage de la logique (ou institution [GB84]) utilisé ; ceci dans le but de pouvoir disposer d'un langage de manipulation de modules (Algèbre de modules).

5.6. PROGRAMMATION PARAMETREE

La réutilisabilité comme outil de conception de programmes est un domaine de recherche très actif aujourd'hui [Gau89][GM88]. Dans ce paragraphe, nous présentons

succinctement ce que LPG1.8 offre comme facilités aidant à réutiliser les programmes. Il s'agit des spécifications paramétrées (ou génériques) [BG77] [TWW82]. Une spécification générique est une spécification ordinaire où certains symboles de sorte, de fonction ou de prédicat ont un statut particulier de paramètres formels. Une spécification générique est donc un moyen de construction d'autres spécifications; et ce par instanciation des paramètres formels. Un avantage majeur de telles spécifications, dans le cadre de la logique des clauses de Horn avec égalité, est la définition de fonctions ou de prédicats paramétrés par d'autres fonctions ou prédicats. Ceci permet alors des spécifications qui sont d'habitude l'apanage des langages d'ordre supérieur, par exemple la fonction *map* ou la relation de fermeture réflexive d'une relation binaire quelconque (voir §5.6.4).

Une fonction ou un prédicat générique est donc défini en utilisant des paramètres formels. Ces paramètres formels peuvent être des symboles de types (sorte), des symboles de fonctions ou des symboles de prédicats. La définition et l'utilisation des spécifications génériques nécessitent trois étapes dans LPG1.8 :

(i) La première étape consiste à caractériser les propriétés des paramètres formels. Cela consiste à donner des axiomes que doivent satisfaire les éventuels paramètres effectifs. Ces axiomes sont décrits dans des modules prévus à cet effet, à savoir les *Propriétés*. Nous présentons cette première étape dans la sous-section §5.6.1.

(ii) La deuxième étape est la définition proprement dite de la spécification générique. Chaque spécification générique comporte essentiellement deux parties : une partie destinée à la description des propriétés exigées sur les paramètres formels, et une deuxième partie définissant des algorithmes (fonctions ou prédicats) génériques. Cette étape est illustrée dans la sous-section §5.6.2.

(iii) La troisième étape, appelée "instanciation", correspond à l'utilisation des spécifications génériques pour engendrer de nouvelles spécifications "ordinaires". Cela consiste à remplacer les paramètres formels d'une spécification générique par des paramètres effectifs. Nous présentons cette dernière étape dans la sous-section §5.6.3

L'instanciation met en jeu les deux premières étapes. Elle utilise en effet les propriétés des paramètres formels pour s'assurer de la validité des paramètres effectifs, et la deuxième étape pour produire une spécification non générique. Notons que notre présentation de la généricité est informelle. Une sémantique rigoureuse (utilisant la théorie des catégories) est en dehors de notre propos. Le lecteur intéressé par de telles études peut consulter par exemple [Bid89] [EM85] [EM90] [Rey87] [Huf89].

5.6.1. Spécification des Paramètres Formels : les *Propriétés*

Si l'on considère un algorithme générique de tri, paramétré par l'ensemble des éléments à trier et par la relation d'ordre sur ces éléments, soit $\leq$, l'obtention d'un programme de tri

sur un domaine fixé est possible seulement si l'opération effective remplaçant le paramètre ≤ constitue bien un ordre sur les éléments. Autrement dit, tout paramètre effectif associé à la relation d'ordre ≤ doit satisfaire les axiomes de l'ordre. D'où le besoin d'expliciter les propriétés requises sur les paramètres effectifs. Dans LPG1.8, ces propriétés sont exprimées, par des axiomes de la logique des clauses de Horn avec égalité, sur les paramètres formels dans des modules appelés *Propriétés*. Le but de ces modules est de limiter les possibilités d'instanciation, évitant en cela des programmes dépourvus de sens. Les Propriétés constituent donc des documents sur les paramètres formels. Ce genre de document, qui aide à la compréhension des programmes et leur utilisation, n'a pas son équivalent dans les langages classiques permettant quelque forme de généricité comme "LISP" ou ADA. Schématiquement, une propriété est une spécification SP = (Σ, C).

Exemple : Nous donnons six exemples de modules de Propriétés.

La première Propriété, `Set`, décrit tous les ensembles possibles (types abstraits $T(\Sigma)_{\equiv}$). Une telle Propriété peut être exigée pour la définition, par exemple, des séquences, des arbres étiquetés, etc.

```
property   Set
     sorts
          s
end Set
```

Ci-dessous, nous donnons l'exemple d'une propriété qui décrit les relations binaires. Nous l'utiliserons pour la définition de la fermeture réflexive d'une relation (§5.6.4).

```
property   Binary_relation
     sorts
          s
     predicates
          P : (s, s)
end Binary_relation
```

La propriété `Equality` spécifie les propriétés pour qu'un opérateur à résultat booléen soit un opérateur d' "égalité". Nous supposons que le type abstrait des booléens, avec ses opérateurs habituels, est défini par ailleurs.

```
property   Equality
     sorts
          s
     operators
          eq : (s, s) -> bool
     variables
          x, y, z : s
```

```
      equations
            1 : eq(x, x) == true
            2 : eq(x, y) == eq(y, x)
            3 : eq(x, y) and eq(y, z) and not(eq(x, z)) == false
end Equality
```

La propriété `Total_order` spécifie les propriétés pour qu'un opérateur booléen soit un opérateur "d'ordre". Nous supposons que le type abstrait des booléens, avec ses opérateurs habituels, est défini par ailleurs.

```
property  Total_order
      sorts
            s
      operators
            ≤, = : (s, s) -> bool
      variables
            x, y, z : s
      equations
            1 : x≤y or y ≤x == true       -- total
            2 : x ≤ x == true             -- reflexivity
            3 : x = y == x≤y and y ≤x      -- antisymmetry
            4 : (x≤y and y≤z) and not(x≤z) == false  -- transitivity
end Total_order
```

Nous donnons enfin l'exemple de la propriété d'une relation d'équivalence.

```
property  Equivalence
      sorts
            s
      predicates
            Q : (s, s)
      variables
            x, y, z : s
      clauses
            1: Q(x, x)
            2: Q(x, y) <== Q(y, x)
            3: Q(x, z) <== Q(x, y), Q(y, z)
end Equivalence
```

Informellement, la sémantique d'une propriété SP est toute la classe E-Mod$_{SP}$. En pratique, la signature Σ peut contenir des sortes ou des symboles que l'on veut interpréter "initialement". Par exemple, dans la propriété `Total_order`, la sorte bool est utilisée ainsi que les opérations `true`, `false`, `or`, `and` et `not`; il est clair que l'interprétation du type abstrait bool est sous-entendu initiale (sinon que devient la spécification si true == false?). Pour tenir compte de ces possibilités, a été introduite la notion de spécifications avec contraintes [BG80] [Rei80].

5.6.2. Spécification Paramétrée

Une spécification paramétrée est un couple SPP = (SPF, SP) [EM85] où SPF est la spécification des paramètres formels, appelée la propriété exigée, et SP est la spécification but qui utilise les paramètres formels. Dans LPG1.8, les modules Type et Enrichissement peuvent être génériques. les paramètres formels d'un module générique sont mentionnés au début de celui-ci, avec le nom de la Propriété qui les caractérise. Nous considérons ci-dessous une spécification générique des séquences.

```
type Sequence
     requires
          Set [elem]
     sorts
          seq
     constructors
          nil : -> seq[elem]
          cons : (elem, seq[elem]) -> seq[elem]
     operators
          length : seq[elem] -> nat
     variables
          x : elem
          u : seq[elem]
     equations
          1 : length(nil) == 0
          2 : length(cons(x, u)) == s(length(u))
end Sequence
```

La spécification `Sequence` est paramétrée par l'ensemble des éléments des séquences. Ceci se traduit syntaxiquement par l'expression

"requires

`Set[elem]"`

Cette expression signifie informellement qu'à partir de n'importe quel modèle dans $E\text{-}Mod_{Set}$, c'est-à-dire un ensemble A, on pourra définir le type de données des séquences des éléments de A. Le nom "elem" dans l'expression "`Set[elem]`" n'est qu'un renommage de la sorte s figurant dans la signature de la Propriété `Set`. En général, dans LPG1.8, un seul module de Propriété doit caractériser les paramètres formels d'un module générique. Ainsi, on trouve dans l'entête de toute spécification générique une expression de la forme :

" **requires**

P [s_1,...,s_i **operators** o_1,...,o_j **predicates** p_1,...,p_k] "

Dans cette expression, on distingue le nom de la propriété exigée, soit P, et un renommage de la signature de cette propriété , ($\Sigma = (S, \Omega, \Pi)$). Ce renommage[1] est une correspondance

[1]Ce renommage est un morphisme de signature. Le morphisme de spécification associé dénote un foncteur libre fortement persistant [Rey87].

univoque entre la signature déduite de [s_1,...,s_i **operators** o_1,...,o_j **predicates** p_1,...,p_k] et la signature de la propriété P. L'ordre d'écriture des symboles de la signature Σ fixe ce renommage. Par exemple, l'expression `Total_order[t operators inf, eq]` figurant dans une unité générique indique la correspondance suivante (rappelons que s, $\leq$ et = sont les symboles de la Propriété `Total_order`) :

$s \rightarrow t$

$\leq \rightarrow$ inf

$= \rightarrow$ eq

Considérons maintenant la définition d'un opérateur booléen définissant l'égalité de deux séquences, soit `eq_seq: (seq[elem], seq[elem]) -> bool`. La définition de cet opérateur nécessite forcément un opérateur booléen exprimant l'égalité sur les éléments. Par conséquent, la définition de l'opérateur générique `eq_seq` ne peut être donnée dans le module `Sequence`, puisque l'opérateur formel `eq` ne figure pas dans la signature de la Propriété `Set`. Il nous faut donc définir un nouveau module générique enrichissant le module `Sequence`. Ce nouveau module, `On_seq`, exige la Propriété `Equality` qui spécifie les contraintes requises sur l'opérateur `eq`. Syntaxiquement, on écrit :

```
enrichment On_seq
      requires
            Equality [elem operators eq]
      operators
            eq_seq : (seq[elem], seq[elem]) -> bool
      variables
            x, y : elem
            s, u : seq[elem]
      equations
            1 : eq_seq(nil, nil) == true
            2 : eq_seq(cons(x, s), nil) == false
            3 : eq_seq(nil, cons(x, s)) == false
            4 : eq_seq(cons(x,s), cons(y,u)) == eq(x,y) and eq_seq(s,u)
end On_seq
```

5.6.3. Instanciation

Le rôle de l'instanciation est d'obtenir des définitions effectives à partir des définitions génériques. Ceci est réalisé en substituant aux paramètres formels d'une définition générique des paramètres effectifs. Les paramètres effectifs devront de plus constituer un modèle de la propriété exigée. Considérons par exemple la spécification ci-dessus `On_seq`. Pour obtenir un opérateur d'égalité sur les séquences de naturels, il suffit d'instancier l'opérateur générique eq_seq par un modèle de sa propriété exigée, à savoir la Propriété Equality.

Dans LPG1.8, un modèle d'une propriété P est noté par

P [s_1,...,s_i **operators** o_1,...,o_j **predicates** p_1,...,p_k]

où les s_m, o_m et p_m sont respectivement des symboles de sortes effectives comme nat, bool, etc; des symboles d'opérateurs effectifs et des symboles de prédicats effectifs. La correspondance entre les symboles de la propriété P et les symboles effectifs s'effectue comme indiqué dans le paragraphe précédent. Quant à la vérification (preuves) que l'interprétation déclarée constitue bien un modèle, elle peut se faire à l'aide d'un démonstrateur (voir [Dra89]).

En considérant l'opération = : (nat, nat) -> bool définie dans exemple 5.1, Equality [nat **operators** =] est bien un modèle de la propriété Equality. L'opérateur d'égalité sur les séquences de naturels est noté dans LPG1.8 par l'expression "eq_seq.Equality [nat **operators** =]". En général, l'instance d'un opérateur (ou prédicat) générique f par un modèle m est notée par "f.m". Cette notation n'est pas toujours facile à utiliser. Il suffit de considérer un exemple pour s'en convaincre. Soit à définir un opérateur d'égalité sur les séquences de séquences de naturels. Une possibilité de définition de cet opérateur serait la suivante :

```
eq_seq.Equality[seq.Set[nat]operators eq_seq.Equality[natoperators =]]
```

où l'opérateur générique eq_seq est instancié par le modèle de la propriété Equality constitué de l'ensemble seq.Set[nat] des séquences de naturels et l'opérateur effectif d'égalité sur les séquences de naturels vu précédemment, à savoir "eq_seq.Equality[nat **operators** =]". Cette lourdeur dans l'écriture nécessite forcément des simplifications syntaxiques. Une solution courante est le renommage des symboles instanciés. Une deuxième solution, qui existe dans LPG1.8, consiste à rendre certaines instanciations implicites (c'est-à-dire, les modèles des propriétés exigées sont déduits automatiquement à partir des profils des paramètres effectifs -opérateurs ou prédicats-). Cette deuxième possibilité repose sur une hiérarchisation des modules des Propriétés. Cette hiérarchisation est définie par des relations d'ordre sémantique reflétant des morphismes de théorie entre les dénotations des propriétés. A ce sujet, nous renvoyons le lecteur à [Ber83] [BDE87][Huf 89][Rey87].

5.6.4. Exemples

Après ce bref survol sur la généricité présente en LPG1.8, nous donnons dans ce sous-paragraphe quelques exemples de fonctions et de prédicats souvent définis dans les langages d'ordre supérieur.

Fonction *map*

Nous commençons par la définition de la fonction *map*. Cette fonction applique une fonction qu'elle reçoit en paramètre à tous les éléments d'une liste. Dans LPG1.8, le module de définition de la fonction générique *map* doit exiger une propriété dont les modèles sont des structures constitués de deux ensembles supports, soient A et B, et d'une application, soit f:A→ B; ce qui donne

```
property   Mapping
      sorts
            s1, s2
      operators
            m : s1 -> s2
end Mapping
```

```
enrichment   Map
      requires
            Mapping [t1, t2 operators f]
      operators
            map : seq[t1] -> seq[t2]
      variables
            x: t1
            s : seq[t1]
      equations
            1 : map (nil) == nil
            2 : map (cons (x, s)) == cons(f(x), map(s))
end Map
```

Graphe et Congruence définis par une application

Ce deuxième exemple défini le graphe d'une application. Un point d'un tel graphe est de la forme (x, f(x)). Nous définissons par la même occasion la relation de congruence *cong* induite par une application f : A→ B sur son domaine A. Le module suivant exige donc la Propriété Mapping définie ci-dessus.

```
enrichment   Graph_Cong
      requires
            Mapping [t1, t2 operators f]
      predicates
            graph : (t1, t2)
            cong : (t1, t1)
      variables
            x, y : t1
      clauses
            1 : cong(x, y) <== f(x) == f(y)
            2 : graph (x, f(x))
end Graph_Cong
```

Fermeture réflexive d'une relation binaire

Nous définissons dans cet exemple le prédicat générique Reflex. L'instance de ce prédicat, par une relation binaire P, a pour extension la fermeture reflexive de la relation P. D'où le programme suivant :

```
enrichment   Reflexivity_closure
      requires
            Binary_relation [t predicates P]
      predicates
            Reflex: (t, t)
      variables
            x, y : t
      clauses
            1 : Reflex(x, y) <== P(x, y)
            2 : Reflex(x, x)
end Reflexivity_closure
```

Intersection et Union de deux relations binaires

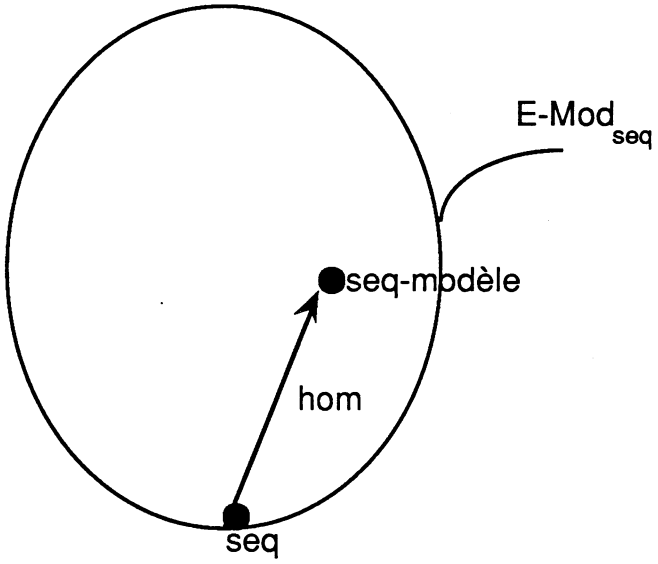
L'exemple ci-dessous définit l'intersection et l'union de deux relations binaires définies sur un même ensemble. Pour cela, nous commençons par définir une propriété spécifiant les structures avec deux relations binaires et un ensemble support.

```
property   Two_Relations
      sorts
            s
      predicates
            r1, r2 : (s, s)
end Two_Relations
```

```
enrichment   Inter_Union
      requires
            Two_Relations [t predicates R, Q]
      predicates
            Inter, Union : (t, t)
      variables
            x, y : t
      clauses
            1: Inter(x, y) <== R(x, y), Q(x, y)
            2: Union(x, y) <== R(x, y)
            3: Union(x, y) <== Q(x, y)
end Inter_Union
```

Définition de l'homomorphisme entre la structure des séquences et les seq-structures Par seq-structure nous entendons une structure constituée de deux ensembles supports A et B, une constante n dans B -constante correspondant à nil- ainsi qu'une application f:(A, B) → B. Cette fonction correspond à l'opération cons des

séquences. L'opérateur générique hom défini pour chaque seq-structure <A,B; n, f> est l'homomorphisme unique entre les séquences de A et cette structure.




```
property   Mod_seq
      sorts
            A, B
      operators
            n : -> B
            f : (A, B) -> B
end Mod_seq
```

```
enrichment   Hom
      requires
            Mod_seq [t1, t2 operators e, add]
      operators
            hom : seq[t1] -> t2
      variables
            x : t1
            s : seq[t1]
      equations
            1 : hom(nil) == e
            2 : hom(cons(x, s)) == add(x, hom(s))
end Hom
```

Une utilisation de l'opérateur générique hom

Dans ce dernier exemple, nous illustrons l'opérateur hom sur deux exemples célèbres, à savoir les calculs des sommes et des produits des éléments d'une liste. L'écriture hom[0,+], ci-dessous, est une écriture LPG qui simplifie l'expression :

```
"hom.Mod-seq[nat,nat operators 0,+]".
```

```
enrichment   Sum_Product
      operators
            sum, product : seq[nat] -> nat
      variables
            s : seq[nat]
      equations
            1 : sum(s) == hom[0,+] (s)
            2 : product(s) == hom[1,*] (s)
end Sum_Product
```

5.7. NOTE SUR L'IMPLANTATION DE L'INTERPRETE

Dans ce paragraphe nous décrivons l'interprète de LPG1.8 que nous avons implanté. Cette description sera simplifiée afin d'éviter les détails fastidieux de l'implantation.

L'interprète de LPG1.8 s'utilise à la manière des langages "logiques", c'est-à-dire que le rôle de l'interprète est de résoudre des buts (définition 2.1). Chaque but est résolu par rapport à une spécification non générique SP=(Σ, E $\cup$ EI $\cup$ C) où Σ contient les symboles nécessaires pour résoudre le but considéré (les symboles de Σ sont effectifs) et E, EI et C contiennent respectivement les définitions des fonctions, des théorèmes (équations utilisées pour la simplification des calculs) et les définitions des prédicats. Notre interprète s'appuie sur les résultats des chapitres précédents, notamment le théorème 2.7 et le corollaire 3.2. Autrement dit, l'interprète implante la règle dite SLDEI-résolution où le calcul des EI-unificateurs est réalisé par une variante de la surréduction. Nous discutons respectivement dans les deux sections suivantes (§5.7.1 et §5.7.2) les implantations de la SLDEI-résolution et de la Surréduction avec stratégie linéaire ; nous exhiberons dans ces deux cas un arbre étiqueté à parcourir. La fusion de ces deux arbres en un seul donne l'arbre étiqueté par les buts que parcourt notre interprète. Nous présentons ce troisième arbre, ainsi que les choix pris pour le développement et le parcours d'un tel arbre, dans une troisième section §5.7.3.

5.7.1. Implantation de la SLDEI-résolution

Nous commençons cette section par la définition d'un arbre de SLDEI-résolution, associé à un but B à résoudre et une spécification SP. Cet arbre, qui est étiqueté par les buts, reflète toutes les SLDEI-dérivations possibles issues de la clause de but "$\Leftarrow$ B" en utilisant une stratégie de Résolution donnée (définition 2.20). La définition ci-dessous est suggérée par le théorème 2.7. Nous nous permettons dans la suite de confondre un nœud avec son contenu (étiquette).

Définition 5.6 : (arbre de SLDEI-résolution)

Soient SP=(Σ, C) une spécification, B= $A_1 \ldots A_n$ un but et SR une stratégie de Résolution. Nous définissons l'arbre de SLDEI-résolution pour le but B par :

(i) L'arbre de SLDEI-résolution est étiqueté par des buts (éventuellement le but vide, noté []),

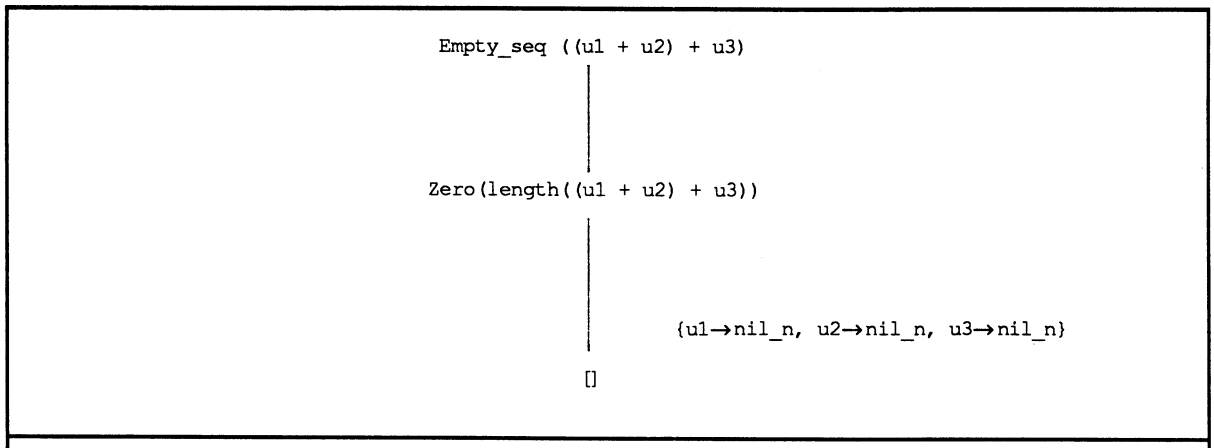
(ii) La racine de l'arbre de SLDEI-résolution est le but B,

(iii) Chaque nœud de l'arbre de SLDEI-résolution, soit D, admet comme nœuds fils les buts D' tels que la clause ⇐ D' soit une EI-résolvante de la clause ⇐ D , en utilisant la stratégie de Résolution SR et une substitution dans l'ensemble complet de EI-unificateurs utilisé (cf. définition 2.16).

Exemple 5.6 : Nous illustrons les différents arbres de ce paragraphe sur l'exemple ci-dessous. Dans cet exemple, nous définissons les séquences de naturels. Les opérateurs `nil-n` et "`.`" sont les constructeurs habituels des séquences ; `length(u)` calcule la longueur d'une séquence `u`; l'opération + concatène deux séquences (cette opération est souvent appelée *append*) ; le prédicat `Empty_seq` est vrai sur la seule séquence `nil_n` (la définition la plus simple de ce prédicat est le fait "`Empty_seq(nil_n)` " seul, nous en donnons une autre définition ci-après pour une meilleure illustration). Le prédicat `Zero` a pour extension le singleton {0}. Nous supposons bien sûr que les naturels sont définis par ailleurs à l'aide de leurs constructeurs classiques `0:-> nat` et `succ:nat -> nat`.

```
type  Seq_nat
      sorts
            seq_nat
      constructors
            nil_n : -> seq_nat
            . : (nat, seq_nat) -> seq_nat
      operators
            length : seq_nat -> nat
            + : (seq_nat, seq_nat) -> seq_nat
      predicates
            Empty_seq : seq_nat
            Zero : nat
      variables
            i : nat
            u, v, w : seq_nat
      equations
            1: length (nil_n) == 0
            2: length (i.u) == succ(length(u))
            3: nil_n + u == u
            4: (i.u) + v == i.(u+v)
      clauses
            5: Empty_seq (u) <== Zero(length(u))
            6: Zero (0)
      theorems
            7: (u +v)+w == u +(v+w)
            8: u + nil_n == u
end Seq_nat
```

Nous nous proposons d'illustrer les arbres de SLDEI-résolution sur le but "Empty_seq ((u1 + u2) + u3)".



Exemple d'arbre de SLDEI-résolution

Remarques 5.1 :

(1) Tout nœud contenant le but vide est une feuille dans un arbre de SLDEI-résolution.

(2) Dans la définition 5.6, nous ne précisons pas l'ordre des fils d'un nœud. Nous le préciserons lors de l'implantation de l'interprète.

(3) On peut associer à un but donné plusieurs arbres de SLDEI-résolution différents. Ceci est dû à la non unicité des ensembles complets de EI-unificateurs. Notons qu'un nœud dans un arbre de SLDEI-résolution, qui n'est pas une feuille, n'admet pas forcément comme fils les buts qui correspondent à tous les EI-résolvantes possibles. Les fils d'un nœud sont déterminés par l'ensemble complet de EI-unificateurs utilisé (Un seul ensemble complet de EI-unificateurs est utilisé par clause intervenant dans le calcul des EI-résolvantes).

(4) Tout chemin entre la racine et une feuille contenant le but vide correspond à une SLDEI-réfutation. La restriction aux variables du but considéré de la composition des substitutions utilisées le long d'un tel chemin est une solution pour ce but.

(5) Tout algorithme parcourant un arbre de SLDEI-résolution pour un but B de façon équitable (aucun nœud n'est oublié à jamais), et en déduisant des solutions à chaque visite d'une feuille contenant le but vide, est un algorithme qui calcule un ensemble complet de solutions pour le but B (cf. Théorème 2.7).

5.7.2. Implantation de la Surréduction avec stratégie linéaire

Nous nous intéressons maintenant à une implantation correspondant au corollaire 3.2, autrement dit, à la résolution d'équations par "surréduction via une stratégie" (ci-après notée

SL-surréduction[1]) dans un contexte qui est une spécification de la forme SP = (Σ, E $\cup$ EI $\cup$ CH) où (cf. chapitre 3) l'ensemble E est défini par un système de réécriture canonique SR'=(Σ, R) et EI, ensemble d'équations valides dans E-MS_{SP}, est défini par un système de réécriture SR"=(Σ, RI) tel que le système SR=(Σ, R $\cup$ RI) soit nœthérien. Comme pour la SLDEI-Résolution, et suivant l'énoncé du corollaire 3.2, nous définissons un arbre étiqueté correspondant aux dérivations de la SL-surréduction utilisant une stratégie SS de choix de positions de surréduction.

Définition 5.7 : (arbre de SL-surréduction)

Soit SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification où l'ensemble E est défini par un système de réécriture canonique SR'=(Σ, R) et EI est défini par un système de réécriture SR"=(Σ, RI) tel que le système SR=(Σ, R $\cup$ RI) soit nœthérien. Soient SS une stratégie de surréduction et t == t' une équation à résoudre (c'est-à-dire, t et t' deux termes à EI-unifier). Nous définissons l'arbre de SL-surréduction pour l'équation t == t' par :

(i) L'arbre de SL-surréduction est étiqueté par des équations,

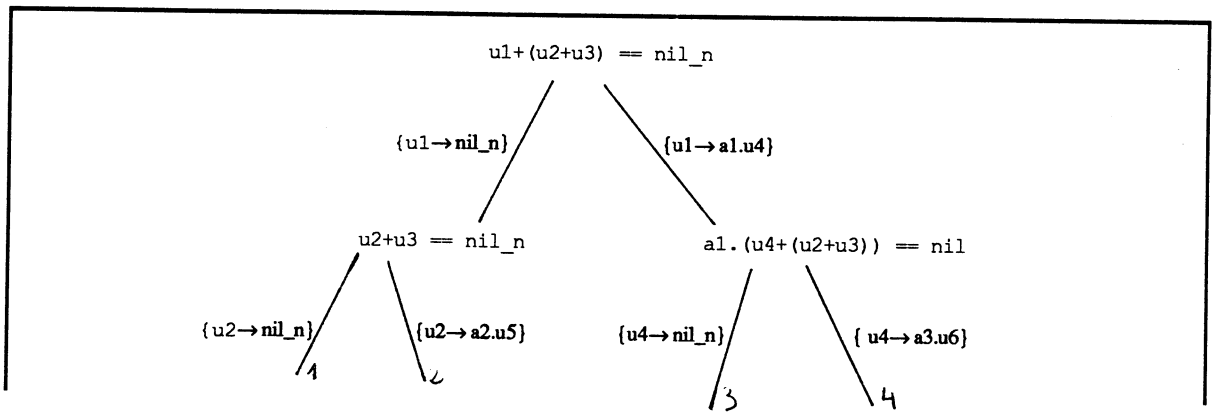
(ii) La racine de l'arbre est l'équation $t!^i == t'!^i$ (une forme normale inductive de t == t', cf. définition 3.22).

(ii) Chaque nœud de l'arbre, soit "$t_1 == t_2$", admet comme nœuds fils toutes les équations "$t_3 == t_4$" telles que il existe une dérivation de surréduction inductivement normale:

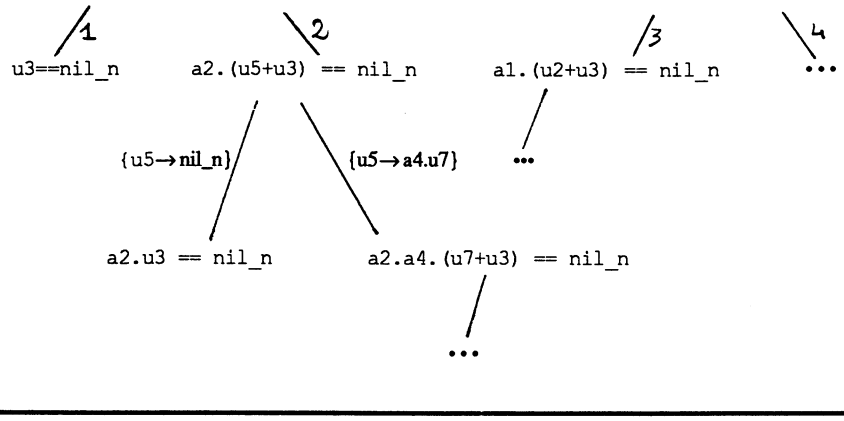
$$t_1 == t_2 \text{ --INN-->}_{[u,\ldots]} t_3 == t_4$$

avec la condition : u = SS(t_1==t_2).

Exemple 5.7 : Nous considérons la spécification Seq_nat de l'exemple 5.6. Nous donnons l'arbre de SL-surréduction relatif au but "(u1+u2)+u3 == nil_n". la stratégie de surréduction utilisée est la stratégie left-most-outer-most. la racine de cet arbre est "u1+(u2+u3) == nil_n" qui est la forme normale inductive obtenue en utilisant le théorème inductif décrivant l'associativité de l'opération +.



[1]SL-surréduction est mis pour surréduction **L**inéaire avec fonction de **S**élection de position



Exemple d'arbre de SL-surréduction

L'arbre ci-dessus est infini. Chaque "•••" désigne un arbre infini. Le chemin le plus à gauche est le seul chemin qui donne la solution escomptée.

Remarques 5.2 :

(1) chaque nœud d'un arbre de SL-surréduction contient une seule équation dont les deux membres sont sous forme normale inductive.

(2) Un nœud d'un arbre de SL-surréduction, t == t', est une feuille si SS(t == t') est indéfini. Si les deux termes d'une feuille t == t' sont unifiables, le chemin allant de la racine de l'arbre de SL-surréduction à cette feuille correspond à une dérivation de surréduction donnant lieu à une solution pour l'équation initiale.

(3) Comme pour l'arbre de SLDEI-Résolution, nous ne précisons pas l'ordre des fils d'un nœud dans un arbre de SL-surréduction. Nous précisons ce paramètre dans la section suivante.

(4) Pour une équation t == t', une spécification SP = (Σ, E $\cup$ EI $\cup$ C) et une stratégie de surréduction SS, l'arbre de SL-surréduction est unique à l'ordre des fils près.

(5) Lorsque la stratégie de surréduction SS est uniforme, tout algorithme parcourant l'arbre de SL-surréduction, pour une équation t == t', de façon équitable (aucun nœud n'est oublié à jamais), et en déduisant des solutions à chaque visite d'une feuille dont les deux membres de l'équation sont unifiables, est un algorithme qui calcule un ensemble complet de EI-unificateurs (cf. corollaire 3.2).

5.7.3. Implantation de l'interprète

Notre interprète réalise la SLDEI-résolution. C'est donc un algorithme qui parcourt au moins un arbre de SLDEI-résolution. Or, le seul point qui ne soit pas assez précis dans le développement de cet arbre est le calcul des ensembles complets de EI-unificateurs. L'arbre de SLDEI-résolution devient unique, à l'ordre des fils près, dès lors que l'on fixe la méthode de calcul des ensembles complets de EI-unificateurs. Nous réalisons ce calcul par

SL-surréduction, comme nous l'avons montré dans la section précédente. Par conséquent, notre algorithme doit développer et parcourir des arbres de SLDEI-résolution pour la résolution des buts, et des arbres de SL-surréduction pour calculer des ensembles complets de EI-unificateurs à chaque pas de SLDEI-résolution. Une implantation naïve de l'interprète peut consister à développer un arbre de SLDEI-résolution correspondant au but initial et autant[1] d'arbres de SL-surréduction que de pas de SLDEI-résolution. Une telle implantation, qui sépare les développements des différents arbres, est certes maladroite. Notre implantation effective développe un seul arbre qui est en fait la combinaison des arbres de SLDEI-résolution et de SL-surréduction nécessaires pour la résolution du but initial. Nous appelons "arbre combiné" un tel arbre.

Définition 5.8 : (arbre combiné)

Soit SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification où l'ensemble E est défini par un système de réécriture canonique SR'=(Σ, R) et EI est défini par un système de réécriture SR"=(Σ, RI) tel que le système SR=(Σ, R $\cup$ RI) soit nœthérien. Soient SS une stratégie de Surréduction et SR une stratégie de Résolution. Nous définissons l'arbre combiné associé à un but B= $A_1 \ldots A_n$ par :

(i) L'arbre combiné est étiqueté par des buts (éventuellement vide),

(ii) La racine de l'arbre combiné contient une forme normale inductive du but initial (soit $B!^i$),

(iii) Soit $D = D_1 \ldots D_p \ldots D_k$ avec $k \geq 1$ le but d'un nœud N de l'arbre combiné et D_p l'atome choisie par la stratégie de Résolution (SR(D) = p). Les fils du nœud N sont calculés comme suit :

(a) Cas où D_p est une équation. Soit $D_p = t == t'$.

(a1) Si $SS(D_p)$ n'est pas défini et les termes t et t' ne sont pas unifiables, alors le nœud N est une feuille (N n'a pas de fils),

(a2) Si $SS(D_p)$ n'est pas défini et les termes t et t' sont unifiables via l'unificateur minimal σ, alors le nœud N a un seul fils qui contient le but $\sigma(D_1 \ldots D_{p-1} D_{p+1} \ldots D_k)!^i$,

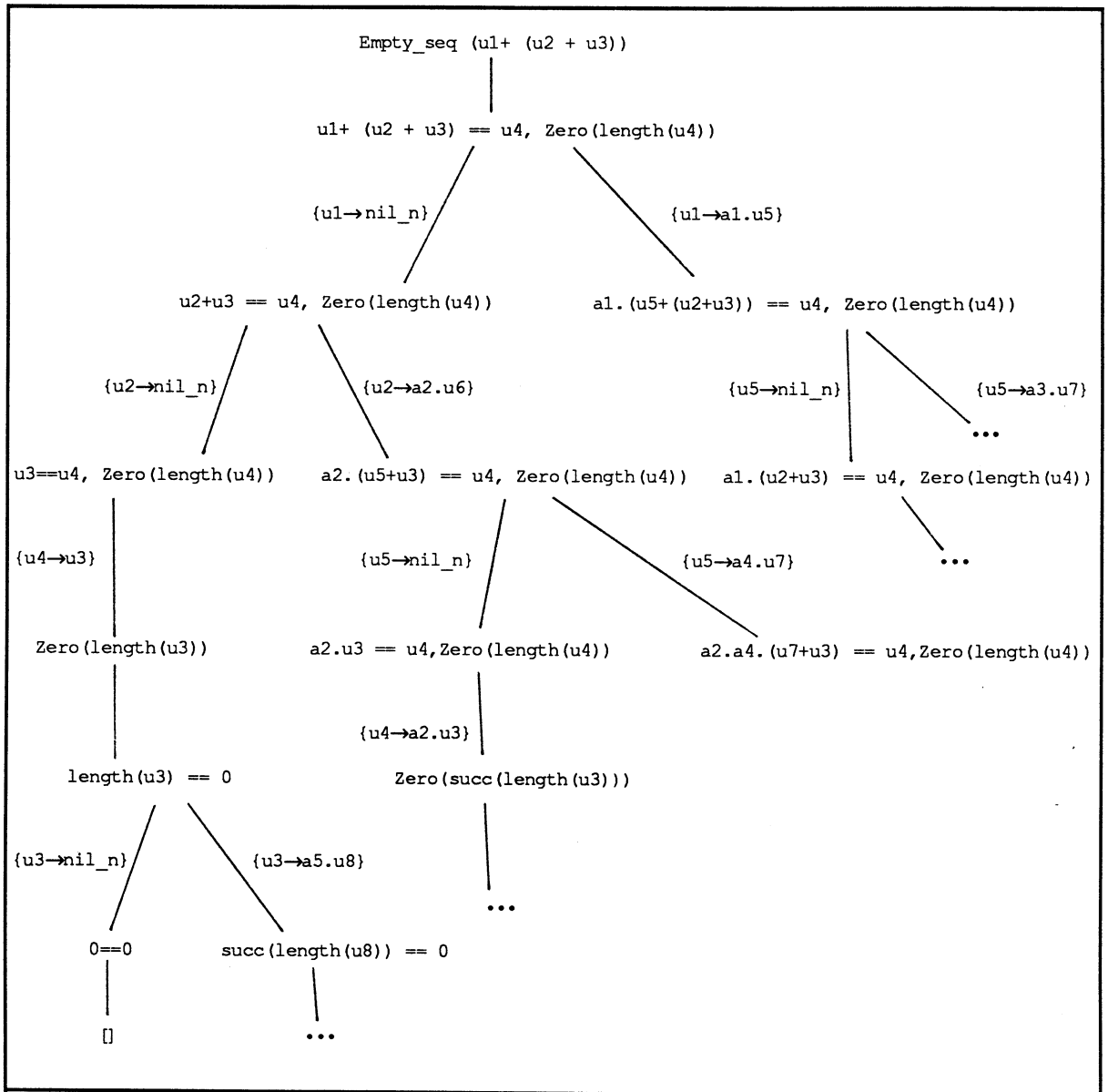
(a3) Cas $SS(D_p)$ est défini. Soient $u = SS(D_p)$, I l'ensemble des indices des règles utilisées pour la surréduction à la position u et σ^j la substitution utilisée dans le pas de dérivation de "surréduction inductivement normale" (cf. définition 3.23) en utilisant la règle j dans R ($j \in I$) : D_p --INN-->$_{[u, j, \sigma j]}$ D_p^j . Dans ce cas le nombre de fils du nœud N est card(I). Pour chaque j dans I, le nœud N^j contenant le but $\sigma^j(D_1 \ldots D_{p-1} D_p^j D_{p+1} \ldots D_k)!^i$ est un fils du nœud N.

---

[1]Plus précisément, le nombre d'arbres de SL-surréduction est *de l'ordre du* nombre de pas de SLDEI-résolution.

(b) Cas où D_p n'est pas une équation. Soit $D_p = P(t_1,...,t_n)$. Pour toute clause dans CH de la forme $P(s_1,...,s_n) \Leftarrow Q_1(...)...Q_m(...)$ avec $m \geq 0$; N a un fils qui contient le but $(D_1...D_{p-1}\ (t_1 == s_1)...(t_n == s_n)\ Q_1(...)...Q_m(...)\ D_{p+1}...D_k)!^i$.

Exemple 5.8 : Nous considérons ici la même spécification ainsi que le même but que dans l'exemple 5.6.



Exemple d'arbre Combiné

L'arbre ci-dessus est infini. Chaque "•••" désigne un arbre infini. Le chemin le plus à gauche est le seul chemin qui donne la solution escomptée.

Remarques 5.3 :

(1) Les nœuds feuilles contiennent

- soit le but vide
- soit un but $B = D_1 \ldots (t == t') \ldots D_k$ tel que l'atome sélectionné est $t == t'$, $SS(t == t')$ n'est pas défini et les termes t et t' ne sont pas unifiables (condition (a1)).
- soit un but $B = D_1 \ldots P(t_1,\ldots,t_n) \ldots D_k$ tel que l'atome sélectionné est $P(t_1,\ldots,t_n)$ et le prédicat P n'est pas défini.

(2) L'arbre combiné associé à un but donné est unique à l'ordre des fils près.

Notation : Soit $B = A_1A_2 \ldots A_n$ un but. Nous noterons par $\mathbf{B \vdash_{i,\sigma} B'}$ le fait que le but B' soit un fils du but B dans un arbre combiné. i est le rang de l'atome sélectionné par la stratégie SR ($i = SR(B)$) et σ est la substitution utilisée. Si l'atome sélectionné A_i n'est pas une équation, la substitution σ est l'identité.

Nous étudions maintenant quelques propriétés des arbres combinés, notamment la cohérence et la complétude des solutions déduites d'un tel arbre. Pour cela, nous commençons par définir les notions de "chemin de solution" et de "solution effective". Dans la suite, nous supposons donnée une spécification $SP=(\Sigma, E \cup EI \cup CH)$ satisfaisant les conditions de la définition 5.8 ; c'est-à-dire que l'ensemble E est défini par un système de réécriture canonique $SR'=(\Sigma, R)$ et EI est défini par un système de réécriture $SR''=(\Sigma, RI)$ tel que le système $SR=(\Sigma, R \cup RI)$ soit nœthérien.

Définition 5.9 : **(chemin de solution)**

Soit AC un arbre combiné associé à un but B. Un chemin de solution est tout chemin dans l'arbre AC allant de la racine à une feuille contenant le but vide :

$$B \vdash_{i_0,\sigma_0} B_1 \vdash_{i_1,\sigma_1} B_2 \ldots \vdash_{i_{n-1},\sigma_{n-1}} B_n \vdash_{i_n,\sigma_n} []$$

Définition 5.10 : **(solution effective)**

Soient AC un arbre combiné associé à un but B, V l'ensemble des variables figurant dans B et "$B \vdash_{i_0,\sigma_0} B_1 \vdash_{i_1,\sigma_1} B_2 \ldots \vdash_{i_{n-1},\sigma_{n-1}} B_n \vdash_{i_n,\sigma_n} []$" un chemin de solution. La composition $\sigma_n\sigma_{n-1}\ldots\sigma_0|_V$ est dite solution effective pour le but B.

Un interprète (pour un langage comme LPG1.8) est un algorithme de parcours de graphe. Ce graphe est un arbre combiné AC déterminé de façon univoque par les données de l'interprète suivantes :

- une spécification $SP = (\Sigma, E \cup EI \cup CH)$,
- un but B à résoudre,
- une stratégie de Résolution SR et

- une stratégie de surréduction SS.

L'interprète fournit en sortie toutes les substitutions effectives qu'il aura détectées lors de son parcours de l'arbre AC. Un interprète peut donner toutes les solutions effectives si c'est un algorithme *complet*. Cependant, certains algorithmes de parcours peuvent manquer des solutions (par exemple les parcours en profondeur d'abord). Pour une discussion sur les algorithmes de parcours de graphe voir, par exemple, [Nil80] [Knu69] [BB84]. Aussi, l'essentiel pour nous est de vérifier la cohérence des solutions effectives ainsi que la complétude de l'ensemble de ces solutions. Nous montrons ces résultats dans le cas où la stratégie de Résolution SR est "left-most", c'est-à-dire la stratégie qui choisit toujours l'atome le plus à gauche dans un but, $SR(A_1A_2\ldots A_n) = 1$. Nous étudions la stratégie de Résolution "left-most" pour deux raisons. La première est que c'est cette stratégie que nous avons implantée. La deuxième raison est motivée par la facilité des preuves de cohérence et de complétude, qui sont des applications des résultats des chapitres précédents. La preuve dans le cas général est technique, mais ne pose pas de difficulté particulière.

Nous étudions maintenant la cohérence des solutions effectives. Pour cela nous commençons par énoncer les quelques propositions suivantes

Proposition 5.2 : Soient $SP = (\Sigma, E \cup EI \cup CH)$ une spécification, B un but, SR la stratégie de Résolution "left-most" et SS une stratégie uniforme de Surréduction.

(1) Si $B = (t == t')$, alors l'arbre combiné associé à B est l'arbre de SL-surréduction pour $t == t'$. L'ensemble des solutions effectives est un ensemble complet de EI-unificateurs pour t et t'.

(2) Si $B = (t==t')A_2\ldots A_n$, alors pour tout chemin de solution ch , il existe un indice q tel que

$$ch = B \vdash_{1,\sigma_0} B_1 \vdash_{1,\sigma_1} B_2 \ldots \vdash_{1,\sigma_q} B_{q+1} \ldots \vdash_{1,\sigma_{n-1}} B_n \vdash_{1,\sigma_n} []$$

avec $\sigma_q\ldots\sigma_0|_{vars(B)}$ un EI-unificateur de t et t'; et $B_{q+1} = \sigma_q\ldots\sigma_0\ (A_2\ldots A_n)!^i$.

(3) Si $B = (t_1 == s_1)\ldots(t_n == s_n)$, alors toute solution effective pour B, soit θ, est solution pour le système d'équations $\{t_i == s_i \mid i = 1\ldots n\}$. L'ensemble des solutions effectives est un ensemble complet de solutions pour le but B.

(4) Si $B = (t_1 == s_1)\ldots(t_n == s_n)A_2\ldots A_n$, alors pour tout chemin de solution ch, il existe un indice q tel que

$$ch = B \vdash_{1,\sigma_0} B_1 \vdash_{1,\sigma_1} B_2 \ldots \vdash_{1,\sigma_q} B_{q+1} \ldots \vdash_{1,\sigma_{n-1}} B_n \vdash_{1,\sigma_n} []$$

avec $\sigma_q\ldots\sigma_0|_{vars(B)}$ une solution du système $\{t_i == s_i \mid i = 1\ldots n\}$; et $B_{q+1} = \sigma_q\ldots\sigma_0\ (A_2\ldots A_n)!^i$.

Preuve :

(1) Se déduit de la remarque5.2(5) et le corollaire 3.2

(2) Conséquence de l'utilisation de la stratégie de Résolution "left-most", de l'assertion (1) et la règle (a3) de la définition 5.8.

(3) θ est solution du système $\{t_i == s_i \mid i = 1...n\}$ est une conséquence de l'assertion (2). La complétude de l'ensemble des solutions effectives est une conséquence de la complétude de la SL-surréduction.

(4) Se déduit de (3). (4) est une généralisation de l'assertion (2) aux systèmes d'équations.

Lemme 5.1 : (cohérence) Soient SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification, B = $A_1A_2...A_n$ un but, SR la stratégie de Résolution "left-most", SS une stratégie uniforme de Surréduction et AC l'arbre combiné associé à B. Alors, toute solution effective θ est une solution cohérente ($E\text{-}MS_{SP} \models \theta(B)$).

Preuve : par induction sur la longueur des chemins de solution.

Cas de base : Soit c un chemin de solution de longueur 1 dans l'arbre AC. Le chemin c est forcément de la forme $t == t' \vdash_{1,\sigma} []$. σ est l'unificateur minimal de t et t'. σ est donc une solution cohérente.

Hypothèse d'induction : Toute solution effective correspondant à un chemin de solution de longueur inférieure ou égale à n est une solution cohérente.

Considérons un chemin de solution de longueur (n+1). Ce chemin est de la forme :

(i) $\quad B \vdash_{1,\sigma_0} B_1 \vdash_{1,\sigma_1} B_2... \vdash_{1,\sigma_{n-1}} B_n \vdash_{1,\sigma_n} []$

Deux cas sont possibles pour passer de B à B_1. Ces deux cas correspondent aux règles de déduction (a3) et (b) de la définition 5.8.

Cas (a3). Cela signifie que l'atome sélectionné, à savoir A_1, est une équation soit $A_1 = t == t'$. D'après la proposition 5.2(2), il existe un indice q, $0 \le q \le n$ tel que le chemin (i) puisse s'écrire sous la forme

$$B \vdash_{A_0,\sigma_0} B_1 \vdash_{A_1,\sigma_1} B_2... \vdash_{A_q,\sigma_q} B_{q+1}... \vdash_{A_{n-1},\sigma_{n-1}} B_n \vdash_{A_n,\sigma_n} []$$

avec $\sigma_q...\sigma_0|_{vars(B)}$ un EI-unificateur de t et t'.

Si q = n, cela veut dire que B est réduit à une seule équation. Donc la solution effective $\sigma_n\sigma_{n-1}...\sigma_0|_{vars(B)}$ est cohérente.

Si $0 \le q < n$, par hypothèse d'induction, on a : $\sigma_n\sigma_{n-1}...\sigma_{q+1}|_{vars(B_{q+1})}$ est une solution cohérente pour le but $B_{q+1} = \sigma_q...\sigma_0(A_2...A_n)$[i]. La substitution $\sigma_q...\sigma_0|_{vars(B)}$ étant un EI-unificateur de t et t', la cohérence de la substitution effective $\sigma_n\sigma_{n-1}...\sigma_0|_{vars(B)}$ est une conséquence de la cohérence du calcul par la règle SLDEI-résolution.

Cas (b). Dans ce cas, l'atome sélectionné A_1 n'est pas une équation et est de la forme $P(t_1,...,t_n)$. D'après la règle (b) de la définition 8.5 le but B_1 est déduit en utilisant une clause de la forme $P(s_1,...,s_n) \Leftarrow Q_1(...)...Q_m(...)$ avec $m \ge 0$. On a donc

$$B_1 = [(t_1 == s_1)\ldots(t_n == s_n)Q_1(\ldots)\ldots Q_m(\ldots)A_2\ldots A_n]!^i$$

D'après la proposition 5.2(4), il existe un indice q , $1 \leq q \leq n$ tel que le chemin (i) puisse s'écrire sous la forme

$$B \vdash_{1,\sigma_0} B_1 \vdash_{1,\sigma_1} B_2 \ldots \vdash_{1,\sigma_q} B_{q+1} \ldots \vdash_{1,\sigma_{n-1}} B_n \vdash_{1,\sigma_n} []$$

avec q l'indice indiquant le nombre de pas nécessaires pour le calcul d'une solution pour le système d'équations $\{(t_1 == s_1)\ldots(t_n == s_n)\}$; cette solution est donc $\sigma_q\sigma_{q-1}\ldots\sigma_1|_{vars(B_1)}$.

Si q = n, cela veut dire que la substitution $\sigma_n\sigma_{n-1}\ldots\sigma_1|_{vars(B_1)}$ est une solution du système d'équations $\{(t_1 == s_1)\ldots(t_n == s_n)\}$ et par suite la substitution $\sigma_n\sigma_{n-1}\ldots\sigma_1\sigma_0|_{vars(B)}$ est une solution cohérente pour B puisqu'elle correspond à la substitution utilisée pour un pas de SLDEI-résolution (rappelons que σ_0 vaut l'identité).

Si $1 \leq q < n$, la substitution $\sigma_q\ldots\sigma_1|_{vars(B_1)}$ étant une solution du système $\{(t_1 == s_1)\ldots(t_n == s_n)\}$, Le but B_{q+1} est une EI-résolvante du but B. Par hypothèse d'induction, $\sigma_n\sigma_{n-1}\ldots\sigma_{q+1}|_{vars(B_{q+1})}$ est une solution cohérente pour le but B_{q+1}. D'après la cohérence du calcul par la règle SLDEI-résolution, on déduit la cohérence de la substitution effective $\sigma_n\sigma_{n-1}\ldots\sigma_0|_{vars(B)}$.cqfd.

Nous venons de voir que toute solution effective pour un but est une solution cohérente par rapport à la spécification utilisée. Nous nous intéressons maintenant à la complétude de l'ensemble des solutions effectives. Pour cela nous établissons le lien entre les chemins de solution et les dérivations de SLDEI-résolution (toujours dans le cas de la stratégie "left-most").

Proposition 5.3 : Soient SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification, B un but, SR la stratégie de Résolution "left-most", SS une stratégie uniforme de Surréduction et AC l'arbre combiné associé à B. Alors, pour tout pas de SLDEI-dérivation $B \xrightarrow{SLDEI}_{[\ldots,\theta]} B'$, où θ est un EI-unificateur calculé par Surréduction en utilisant la stratégie SS, il existe un chemin dans l'arbre AC :

$$B!^i \vdash_{1,\sigma_0} B_1 \vdash_{1,\sigma_1} B_2 \ldots B_q \vdash_{1,\sigma_q} B'!^i$$

tel que $\sigma_n\sigma_{n-1}\ldots\sigma_0|_{vars(B)} = \theta$.

Preuve : Soit $B = A_1A_2\ldots A_n$. Nous distinguons les deux cas possibles pour l'atome A_1.

Cas A_1 est une équation. Soit A_1 = t==t'. On a par hypothèse $\theta \in Sol^{SS}_{INN}(R \cup RI, t, t')$ et $B' = \theta(A_2\ldots A_n)$. Or, d'après la règle (a3) de la définition 5.8, qui réalise la surréduction via la stratégie SS, et la proposition 5.2(2), nous déduisons l'existence d'un chemin $B!^i \vdash_{1,\sigma_0} B_1 \vdash_{1,\sigma_1} B_2\ldots B_q \vdash_{1,\sigma_q} B'!^i$ tel que $\sigma_n\sigma_{n-1}\ldots\sigma_0|_{vars(B)} = \theta$.

Cas A_1 n'est pas une équation. A_1 est donc de la forme $P(t_1,\ldots,t_n)$. Soit $P(s_1,\ldots,s_n) \Leftarrow Q_1(\ldots)\ldots Q_k(\ldots)$ la clause utilisée pour le pas de SLDEI-résolution. La substitution θ est un

EI-unificateur du système d'équations $\{(t_1 == s_1)\ldots(t_n == s_n)\}$. D'après la proposition 5.2 (4), il existe un chemin dans AC

$$B!^i \vdash_{1,\sigma_0} B_1 \vdash_{1,\sigma_1} B_2 \ldots B_q \vdash_{1,\sigma_q} B'!^i$$

tel que $\sigma_n\sigma_{n-1}\ldots\sigma_0|_{vars(B)} = \theta$. cqfd.

Corollaire 5.1 : Soient SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification, B un but, SR la stratégie de Résolution "left-most" et SS une stratégie uniforme de Surréduction. Alors, pour toute solution calculée θ correspondant à une SLDEI-réfutation du but B où les calculs des EI-unificateurs sont réalisés par surréduction en utilisant la stratégie SS, il existe un chemin de solution dans l'arbre combiné associé à B,

$$B!^i \vdash_{1,\sigma_0} B_1 \vdash_{1,\sigma_1} \ldots \vdash_{1,\sigma_{n-1}} B_n \vdash_{1,\sigma_n} []$$

tel que $\sigma_n\sigma_{n-1}\ldots\sigma_0|_{vars(B)} = \theta$.

Preuve : Conséquence de la proposition 5.3. La preuve est par induction sur la longueur des SLDEI-réfutations.

Lemme 5.2 : (complétude) Soient SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification, B un but, SR la stratégie de Résolution "left-most" et SS une stratégie uniforme de Surréduction. Alors, pour toute solution fermée θ pour un but B (E-$MS_{SP} \models \theta(B)$), il existe une solution effective σ telle que $\sigma \leq^i_{SP} \theta$ [vars(B)].

Preuve : conséquence de la complétude de la SLDEI-résolution et du corollaire 5.1.

Après la données des lemmes 5.1 et 5.2, nous pouvons enfin énoncer le

Théorème 5.1 : Soient SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification, B un but, W un ensemble de variables contenant V = vars(B), SR la stratégie de Résolution "left-most", SS une stratégie uniforme de Surréduction et AC un arbre combiné associé à B. L'ensemble de toutes les solutions effectives de l'arbre AC est un ensemble complet de solutions pour B en dehors de W.

Nous avons ainsi montré que l'ensemble des solutions effectives d'un arbre combiné, développé à l'aide d'une stratégie uniforme de Surréduction et la stratégie de Résolution left-most, est un ensemble complet de solutions pour le but considéré. De plus, l'ensemble des solutions effectives correspond à l'ensemble $Sol^{SR}_{SLDEI\text{-résolution}}$ (SP, B) (cf. corollaire 5.1) où la stratégie de Résolution SR est la stratégie left-most, et où le calcul des EI-unificateurs est réalisé par Surréduction en utilisant une stratégie uniforme.

Dans la définition d'un arbre combiné (définition 5.8) nous avons représenté toutes les branches indiquées par le théorème 2.7 et le corollaire 3.2, sans aucun souci de simplification

de l'arbre. Plusieurs règles peuvent en fait être utilisées pour diminuer la taille d'un arbre combiné. Nous utilisons quatre règles de simplification dans notre implantation. Nous introduisons ces règles dans la nouvelle définition de l'arbre combiné que nous appelons *arbre combiné simplifié*. Ces nouvelles règles sont (a0), (a2'), (a2'') et (a2''').

Définition 5.11 : (arbre combiné simplifié)

Soit SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification où l'ensemble E est défini par un système de réécriture canonique SR'=(Σ, R) et EI est défini par un système de réécriture SR''=(Σ, RI) tel que le système SR=(Σ, R $\cup$ RI) soit nœthérien. Soient SS une stratégie de Surréduction et SR une stratégie de Résolution. Nous définissons l'arbre combiné simplifié associé à un but B= $A_1 \ldots A_n$ par :

(i) L'arbre combiné simplifié est étiqueté par des buts (éventuellement vide),

(ii) La racine de l'arbre combiné simplifié contient une forme normale inductive du but initial (soit $B!^i$),

(iii) Soit D = $D_1 \ldots D_p \ldots D_k$ avec $k \geq 1$ le but d'un nœud N de l'arbre combiné et D_p l'atome choisie par la stratégie de Résolution (SR(D) = p). Les fils du nœud N sont calculés comme suit :

(a) Cas où D_p est une équation. Soit D_p= t == t'. Les fils du nœud N sont calculés par l'une des sept règles suivantes. Leur application se fait dans l'ordre (a0)<...<(a3) :

(a0) Si t = t' (identité syntaxique), alors le nœud N a un seul fils qui contient le but $D_1 \ldots D_{p-1} D_{p+1} \ldots D_k$,

(a1) Si $SS(D_p)$ n'est pas défini et les termes t et t' ne sont pas unifiables, alors le nœud N est une feuille (N n'a pas de fils),

(a2) Si $SS(D_p)$ n'est pas défini et les termes t et t' sont unifiables via l'unificateur minimal σ, alors le nœud N a un seul fils qui contient le but $\sigma(D_1 \ldots D_{p-1} D_{p+1} \ldots D_k)!^i$,

(a2') Si D_p = x == t, ou bien D_p = t == x, avec x une variable telle que $x \notin$ vars(t), alors le nœud N a un seul fils qui contient le but $\{x \rightarrow t\}(D_1 \ldots D_{p-1} D_{p+1} \ldots D_k)!^i$,

(a2'') Si $D_p = c(t_1,\ldots,t_n) == c(s_1,\ldots,s_n)$ avec c un opérateur constructeur (c ne peut disparaître de la racine des deux termes), alors le nœud N a un seul fils qui contient le but $(D_1 \ldots D_{p-1} (t_1 == s_1) \ldots (t_n == s_n) D_{p+1} \ldots D_k)!^i$.

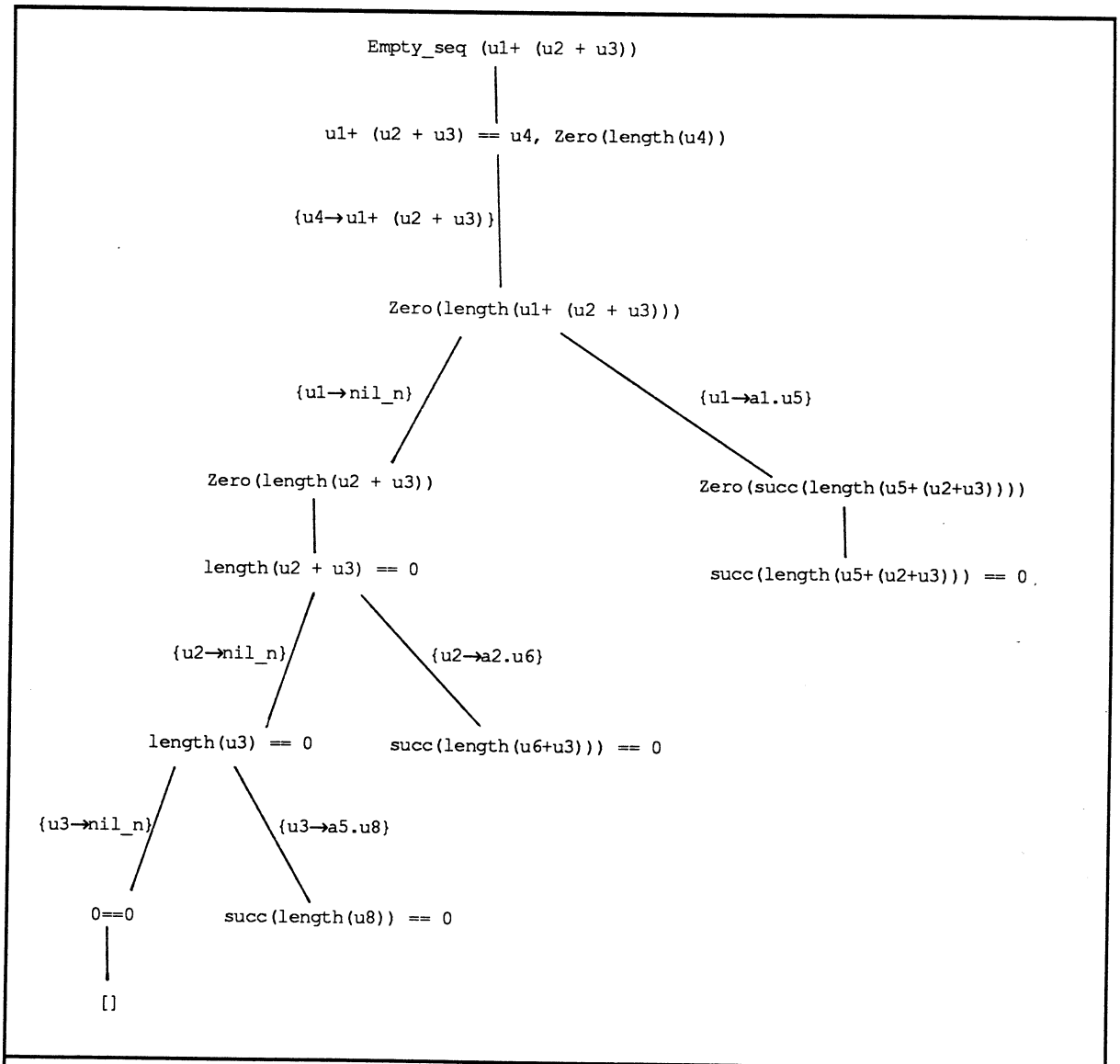
(a2''') Si $D_p = c(t_1,\ldots,t_n) == d(s_1,\ldots,s_m)$ avec c et d deux opérateurs constructeurs différents ($c \neq d$, et c et d ne peuvent pas disparaître de la racine des deux termes), alors le nœud N est une feuille dans l'arbre,

(a3) Cas $SS(D_p)$ est défini. Soient u = $SS(D_p)$, I l'ensemble des indices des règles utilisées pour la surréduction à la position u et σ^j la substitution utilisée dans le pas de dérivation de "surréduction inductivement normale" (cf. définition 3.23) en utilisant la règle j

dans R ($j \in I$) : D_p --INN-->$_{[u, j, \sigma^j]}$ D^j_p . Dans ce cas, le nombre de fils du nœud N est card(I). Pour chaque j dans I, le nœud N^j contenant le but $\sigma^j(D_1 \ldots D_{p-1}\ D^j_p\ D_{p+1} \ldots D_k)!^i$ est un fils du nœud N.

(b) Cas où D_p n'est pas une équation. Soit $D_p = P(t_1,\ldots,t_n)$. Pour toute clause dans CH de la forme $P(s_1,\ldots,s_n) \Leftarrow Q_1(\ldots)\ldots Q_m(\ldots)$ avec $m \geq 0$; N a un fils qui contient le but $(D_1 \ldots D_{p-1}\ (t_1 == s_1)\ldots(t_n == s_n)\ Q_1(\ldots)\ldots Q_m(\ldots)\ D_{p+1}\ldots D_k)!^i$.

Exemple 5.9 : Nous considérons ici la même spécification ainsi que le même but que dans l'exemple 5.6. On remarquera que cette fois-ci l'arbre est fini, contrairement à celui de l'exemple 5.8.



Exemple d'arbre Combiné Simplifié

Remarque 5.4 : On peut facilement vérifier la cohérence des nouvelles règles de simplification. En effet, les règles (a0), (a2') et (a2") transforment un but B en un but B' tel que tout ensemble de solutions complet pour B' est aussi un ensemble de solutions complet pour B (la réciproque est fausse). La règle (a2"') détecte certains buts insatisfaisables et élague ainsi certaines branches inutiles dans l'arbre combiné.

Le théorème 5.1 reste valable dans le cas des arbres combinés simplifiés. On a alors

Corollaire 5.2 : Soient SP = (Σ, E $\cup$ EI $\cup$ CH) une spécification, B un but, W un ensemble de variables contenant V = vars(B), SR la stratégie de Résolution "left-most", SS une stratégie uniforme de Surréduction et AC l'arbre combiné simplifié associé à B. L'ensemble de toutes les solutions effectives de l'arbre AC est un ensemble complet de solutions pour B en dehors de W.

Preuve : se déduit de la cohérence des règles de simplification (remarque 5.4) et du théorème 5.1.

Remarque 5.5 : Le théorème 5.1 et le corollaire 5.2 montrent que les règles de déductions de la définition 5.8, respectivement de la définition 5.11, constituent un calcul complet pour les spécifications SP de la forme SP = (Σ, E $\cup$ EI $\cup$ CH).

Pour clore ce paragraphe sur l'implantation il nous suffit de préciser les paramètres nécessaires pour la description de l'arbre combiné simplifié qui est développé par notre interprète, ainsi que le parcours de cet arbre. Ces paramètres sont : la stratégie de Résolution, la stratégie de surréduction, la stratégie de réduction (cette information est intéressante puisqu'elle précise le calcul des formes normales inductives), l'ordre de considération des clauses pour la SLDEI-résolution (cette information précise l'ordre des fils de certains nœuds -règles (b) de la définition 5.11), l'ordre de considération des règles de réécriture (cette information précise l'ordre des fils de certains nœuds -règles (a3) de la définition 5.11) et enfin la stratégie de parcours de l'arbre. Les choix que nous avons faits sont les suivants :

1- la stratégie SR de la SLDEI-résolution : left-most
2- la stratégie SS de la SL-surréduction : inner-most left-most
3- la stratégie de réduction : outer-most left-most
4- l'ordre de considération des clauses : de haut en bas
5- l'ordre de considération des équations : de haut en bas
6- le parcours de l'arbre combiné : en profondeur, de gauche à droite

Ces choix ont été effectués de façon arbitraire, le but essentiel étant d'expérimenter la faisabilité de l'approche étudiée dans ce mémoire, pour intégrer les langages fonctionnels et prédicatifs.

5.8. AUTRES TRAVAUX

Dans ce paragraphe nous donnons un bref panorama sur les propositions de langages logico-fonctionnels. Ce survol n'est en aucun cas complet.

Robinson et Sibert ont été les premiers a proposer un langage logico-fonctionnel. Leur langage Loglisp [RS81], est une implantation de Prolog dans un environnement Lisp. Le but de leur proposition est de faire communiquer les deux langages Lisp et Prolog (Prolog avec une implantation breadth-first). Cette contribution aux langages logico-fonctionnels basée sur la combinaison des langages à la lisp et le langage des clauses de Horn, a été suivie par plusieurs auteurs, nous citons à titre d'exemple les langages Qlog [Kom82], LeFun [AN89], Qute [MT86] et APPLOG [Coh86]. Ces langages souffrent d'un manque de sémantique rigoureuse. En tout cas, leur expressivité (langages d'ordre supérieur) sacrifie malheureusement la complétude de leur sémantique opérationnelle.

Nous rassemblons dans une deuxième classe les langages logico-fonctionnels basés sur la logique du premier ordre avec égalité. Dans ce registre, on trouve par exemple le langage TABLOG [MW86]. Ce langage est à notre connaissance le seul (en tant que langage de programmation) qui englobe toute la logique du premier ordre avec égalité. Sa sémantique opérationnelle est basée sur le système formel "Deductive-tableau" de Manna et Waldinger [MW80]. Les autres langages de cette classe se basent sur la logique des clauses de Horn avec égalité, comme EQLOG [GM84], LEAF[BBLM86], TEL [Smo88] ou LPG1.8 [BE86]. Ces langages sont très proches d'un point de vue syntaxique, quoique EQLOG et TEL offrent en prime la possibilité d'utiliser les sous-sortes, mais diffèrent par leurs sémantiques opérationnelles respectives. L'interêt de ces langages est qu'ils incorporent, comme l'approche précédente, les deux notions de prédicat et de fonction; ils admettent de plus des sémantiques relativement bien établies.

La troisième classe de langages logico-fonctionnels que nous distinguons est constituée des langages algébriques. Ces langages, comme SLOG [Fri85], RAP [Hus85] ou BABEL [MR89], se basent sur la logique conditionnelle. Nous les considérons en tant que langages logico-fonctionnels car leur sémantique opérationnelle, basée sur la surréduction conditionnelle, leur confère quasiment les mêmes possibilités que les langages basés sur la logique des clauses de Horn avec égalité. Leur seul inconvénient par rapport aux langages de la classe précédente réside dans le fait que les prédicats (ou relations) soient définies par leurs fonctions caractéristiques (sous forme de fonctions booléennes).

Une autre approche des langages logico-fonctionnels est manifestée dans les langages PROLOGIII ou Chip. De tels langages sont en fait des instances de la programmation logique

avec contraintes [JL87]. Ces langages trouvent leur place parmi des langages logico-fonctionnels puisqu'ils permettent l'utilisation de certaines fonctions (limitées en nombre) dans les définitions de prédicats. Cette approche est très pragmatique. Le succès de ces langages constitue un encouragement aux différentes recherches sur l'intégration des langages fonctionnels et logiques.

Le lecteur intéressé par quelques autres propositions peut consulter [DL86]. La course vers un langage logico-fonctionnel n'est pas encore terminée. Des efforts[1] considérables continuent à être déployés pour mieux maîtriser les problèmes qui ont surgi à la suite des investigations effectuées ces dernières années. Un des problèmes majeurs qui mérite d'être étudié est celui de l'arrêt (ou contrôle). En effet ce problème est un obstacle qu'il faut franchir afin que cette nouvelle génération de langages de programmation puisse être effective. Dans la conclusion de ce mémoire, nous donnons des perspectives qui vont dans ce sens.

APPENDICE (trace d'une session de LPG1.8)

Nous présentons ci-dessous une courte session de LPG1.8, version[2] implantée à l'Université Catholique de Louvain la nœuve (UCL). Chaque session de LPG1.8 comporte des résolutions de buts. Un but est présenté par une conjonction d'atomes suivie du symbole "?". Les variables figurant dans les buts doivent être déclarées à la volée, pendant la session.

Dans cette session, nous nous proposons de résoudre les buts suivants :

```
ordered ([0, i, 2]) ?
ordered ([0,2, i]) ?
permutation ([0,i,2], [j, k]+u)?
u + v = [1, 2] == false ?
sum([3, 7, i]) >= 12 == true ?
graph[sort]([3,8,5,3],u)?
Inter[graph[id],graph[rev]] (u, v)?
Queens ([4,3,2,1], pos) ?
```

Le prédicat `ordered(u)` est vrai si et seulement si u est une séquence ordonnée (cf. exemple 5.4). Le prédicat `permutation(u, v)` est vrai si et seulement si la séquence v est une permutation de la séquence u (voir le prédicat permut dans exemple 5.3). L'opérateur

---

[1]comme en témoigne le projet européen Intégration [DR90] ou le futur Dagstuhl-Seminar sur l'intégration de la programmation fonctionnel et logique, qui aura lieu en mars 1991.

[2]Cette version est la traduction en ADA d'une ancienne version, réalisée à Grenoble, écrite en PL/I.

`sum` ainsi que les prédicats `graph` et `Inter` sont définis dans le §5.6.4. Le prédicat `Queens` est défini dans l'exemple 5.3. L'opération + définie sur les séquences représente la concaténation, voir exemple 5.6. L'opération `sort` trie une séquence. Enfin les opérations `id` et `rev` définissent respectivement l'identité et l'inverse d'une séquence. Le symbole <+ qui apparaît dans la résolution du but "**u + v = [1, 2] == false** ?" désigne le symbole de concaténation d'un élément à gauche d'une séquence (c'est le traditionnel *cons* de Lisp). Le but "**Inter[graph[id],graph[rev]] (u, v)**?" désigne l'ensemble des couples (u, u) tels que u est un palindrome. Ce but a les mêmes solutions que **graph[rev](u, u).**

Les variables comme v336 sont engendrées par l'interprète. Ce dernier précise la sorte des variables par <variable-name> : <sorte-name>. En gras sont représentées le interventions de l'utilisateur.

```
Lpg 1.8 ? e

{ 1}  variables i, j, k : nat
                  u, v : seq[nat]
      end

{ 1}  ordered ([0, i, 2]) ?
i == 0
more solutions? y
i == 1
more solutions? y
i == 2
more solutions? y

No more solutions.

{ 1}  ordered ([0,2, i]) ?
for all v336 : nat
i == succ(succ(v336))
more solutions? y

No more solutions.

{ 1}  permutation ([0,i,2], [j, k]+u)?
for all k : nat
i == k
u == [2]
j == 0
more solutions? y
for all j : nat
i == j
u == [2]
k == 0
more solutions? y
for all j : nat
i == j
u == [0]
k == 2
more solutions? y
for all i : nat
u == [i]
k == 2
j == 0
more solutions? y
for all i : nat
u == [i]
k == 0
j == 2
more solutions? y
for all k : nat
i == k
```

```
u == [0]
j == 2
more solutions? y

No more solutions.

{ 1}   u + v = [1, 2] == false ?
v == nil
u == nil
more solutions? y
v == [1]
u == nil
more solutions? y
for all v467 : seq[nat]
for all v466 : nat
v == (1 <+ (2 <+ (v466 <+ v467)))
u == nil
more solutions? y
for all v463 : seq[nat]
v == (1 <+ (0 <+ v463))
u == nil
more solutions? y
for all v463 : seq[nat]
v == (1 <+ (1 <+ v463))
u == nil
more solutions? y
for all v514 : nat
for all v463 : seq[nat]
v == (1 <+ (succ(succ(succ(v514))) <+ v463))
u == nil
more solutions? y
for all v450 : seq[nat]
v == (0 <+ v450)
u == nil
more solutions? y
for all v450 : seq[nat]
for all v530 : nat
v == (succ(succ(v530)) <+ v450)
u == nil
more solutions? y
v == nil
u == [1]
more solutions? y
for all v557 : nat
for all v558 : seq[nat]
v == (2 <+ (v557 <+ v558))
u == [1]
more solutions? y
for all v554 : seq[nat]
v == (0 <+ v554)
u == [1]
more solutions? y
for all v554 : seq[nat]
v == (1 <+ v554)
u == [1]
more solutions? y
for all v554 : seq[nat]
for all v609 : nat
v == (succ(succ(succ(v609))) <+ v554)
u == [1]
more solutions? y
for all v626 : seq[nat]
for all v625 : nat
v == (v625 <+ v626)
u == [1,2]
more solutions? y
for all v654 : seq[nat]
for all v653 : nat
v == v
u == (1 <+ (2 <+ (v653 <+ v654)))
more solutions? y
for all v622 : seq[nat]
v == v
u == (1 <+ (0 <+ v622))
more solutions? y
for all v622 : seq[nat]
v == v
```

```
u == (1 <+ (1 <+ v622))
more solutions? y
for all v700 : nat
for all v622 : seq[nat]
v == v
u == (1 <+ (succ(succ(succ(v700))) <+ v622))
more solutions? y
for all v537 : seq[nat]
v == v
u == (0 <+ v537)
more solutions? y
for all v537 : seq[nat]
for all v716 : nat
v == v
u == (succ(succ(v716)) <+ v537)
more solutions? y

No more solutions.
{ 1}  sum([3, 7, i]) >= 12 == true ?
for all v734 : nat
i == succ(succ(v734))
more solutions? y

No more solutions.
{ 1}  graph[sort]([3,8,5,3],u)?
u == [3,3,5,8]
more solutions? y

No more solutions.

{ 1}  Inter[graph[id],graph[rev]]  (u,  v)?
u == nil
v == nil
more solutions? y
for all v749 : nat
u == [v749]
v == [v749]
more solutions? y
for all v752 : nat
u == [v752,v752]
v == [v752,v752]
more solutions? y
for all v752 : nat
for all v754 : nat
u == [v754,v752,v754]
v == [v754,v752,v754]
more solutions? y
for all v754 : nat
for all v756 : nat
u == [v756,v754,v754,v756]
v == [v756,v754,v754,v756]
more solutions? y
for all v756 : nat
for all v754 : nat
for all v758 : nat
u == [v758,v756,v754,v756,v758]
v == [v758,v756,v754,v756,v758]
more solutions? n


{ 1}  variables pos : seq_dot end

{ 1}  Queens ([4,3,2,1], pos) ?
pos == cons_d(d(4,2),cons_d(d(3,4),cons_d(d(2,1),cons_d(d(1,3),nil_d))))
more solutions? y
pos == cons_d(d(4,3),cons_d(d(3,1),cons_d(d(2,4),cons_d(d(1,2),nil_d))))
more solutions? y

No more solutions.
{ 1}  end
Lpg 1.8 ?
```

Conclusion

Résultats

Les études menées par les logiciens sur les langages de définition de fonctions et de prédicats ont facilité la conception des langages de programmation logico-fonctionnels fondés sur la logique des clauses de Horn avec égalité. Cependant, la mise en œuvre de tels langages reste l'objet de multiples recherches. Dans ce mémoire, nous avons considéré un sous-langage des clauses de Horn avec égalité : les fonctions y sont définies équationnellement et les prédicats, autres que l'égalité, sont définis par des clauses de Horn avec égalité. Un programme dans ce langage est une collection de définitions de fonctions et de prédicats. La sémantique que nous associons à un programme est son modèle initial.

Nous nous sommes surtout intéressé dans ce travail au calcul pour ce langage. Ainsi, en tenant compte de la sémantique adoptée, nous avons proposé la règle SLDEI-résolution comme règle unique de calcul. Cette règle est obtenue à partir de la règle SLD-résolution en remplaçant l'unification syntaxique par la résolution d'équations (EI-unification) dans le modèle initial défini par le programme considéré. Nous avons prouvé la cohérence, la complétude ainsi que la complétude forte de cette règle. Notons que la sémantique des programmes adoptée dans ce mémoire est différente de celle habituellement prise dans le domaine de la démonstration automatique. Cette différence se reflète, bien sûr, dans les définitions de la cohérence et de la complétude.

La résolution d'équations, nécessaire dans la réalisation de la règle SLDEI-résolution, est effectuée par des techniques de réécriture et notamment la relation de surréduction. Le choix de ces techniques a surtout été influencé par l'état de l'art en la matière. En effet, puisque les programmes considérés permettent la définition de nouveaux types de donnés, une méthode *générale* de résolution d'équations s'impose. Pour cela, nous avons opté pour la surréduction. Ce choix a été motivé par l'utilisation d'équations orientables (règles de réécriture) dans les programmes. D'autres algorithmes de résolution d'équations pourraient être utilisés comme par exemple celui proposé dans [GS88]. Nous n'avons pas opté pour cette possibilité pour des raisons d'efficacité. Ainsi, en partant des travaux pionniers sur la résolution d'équations par surréduction effectués par Fay et Hullot, nous avons cherché à les améliorer et en particulier à éliminer certaines redondances présentes dans les algorithmes

qu'ils ont proposés. Pour cela, nous avons utilisé des stratégies de choix de position de surréduction. Cependant, l'utilisation de ces stratégies ne donne pas toujours lieu à des algorithmes complets de EI-unification. Dans un premier temps, nous avons donné des conditions suffisantes sur les stratégies de surréduction qui assurent la complétude de la résolution d'équations. Ces conditions généralise le résultat particulier de Fribourg concernant les stratégies de surréduction inner-most sur les programmes dits *lhs-innermost* [Fri85].

Nous avons ensuite défini une classe de programmes que nous appelons *uniformes*. Un programme est dit uniforme si et seulement si toutes les stratégies possibles de surréduction satisfont les conditions suffisantes déjà mentionnées, ce qui assure la complétude de la résolution d'équations en présence de n'importe quelle stratégie de surréduction. Nous avons donné aussi un moyen pour décider l'uniformité d'un programme.

Enfin, nous avons proposé des conditions syntaxiques qui assurent l'uniformité d'un programme. Ces conditions reposent en particulier sur la notion clef de *non sous-unification stricte* des membres gauches des règles de réécriture. Bien que ces conditions peuvent paraître trop restrictives au premier abord, nous avons donné une procédure qui transforme les programmes à la Huet-Hullot [HH80] ou les programmes lhs-innermost de Fribourg [Fri85] en des programmes satisfaisant les critères que nous avons développés.

Ces résultats ont été appliqués avec succès au langage LPG1.8 [BE86] [BDE87b]. Parmi les approches existantes pour l'intégration des langages fonctionnels et logiques que nous avons mentionnées dans l'introduction, celle présentée dans ce mémoire, et qui a été suivie, à des détails près, par quelques autres auteurs, nous paraît l'une des plus prometteuse, avec celle des langages logiques avec contraintes. Elles jouissent en effet toutes les deux de fondements mathématiques rigoureux, aussi bien d'un point de vue sémantique "déclarative" que sémantique opérationnelle. Les langages fondés sur la logique conditionnelle ne sont que des cas particuliers de l'approche présentée ici.

Perspectives

La SLDEI-résolution ainsi que la surréduction sont des méthodes générales. La généralité de ces calculs s'avère à la fois un avantage et un inconvénient. On peut en effet apprécier l'utilisation d'un même algorithme pour n'importe quel programme, ainsi que son efficacité sur des structures "pauvres" algébriquement, comme les séquences, les arbres, etc. ou sur des structures dont les ensembles supports sont finis comme les booléens. Par contre, on peut déplorer la lenteur et souvent la non terminaison de telles méthodes de calcul lorsqu'il s'agit de structures "riches" algébriquement, comme les entiers relatifs, pour lesquels des algorithmes spéciaux performants existent déjà dans la littérature. Cette constatation suggère de façon naturelle les deux directions suivantes :

• L'étude du contrôle (terminaison) pour les méthodes générales. Ce domaine classique en programmation logique, et qui est déjà difficile, voir par exemple [Dev87] [BAK90] ou [Bau88] [VP86], est compliqué par l'utilisation de la surréduction. L'investigation de ce problème est cruciale pour une utilisation effective de cette nouvelle génération de langages de programmation.

• L'autre direction de recherche est plutôt pragmatique et consiste en l'utilisation des *built-in*s dans les méthodes générales. En effet, les algorithmes spécialisés pour la résolution de buts dans des théories particulières sont certes plus efficaces que les algorithmes généraux. Aussi l'intégration de méthodes générales et d'autres particulières à certaines structures nous semble être une nécessité. Le cas de la programmation logique avec contraintes réalise cette combinaison en partie seulement puisque les buts à résoudre a l'aide d'algorithmes spécialisés sont clairement séparés des problèmes d'ordre général. Une telle étude de combinaison d'algorithmes de résolution a déjà été entamée dans le cadre de l'unification, voir par exemple [Kir85] ou [Yel85].

D'autre part, l'étude des fonctions partielles a toujours posé un problème dans le cadre des langages de spécifications algébriques. Plusieurs approches ont été proposés, voir [Wir89] §3.3 pour un résumé. De notre côté, nous n'avons pas considéré ce problème dans notre travail. Mais ce problème est important et mérite d'être étudié davantage. Moreno-Navarro et Rodriguez-Artalejo [MR89] ont proposé un cadre théorique satisfaisant lorsque les fonctions sont strictes. Par contre, quand les fonctions ne sont pas strictes, et en particulier lorsqu'on utilise des "récupérateurs", les sémantiques aussi bien déclarative qu'opérationnelle ne sont plus immédiates à obtenir.

Enfin, nous signalons un problème que nous n'avons pas abordé dans ce mémoire. Il s'agit de la validation des modèles effectifs lors des instances de modules génériques. C'est-à-dire la vérification de la validité de certaines formules dans des modèles initiaux. Ce problème, qui est bien compris d'un point de vue sémantique, se ramène en pratique à faire des preuves dans des théories *inductives* de Horn. Des techniques pour réaliser de telles preuves restent encore à améliorer.

Bibliographie

[AN89] H. Ait-Kaci, R. Nasr : *Integrating Logic and Functional Programming*. J. Lisp and Symbolic Computation, 2, 1989, 51-89.

[AL88] H. Ait-Kaci, P. Lincoln : *LIFE : A Natural Language for Natural Language*. MCC Technical report ACA-ST-074-88.

[And70] R.Anderson : *Completeness results for E-resolution*. Proc. Spring Joint Conference, 1970, 653-656.

[AE82] K.R.Apt, M.H.Van Emden : *Contributions to the Theory of Logic Programming*.JACM,Vol. 29, n° 3,July 1982, 841-862.

[AM75] M.A.Arbib, E.G.Manes : *Arrows, Structures and Functors*. Academic Press, 1975.

[Aub79a] R. Aubin : *Mechanizing Structural Induction* part I : Formal System. TCS 9, 1979, 329-345.

[Aub79b] R. Aubin : *Mechanizing Structural Induction* part II : Strategies. TCS 9, 1979, 346-362.

[BAK90] R.N. Bol, K.R. Apt, J.W. Klop : *On the Power of Subsumption and Context Checks*. DISCO 90, Capri, Italy, LNCS 429. 131-140.

[Bac77] J.W.Backus : Can Programming be Liberated From The Von Neumann Style? A Functional Style and its Algebra of Programs. CACM, 21, 1978, 613-641.

[BD87] L. Bachmair, N. Dershowitz : *Inference Rules for Rewrite-Based First-Order Theorem Proving*. Proc. of Symposium on Logic in Computer Science, Ithaca, New York, June 1987, IEE press, 331-337.

[BDH86] L. Bachmair, N. Dershowitz, J. Hsiang : *Orderings for Equational Proofs*. Proc. of Symposium on Logic in Computer Science, Cambridge, Massachusetts, June 1986. 346-357.

[BR90] J.W. de Bakker, J.J.M.M. Rutten : Integration ESPRIT Basic Research Action 3020. Bullettin of the EATCS. Number 40, February 1990, 79-99.

[BBLM86] R. Barbuti, M. Bellia, G. Levi, M.Martelli : LEAF : *A Language which Integrates Logic, Equations and Functions*.in Functional and Logic Programming, ed. DeGroot and Lindstrom, Prentice-Hall, 1986.

[Bau88] M. Baudinet : *Proving Termination Properties of PROLOG Programs: a Semantic Approach*. Proc of Logic in Computer Science,Edinburgh, Scotland, July 1988, 334-347.

[BL86] M.Bellia, G.Levi : *The Relation between Logic and Functional Languages: Asurvey*. J. Logic Programming, 217-236,1986.

[BK86] J.A. Bergstra, J.W. Klop : *Conditional Rewrite Rules: Confluence and Termination*. J. Computer and System Sciences 32, 1986, 323-362.

[BT87] J.A.Bergstra, J.V.Tucker : *Algebraic Specifications of Computable and Semicomputable Data Types*. TCS 50, 1987, 137-181.

[Ber89] G. Bernot : Correctness Proofs for Abstract Implementation. Information and Computation 80, (1989) 121-151.

[Ber83a] D.Bert : Manuel de Référence de LPG1.2. RR-IMAG 408. Grenoble, 1983.

[Ber83b] D.Bert : *Refinements of Generic Specifications with Algebraic Tools*. Information Processing 83, 1983, 815-820.

[BDE87a] D. Bert, P. Drabik, R. Echahed: Manuel de Référence de LPG version 1.8. Technical report RT17-IMAG, University of Grenoble, 1987.

[BDE87b] D. Bert, P. Drabik and R. Echahed, LPG: A Generic, Logic and Functional Programming Language, Proc. of the fourth Symposium on Theoretical Aspects in Computer Science (STACS'87), LNCS 247, Springer, 1987.

[Be et al90] D. Bert, P. Drabik, R. Echahed, J-M. Hufflen, O. Declerfayt, B. Demeuse, P-Y. Schobbens, F. Wautier: Reference Manual of the Specification Language LPG. Technical report RT59-IMAG, University of Grenoble, March 1990.

[Be et al 88] D. Bert, O. Declerfay, B. Demeuse, P. Drabik, R. Echahed, P-Y. Schobbens and F. Wautier, LPG: A Generic, Logic and Functional Programming Language, Proc. of the Second European Symposium on Programming (ESOP'88), LNCS 300, Springer, 1988.

[BDEH90] D. Bert, P. Drabik, R. Echahed, J-M. Hufflen: Examples of LPG Specifications. Technical report RT58-IMAG, University of Grenoble, May 1990.

[BE86] D.Bert, R.Echahed : *Design and Implementation of a Generic, Logic and Functional Programming Language*. Proc. of ESOP'86, LNCS n° 213,Saarebrücken, March 1986 119-132.

[BB88] P.Berlioux, Ph.Bizard : Algorithmique 2: Structures de Données et Algorithmes de Recherche. Dunod-Informatique, 1988.

[Bid89] M.Bidoit : PLUSS : un Langage Pour le Développement de Spécifications Algébriques Modulaires. Thèse d'Etat. Centre d'Orsay. Universite de Paris-Sud, 1989.

[Boc88] A. Bockmayr : Narrowing with Built-in Theories. Presented at the second International Workshop on Unification, Val d'Ajol, 1988.

[BGM87] P.G.Bosco, E.Giovannetti, C.Moiso : *Refined Strategies for semantic unification*. Proc. of TAPSOFT '87, LNCS n° 250, Pisa, March 1987, 276-290.

[BS86] R.V. Book, J.H. Siekmann : *On Unification : Equational Theories Are not Bounded*. J. Symbolic Computation 2, 1986, 317-324.

[BK81] K.A. Bowen, R.A. Kowalski : *Amalgamating Language and Metalanguage in Logic Programming*. Research Report DOC 81/30, University of London, June 1981.

[Bra75] D. Brand : Proving Theories with the Modification Method. SIAM J. Computt., vol4, n° 4, 1975, 412-430.

[Bro87] M. Broy : *Specification and Top-Down Design of Distributed Systems*. J. Computer and System Sciences 34, 1987, 236-265.

[BW82] M.Broy, M.Wirsing : Partial Abstract Types. Acta Informatica 18,1982, 47-64.

[BWP84] M.Broy, M.Wirsing, C.Pair : *A Systematic Study of Models of Abstract Data Types*. TCS, 33, 1984, 139-174.

[Buc85] B. Buchberger : *Basic Features and Development of the Critical-Pair/Completion Procedure*. Proc. of 1st RTA, LNCS n° 202, Dijon 1985,1-45.

[Bur69] R.M. Burstall : *Proving Properties of Programs by Structural Induction.* Computer Journal, vol12, 1, February 1969, 41-48.

[BG77] R.M. Burstall, J.A.Goguen : Putting Theories Together to make Specifications. in Proc. of 5th IJCAI, 1977, 1045-1058.

[BG80] R.M. Burstall, J.A.Goguen : The semantics of CLEAR, a Specification Language. LNCS 86, 1980, 292-332.

[BL69] R.M.Burstall, P.J.Landin : *Programs and their Proofs : an Algebraic Approch.* Machine Intelligence 4, 1969,17-43.

[BQS80] R.M. Burstall, D.B MacQueen, D.T. Sanella : HOPE : An Experimental Applicative Language. Internal Report CSR-62-80, University of Edinburg.

[Car87] L.Cardelli : *Basic Polymorphic Typechecking*. Science of computer Programming 8, 1987, 147-172.

[CW85] L.Cardelli, P.Wegner : *On Understanding Types, Data Abstraction, and Polymorphism*. ACM Computing Surveys, n°4, vol. 17,1985, 471-522.

[CD88] R. Cartwright, A. Demers : *The Topology of Program Termination.* Proc of Logic in Computer Science, Edinburgh, Scotland, July 1988, 296-308.

[CS79] C.L. Chang, J.R. Slagle : *Using Rewriting Rules for Connection Graphs to Prove Theorems*. Artificial Intelligence 12, 1979, 159-180.

[CL73] C.L. Chang, C.C.T. Lee : Symbolic Logic and Mechanical Theorem Proving. Academic Press, 1973.

[CS88] V. Chavatal, E. Szemerrdi : Many Hard Examples for Resolution. JACM, vol 35, n° 4, Octobre 1988, 759-768.

[Cla78] K.L. Clark : *Negation as Failure.* in H. Gallaire and J. Minker (eds), Logic and Data Bases, Plenum, New York, 1978, 293-322.

[Col88] A. Colmerauer : Les Avancées en Prolog III. in Nouveaux Enjeux pour la Recherche et ses Applications, Greco de Programmation du CNRS press, 1988, 66-71.

[Col82] A. Colmerauer : Prolog II Manuel de référence et Modèle Théorique. 1982, Rapport GIA Marseille.

[Col89] A. Colmerauer : Une Introduction à Prolog III. in Etat de l'art et Perspectives en Programmation Logique. Afcet, 1989.

[Col87] A. Colmerauer : Opening the Prolog III Universe. Byte 1987, 177-182

[Com86] H. Comon : *Sufficient Completeness, Term Rewriting Systems and "Anti_unification"*. Proc. of CADE-8, Oxford, 1986.

[Com88] H. Comon : Unification et Disunification. Théorie et Applications. Thèse de L'INPG. 1988.

[Dar**] J.L. Darlington : *Automatic Theorem Proving with Equality Substitutions and Mathematical Induction.* Machine Intelligence 3, 113-127.

[DG89a] J.L. Darlington, Y. Guo : Narrowing and Unification in Functional Programming -- An Evaluation Mechanism for Absolute Set Abstraction. Proc of RTA, Chapel Hill, YSA. April 1989, LNCS 355, 82-108.

[DG89b] J.L. Darlington, Y. Guo : The Unification of Functional and Logic Programming Languqges -- Towards Constraint Functional Programming. Proc of TENCON'89 IEEE Press, Bombay, India, 1989.

[Dau87] M. Dauchet : Termination of Rewriting is Undecidable in the One Rule System. Serie IT, Publication n°110, Laboratoire d'Informatique Fondamental de Lille, 1987.

[DDSW88] O. Declerfayt, B. Demeuse, P.Y. Schobbens, F. Wautier : Adequation des Specifications Formelles aux Problemes de Grande Envergure. International Workshop Software Engineering and its Applications. 5-9 Dec. 1988, EC2 Press. 483-505.

[DL86] D.DeGroot, G.Lindstrom : Logic Programming: Relations, Functions, and Equations. Prentice Hall, Englewood Cliffs, NJ, 1986.

[Der85] N.Dershowitz : *Computing with Rewrite Systems*. Information and Control 65,1985, 122-157.

[Der87] N. Dershowitz : *Termination of Rewriting* : J.of symbolic computation, 3, April 1987, 69-116.

[DJ89] N. Dershowitz, J.P Jouannaud : Rewrite Systems. Draft Of Chapter 15 of Volume B of "Handbook of Theoretical Computer Science", North-Holland, 1989.

[DP85] N. Dershowitz, D.A. Plaisted : Logic Programming cum Applicative Programming. Proc. SLP'85, Boston, July 1985, 54-66.

[DP**] N. Dershowitz, D.A. Plaisted : Equational Programming. in Machine Intelligence11, J.E. Hayes, D. Michie and J. Richards eds., 21-56.

[DO88] N.Dershowitz, M. Okada : *Proof-Theoretic Techniques for Term Rewriting Theory*. Proc of Logic in Computer Science,Edinburgh, Scotland, July 1988,104-111.

[DS88] N.Dershowitz, G. Sivakumar : *Goal-Directed Equation Solving*. Proc. of AAAI'88,vol. 1, St. Paul, Minnesota, USA, August, 1988, 166-170.

[Dev87] P. Devienne : *Les Graphes Orientés Pondérés : un Outil pour l'Etude de la Terminaison et de la Complexité dans les Systèmes de Réécriture et en Prodrammation Logique* : Thesis, November 1987, University of Lille, France.

[Dev88] P. Devienne : Weitghted Graphs, a Tool for Expressing the Behaviour of Recursive Rules in Logic Programming: Draft, June 1988.

[DSH87] M. Dinkbas, H. Simonis, P. van Hentenryck : Extending Equation Solving and Constraint Handling in Logic Programming. Proc. CREAS, Austin, Texas, May 1987.

[DH87] M. Dinkbas, P. van Hentenryck : Extending Unification Algorithms for the Integration of functional Programming int Logic Programming. J . Logic Programming 1987, 4, 199-227.

[DHSAG88]M. Dinkbas, P.van Hentenryck, H. Simonis, A. Aggoun, T. Graf : Application of CHIP to Industrial and Engineering Problems. in First International Conference on Industrial and Engineering Applications of Artifitial Intelligence and Expert Systems, Tullahoma, Tennessee USA, June 1988.

[Dra89] P. Drabik : *Validation Sémantique dans les Théories Structurées : Application à un langage de Programmation générique*. Thèse de L'INPG, 1989.

[Ech85] R. Echahed : Prédicat et sous-types en LPG -Réalisation de la E-unification-. RR-550 IMAG, University of Grenoble, 1985.

[Ech88] R. Echahed : On Completeness of Narrowing Strategies. Proc. CAAP'88. LNCS 299. Also appeared in TCS 72, 1990, 133-146.

[EB87] R. Echahed and D. Bert, An Overview of existing tools for manipulating algebraic components, ESPRIT-Project 283, Task: SEN.1, November 1987.

[EM85] H. Ehrig, B. Mahr : Fundamentals of Algebraic Specifications 1 : Equations and Initial Semantics. EATCS Monographs on Theoretical Computer science. W.Brauer, G.Rozenberg, A.Salomaa. Springer-Verlag. 1985.

[EM90] H. Ehrig, B. Mahr : Fundamentals of Algebraic Specifications 2 : Module Specifications and Constraints. EATCS Monographs on Theoretical Computer science. W.Brauer, G.Rozenberg, A.Salomaa. Springer-Verlag. 1990.

[EK76] M.H.van Emden, R.A.Kowalski : *The semantics of Predicate Logic as a Programming Language*. JACM, n°4, vol. 23, 1976,733-742.

[Fag87] F. Fages : Associative-Commutative Unification. J. of Symbolic Computation. (1987) 3, 257-275.

[FH86] F. Fages, G. Huet : Complete sets of Unifiers and Matchers in Equational Theories. TCS 43, 1986, 189-200.

[Fay79] M.J.Fay : *First Order Unification in an Equational Theory*. Proc. of the 4th workshop on automated Deduction, Austin, Texas, February 1979, 161-167.

[Fri84a] L.Fribourg : *Oriented Equational clauses as a Programming Language*. J.Logic Programming 2,1984, 165-177.

[Fri84b] L.Fribourg : A Narrowing Procedure for Theories with Constructors. 7th CADE, LNCS 170, 1984, 259-301.

[Fri85] L.Fribourg : *SLOG : A Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting*. Proc. of SLP '85, Boston, July 1985, 172-184.

[Fri86] L.Fribourg : Prolog with Simplification. France-Japan Artificial Intelligence and Computer Science Symposium 86. 239-265.

[FF90] D. de Frutos-Escrig, M.I. Fernandez-Camacho : On Narrowing strategies for partial non-strict functions. To appear in Proc. of TAPSOFT '91.

[FGJM84] K. Futatsugi, J.A. Goguen, J-P. Jouannand, J. Meseguer : Principles of OBJ2. Research report, Crin, 84-R-066, 1984

[FHS88] U. Furbach, S. Hölldobler, J. Schreiber : *Horn Equality Theories and Paramodulation*. Research report Bericht Nr.8801, Universität der Bundeswehr München, January 1988.

[GS89] H. Gaifman, E. Shapiro : Fully Abstract Compositional Semantics for Logic Programs. 6th POPL,1989.

[Gal86] J.H.Gallier : Logic for Computer Science: Foundations of Automatic Theorem Proving. Harper and Row, 1986.

[Gal87] J.H. Gallier : Fast Algorithms for Testing Unsatisfiability of Ground Horn Clauses with Equations. Journal of Symbolic Computation (1987) 4, 233-254.

[GR89] J.H. Gallier, S. Raatz : Extending SLD-Resolution to Equational Horn Clauses using E-unification. J.Logic Programming 1989, 3-43.

[GNRS88] J. Gallier, P. Narendran, S. Raatz, W. Snyder : RIGID E-Unification: NP-Completeness and Applications to Equational Matings. draft, August 1988.

[GS88] J.H. Gallier, W. Snyder : Complete Sets of Transformations for General E-Unification.draft, 1988.

[GS90] J.H. Gallier, W. Snyder : Designing Unification Procedures Using Transformations : a Survey. Bulletin of the EATCS. Number 40, february 1990, 273-325.

[GG88] S.J. Garland, J.V. Guttag: Inductive Methods for Reasoning about Abstract Data Types. Proc. 15th POPL, ACM press, San Diego, California, 1988, 219-228.

[Gau89] M.C. Gaudel : Le Génie Logiciel. LRI Technical report n° 469, 1989.

[GM88] M.C. Gaudel, T. Moineau : A Theory of Software Reusability. in Proc. of ESOP'88, LNCS 300, 1988, 115-130.

[GH86] A.Geser, H.Hussmann : *Experiences with the RAP system -a Specification interpreter combining term rewriting and resolution.* Proc. of ESOP'86, LNCS n° 213, Saarebrücken, March 1986, 339-350.

[GM86] E. Giovannetti, C. Moiso : A Completeness Result for E-unification Algorithms based on Conditional Narrowing. Proc Of Foundations of Logic and Functional Programming. Trento, Italy 15-19 Dec.1986. LNCS 306, 157-167.

[Göb87] R. Göbel : Ground Confluence. Proc. RTA'87, LNCS, n° 256, Bordeaux, May 1987, 154-167.

[GDLE84] M. Gogolla, K. Drosten, U. Lipeck, H.-D. Ehrich : Algebraic and Operational Semantics of Specifications Allowing Exceptions and Errors. TCS 34, 1984, 289-313.

[Gog80] J.A.Goguen : How to Prove Algebraic Inductive Hypothesis Without Induction with Applications to the Correctness of Data Type Implementaion. 5th CADE 1980, LNCS 87, 356-373.

[Gog88] J.A.Goguen : Higher Order Functions Considered Unnecessary for Higher Order Programming. SRI-CSL-88-1R, SRI International.

[GB84a] J.A.Goguen, R. M.Burstall : INSTITUTIONS: Absract Model Theory for Computer Science. Draft of February 1985. A previous version appered in Proc. of Logics and Programming Workshop, E. Clarke and D. Kozen Ed., Springer-Verlag, 1984, 221-256.

[GB84b] J.A Goguen, R. M.Burstall : Some Fundamental Algebraic Tools for the Semantics of Computation, Part1 : Comma Categories, Colimits, Signatures and Theories. TCS 31 1984, 175-209.

[GB84c] J.A Goguen, R. M.Burstall : Some Fundamental Algebraic Tools for the Semantics of Computation, Part2 : Signed and Abstract Theories. TCS 31 1984, 263-295.

[GM84] J.A.Goguen, J.Meseguer : *Equality, Types, Modules and (why not ?) generics for logic programming*.J. Logic Programming,vol.1, 1984, 179-210.

[GM86] J.A.Goguen, J.Meseguer : EQLOG: Equality, Types and Generic Modules for Logic Programming. in Functional and Logic Programming, ed. DeGroot and Lindstrom, Prentice-Hall, 1986.

[GM87] J.A.Goguen, J.Meseguer : Remarks on Remarks on Many-sorted Equational Logic. in SIGPLAN notices, 1987, 41-48.

[GM87] J.A.Goguen, J.Meseguer : *Models and Equality for Logical Programming*. Proc. of TAPSOFT'87, LNCS n° ??, Pisa, March 1987, 1-22.

[GTW78] J.A.Goguen, J.W. Thatcher, E.G. Wagner : An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types.in R.T. Yeh ed., current Trends in Programming Methodology, 4, Prentice Hall, 1978, 80-149.

[GMW79] M.J. Gordon, R. Milner, C.P. Wadsworth : Edinburgh LCF. LNCS 78, Springer-Verlag, 1979.

[Gut75] J.Guttag : The Specification and Application to programming of Abstract data Types. PhD Thesis, University of Toronto, 1975.

[Gut77] J. Guttag : Abstract Data Types and the Development of Data Strucrures. Com. of the ACM, 6, vol 20, Juin 1977. 396-404.

[GHM78] J. Guttag, E. Horowitz, D.R. Musser : Abstract Data Types and Software Validation. Com. of the ACM, vol 21, n° 12,Dec. 1978. 1048-1064.

[GH78] J. Guttag, J.J Horning : The Algebraic Specification of Abstract Data Types. Acta Informatica, 10, (1978) 27-52.

[HR78] M.C. Harrison, N. Rubin : Another Generalisation of Resolution. JACM, vol 25, n° 3, July 1978,341-351.

[HH87] S. Heilbrunner, S. Hölldobler : The Undecidability of the Unification and Matching Problem for Canonical Theories. in Acta Informatica 24, 1987, 157-171.

[HD87] P. van Hentenryck, M. Dincbas : Forward Checking in Logic Programming. Proc. 4th International Conference on Logic Programming, Melbourne, May, 1987,229-256.

[Her86] A.Herold. Narrowing Techniques Applied to Idempotent Unification. SEKI Technical report SR-86-16, 1986.

[Höl88] S. Hölldobler : *SLDE-Resolution.* draft, March 1988.

[Höl89] S. Hölldobler : *Foundations of Equational Logic Programming*. LNAI n° 353, Edt. J. Siekmann, Springer Verlag.1989.

[HR87] J. Hsiang, M. Rusinowitch : On Word Problems in Equational Theories. Proc. ICALP'87, LNCS 267, 1987, 54-71.

[Hu89] P. Hudak : Conception, Evolution, and Application of Functional Programming Languages. in ACM Computing Surveys, vol 21, n°3, september 1989,359-411.

[Hue76] G.Huet : Résolution d'équations dans les langages d'ordre 1, 2,...,ω . Thèse d'état. Univ. Paris VII, 1976.

[Hue80] G.Huet : Confluent Reductions : Abstract Properties and Applications to Term Rewriting Systems. J of the A.C.M. n°4, vol 27, Oct. 1980, 797-821.

[Hue80] G.Huet : A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm. Rapport de recherche INRIA n° 25, 1980.

[HH80] G.Huet, J-M.Hullot : *Proofs by Induction in Equational Theories with constructors*. Proc. of 21st Annual Symposium on Foundationn of Computer Science,1980, 96-107.

[HL78] G. Huet, D. Lankford : On the Uniform Halting Problem for Term Rewriting Systems. Rapport de recherche INRIA n° 283. Mars 1978.

[HO80] G.Huet,D.C.Oppen : *Equations and Rewrite rules : a Survey*. In "Formal Languages : Perspectives and Open problems". Ed. R.Book, Academic press, 1980.

[Huf89] J-M. Hufflen : Fonctions et Généricité dans un langage de Programmation parallèle. Thèse de l'INPG. 1989.

[Hul80a] J-M. Hullot : *Canonical Forms and Unification*. Proc.of 5th CADE, LNCS n° 87, 1980, 318-334.

[Hul80b] J-M.Hullot : Compilation de Formes Canoniques Dans les Théories Equationnelles. Thèse de 3éme cycle. Université Paris-Sud Centre Orsay. 1980.

[Hau85a] H.Hussmann : *Unification in Conditional Equational Theories*. Proc. of EUROCAL'85, LNCS, n°204, 1985, 543-553.

[Hus85b] H. Hussmann : Rapid Prototyping for Algebraic Specifications - RAP - System User's Manual. Report MIP - 8540, Mars 1985.

[Hus88] H. Hussmann : Nondeterministic Algebraic Specifications. PhD Thesis, Universität Passau, 1988 (english translation, 1990).

[Ive62] K. Iverson : A Programming Language. Wiley, 1962.

[JL87] J. Jaffar, J-L. Lassez : Constraint Logic Programming. POPL'87, 1987,111-119.

[JLM84] J. Jaffar, J-L. Lassez, M.J Maher : A Theory of Complete Logic Programs with Equality. J. Logic Programming 1984:3:211-233.

[JLM86] J. Jaffar, J-L. Lassez, M.J Maher : Some Issues and Trends in the Semantics of Logic Programming. Third International Conference on Logic Programming, London. LNCS 225, 223-241

[Jon84] T.Capers Jones : Reusability in Programming : A Survey of the state of the Art. IEEE Transaction on Soft. Eng. vol SE-10, n° 5, Sep 1984, 488-493.

[JD86] A.Josephson, N.Dershowitz : *An Implementation of Narrowing: The RITE Way*. Proc. of SLP'86,IEEE press, Salt lake city, Utah, 1986,187-197.

[JKK82] J-P.Jouannaud, C. Kirchner, H. Kirchner : Incremental Unification in Equational Theories. Greco Report 20-82, 1982.

[JK90] J-P.Jouannaud, C. Kirchner : Solving Equations in Abstract Algebras : A Rule-Based Survey of Unification. LRI, Unite AL Khowarizmi, rapport de recherche 561.

[JK83] J-P.Jouannaud, H. Kirchner : Completion of a set of Rules Modulo a Set of Equations. Rapport GRECO 10.83, Université de Bordeaux.

[JK86] J-P.Jouannaud, E.Kounalis : *Automatic Proofs by Induction in Equational Theories without Constructors*. Proc. of 1st IEEE Symposium on Logic in Computer Science, 1986.

[JL86] J-P.Jouannaud, P.Lescanne : La Réécriture. TSI 5,6,1986,43-452.

[JW86] J-P.Jouannaud, B. Waldmann : Reductive Conditional Term Rewriting Systems. Rapport de Recherche LRI 86.R.026.

[Kap84] S.Kaplan : *Fair Conditional Term Rewriting Systems : Unification, Termination and Confluence*. Rapport LRIn° 194.

[Kam] S. Kamin : Final Data Types and their Specifications.

[Kap87] S.Kaplan : *Simplifying Conditional Term Rewriting Systems : Unification, Termination* . J. Symbolic Computation (1987) 4, 295-334.

[Kap84] S.Kaplan : *Conditional Rewrite Rules*. TCS, 33, 1984, 175-193.

[Kap88] S.Kaplan : Positive/Negative Conditional Rewriting. Proc. MFCS'88, LNCS*, 381-395.

[KR88] S.Kaplan, J-L. Rémy : Completion Algorithms for Conditional Rewriting Systems. Draft, July 1988.

[KNZ87] D.Kapur,P.Narendran, H.Zhang : On Sufficient Completeness and Related Properties of Term Rewriting Systems. Acta Informatica 24, 395-415.1987.

[Kir85] C. Kirchner : Mhéthodes et Outils de Conception Systématique d'Algorithmes d'Unification dans les Théories Equationnelles. Thèse d'Etat, Université de Nancy I, 1985.

[Kir*] C. Kirchner : Order-Sorted Equational Unification.

[KK82] C. Kirchner, H. Kirchner : Unification Dans les Theories Equationnelles. Rapport GRECO n° 3.82.

[Kla84] H.A Klaeren : A Constructive Method for Abstract Algebraic Software Specifications. TCS 30 (1984), 139-204.

[Kni89] K. Knight : Unification : A Multidisciplinary Survey. ACM Computing Surveys, vol 21, n° 1, March 1989.

[Knu69] D.E. Knuth : The Art of Computer Programming. Vol I, Fundamental Algorithms. Addison-Wesley, 1969.

[KB70] D.E. Knuth, P.B. Bendix : Simple Word Problem in Universal Algebras. Computational Problems in Abstract Algebra, Leech ed., Pergamon Press, 1970, 263-297.

[Kom82] H. Jan Komorwski : QLOG - The Programming Environment for Prolog in Lisp. in K.L Clark, S.A. Tonlund (eds), Logic Programming, Academic press, 1982, 315-322.

[KM87] H. Jan Komorwski, J. Maluszynski : Logic Programming and rapid Prototyping. Science of computer programming, 9 (1987) 179-205.

[Kou90] E. Kounalis : Testing for Inductive (Co)-Reducibility. CAAP 90, LNCS ? , 221-238.

[Kow74] R.Kowalski : *Predicate Logic as Programming Language*. Information Processing, 74, 1974, 569-574.

[Kow79] R.Kowalski : *Algorithm = Logic + Control.* CACM, n° 7, vol. 22, 1979, 424-436.

[KK71] R.Kowalski, D.Kuehner : Linear Resolution with Selection Function. Artificial Intelligence, 1971, 227-260.

[Lan66] P.J. Landin : The Next 700 Programming Languages. CACM 9, 3, 1966, 157-166.

[JMM86] J-L.Lassez, M.J.Maher, K.Marriott : Unification Revisited. in Proc. of Foundations of Logic and Functional Programming, Trento, Italy, December 1986,67-113.

[LNS82] J-L.Lassez, V.L. Nguyen, E.A. Sonenberg : Fixed Point Theorems and Semantics : A Folk Talk. Information Processing Letters,n° 3, vol 14,1982, 112-116.

[LLT90] A.Lazrek, P.Lescanne, J-J. Thiel : Tools for Proving In ductive Equalities, Relative Completeness, and ω-Completeness. Information and Computation 84, 1990,47-70.

[LW82] J. Leszczylowski, M. Wirsing : A System for Reasoning within and about Algebraic specifications. LNCS 137, Torino 1982, 257-282.

[Les83] P.Lescanne : Computer Experiments with the REVE Term Rewriting System Generator. in Proc. of 10th POPL, 1983, 99-108.

[LB87] G. Levi, P.G. Bosco : A Complete Semantics Characterization of K-LEAF, A Logic Language with Partial Functions. in Proc. of SLP'87, 1987, 318-327.

[LZ74] B. Liskov, S. Zilles : Programming with Abstract Data Types. Sigplan Notices. 4, vol 9 , 74

[Llo87] J.W Lloyd : Foundations of Logic Programming. Springer-Verlag (Symbolic Computation Artificial Intelligence series) 1987.

[Lov78] D.W. Loveland : Automated Theorem Proving : A Logical Basis. North-Holland Publishing Company. 1978.

[McC60] J.McCarthy : Recursive Functions of Symbolic Expressions and their Computation by machine. Part I, CACM 3, 4, 1960, 184-195.

[MM84] B. Mahr, J.A. Makowsky : Characterizing Specification Languages which Admit Initial Semantics. TCS 31, 1984, 49-59.

[Mak87] J.A. Makowsky : Why Horn Formulas Matter in Computer Science: Initial Structures and Generic Examples.in JCSS 34, 1987, 266-292.

[MR89] J.J. Moreno-Navarro, M. Rodriguez-Artalejo : Logic Programming with Functions and Predicats : The Language BABEL. Rapport DIA/89/3, Universidad Complutense Madrid.

[MW80] Z. Manna, R. Waldinger : A Deductive Approach to Program Synthesis. TOPLAS 2, 1, 1980, 92-121.

[MW86] Z. Manna, R. Waldinger : Special Ralations in Automated Deduction. JACM, vol 33,n° 1, January 1986, 1-59.

[MMW86] Y. Malachi, Z. Manna, R. Waldinger : A New Approach to Logic Programming. in K.L Clark, S.A. Tonlund (eds), Logic Programming, Academic press, 1982, 365-394.

[MP88] P. Mancarella, D. Pedreschi : An Algebra of Logic Programs. Proc. Joint Int. Symposium and Conference on Logic Programming, Seattle, 1988.

[MMR86] A. Martelli, C. Moiso, G.F. Rossi : An Algorithm for Unification in Equational Theories. in Proc SLP 86 Saltlakecity.

[MM82] A. Martelli, U.Montanari : An Efficient Unification Algorithm. ACM TOPLAS, vol 4, n° 2, april 1982, 258-282.

[MG85] J.Meseguer, J.A.Goguen : *Initiality,Induction and Computability*. In Algebraic Methods in Semantics, M.Nivat and J.C.Reynolds, Eds, Cambridge University Press, 1985, 459-541.

[Mil86] R. Milner : Is Computing an Experimental Science?. LFCS-report, 1986

[MH88] J.C.Mitchell, R.Harper : The essence of ML. Proc. of POPL, 1988, 28-46.

[MS86] C.K.Mohan, M.K.Srivas : Function Definitions in Term Rewriting and Applicative Programming. Information and Control 71, 1986, 186-217.

[Mey86] B. Meyer : Genericity versus Inheritance. OOPSALA'86 Proceedings, 391-405.

[Mis84] P. Mishra : Towards a Theory of Types in Prolog. Proc SLP'84, Atlantic city, 1984, 289-298.

[Mor69] J.B. Morris : E-Resolution : Extension of Resolution to Include The Equality Relation. IJCAI69, 1969, 287-294.

[Muss80] D. Musser : Abstract Data Type Specification in the AFFIRM System. IEEE Transaction onSofware Engineering, vol 6, n° 1, January 1980, 24-32.

[Mus80] D.Musser : On Proving Inductive Properties Of Abstract Data Types. Proc. of 7th POPL, Las Vegas, 1980, 154-162.

[NNGG89] E.Nagel, J.R.Newman, K.Gödel, J-Y.Girard : "Le Théorème de Gödel. Seuil, 1989.

[NO87] M. Navarro, F. Orejas : Parametreized Horn Clause Specifications : Proof Theory and Correctness. TAPSOFT'87, LNCS 249, 202-216.

[Nar88] S.Narain : LOG(F) : An Optimal Combination of Logic Programming, Rewriting, and Lazy Evaluation. RAND/P-7437, April 1988.

[Nil80] N.J. Nilsson : *Principles of Artificial Intelligence*.Springer-Verlag,1980.

[Nil83] J.F. Nilsson : On the Compilation of a Domain-Based PROLOG. Information Processing 83, IFIP, 1983, 293-298.

[NW83] T.Nipkow, G.Weikum : A Decidability Result about Sufficient-Completeness of Axiomatically Specified Abstract Data Types. LNCS 145, 1983, 257-268.

[NRS87] W. Nutt, P. Réty, G. Smolka : Basic Narrowing Revisited. Seki-Report SR-87-07, University of Kaiserslautern, 1987.

[O'Do85] M.J.O'Donnell : Equational Logic as a Programming Language. MIT Press, 1985.

[O'Do87] M.J. O'Donnell : Survey of the Equational Logic Programming Project. Proc of C.R.E.A.S. 1987.

[OS88] A. Ohsuga, K. Sakai : An Efficient Implementation Method of Reduction and Narrowing in Metis. presented at the second International Workshop on Unification, Val d'Ajol, June 1988.

[Pad87] P. Padawitz : *Strategy-Controlled Reduction and Narrowing*. Proc. RTA'87, LNCS n° 256, Bordeaux, May 1987, 242-255.

[Pad88a] P. Padawitz : Computing in Horn Clause Theories, Springer, 1988.

[Pad88b] P.Padawitz : Inductive Proofs of Constructor-based Horn Clauses. Resarch report MIP-8810, University of Passau,1988.

[Par72] D.C.Parnas : On the Criteria to be Used in Decomposing Systems into Modules. CACM 15, 12, 1972, 1053-1058.

[Pat78] M.S.Paterson, M.N.Wegman : Linear Unification. JCSS 16, 1978, 158-167.

[Pau**] E.Paul : Proof by Induction in Equational Theories with relations between Constructors.

[Pau85] E.Paul : Equational Methods in First Order Predicate Calculus. Journal of Symbolic Computation (1985) 1, 7-29.

[Pau86] E.Paul : On solving the Equality Problem in Theories Defined by Horn Clauses. T.C.S 44 (1986) 127-153.

[Pet83] G.E.Peterson : A Technique for Establishing Completeness Results in Theorem Proving with Equality.SIAM J. Comput., vol 12, n° 1, Ferbruary 1983,82-100

[Pey87] S.L. Peyton Jones : The Implementation of Functional Programming Languages. Prentice Hall, 1987.

[Pla85] D.A.Plaisted : Semantic Confluence Tests and Completion Methods. J. Information and Control 65, 1985, 182-215.

[Pla88] D.A.Plaisted : Non-Horn Clause Logic Programming without Contrapositives. J. Automated Reasoning 4, 1988, 287-325.

[Pla90] D.A.Plaisted : Mechanical Theorem Proving. in Formal Techniques in Artificial Intelligence. A Sourcebook. R.B. Banerji (ed). Elseiver Science Publishers (North-Holland). 1990.

[Plo72] G.D.Plotkin : Building-in Equational Theories. Machine Intelligence 7, 1972, 73-90.

[Red86] U.S.Reddy : Narrowing as the Operational Semantics of Functional Languages. In: Logic Programming: Relations, Functions, and Equations, D.DeGroot and G.Lindstrom, eds. Prentice Hall, Englewood Cliffs, NJ, 1986.

[Rei80] H. Reichel : Initially Restricting Algebraic Theories. Proc. MFCS'80, LNCS 88, 1980, 504-514.

[Rét88] P.Réty : Methodes d'Unification par Surreduction. Thèse de Doctorat d'université de Nancy 1. 1988.

[Rét87] P.Réty : *Improving Basic Narrowing Techniques*. Proc. RTA'87, LNCS, n° 256, Bordeaux, May 1987, 228-241.

[RKKL85] P.Réty, C.Kirshner, H.Kirshner, P.Lescanne : Narrower: a new Algorithm for Unification and its application to Logic Programming. Proc. RTA'85, 1985.

[Rey87] J-C. Reynaud : Sémantique de LPG.RR-IMAG 651, Grenoble, 1987.

[Rey87] J-C. Reynaud : Putting Algebraic Component Together. Proc. DISCO'90, LNCS 429, 1990.

[Rob65] J.A.Robinson : *A Machine-Oriented Logic Based on the Resolution Principle*. JACM, 12, 1965,23-41.

[Rob**] J.A.Robinson : Beyond LOGLISP : combining functional and relational programming in a reduction setting. in Machine Intelligence 11, J.E. Hayes, D. Michie and J. Richards eds.57-68.

[RS81] J.A. Robinson, E.E. Sibert : The LOGLISP User's Manual. Report 12/81. Syracuse University. New York.

[RS82] J.A. Robinson, E.E. Sibert : LOGLISP : Motivation, Design and Implementation. in K.L Clark, S.A. Tonlund (eds), Logic Programming, Academic press, 1982, 299-314.

[RS82] J.A. Robinson, E.E. Sibert : LOGLISP : an Alternative to Polog. in Machine Intelligence 10, E. Hirwood (ed), 1982, 399-419.

[RW69] J.A.Robinson, L.Wos : Paramodulation and Theorem-Proving in First-Order Theories with Equality. Machine Intelligence 4, 1969, 135-150.

[Rus89] M. Rusinowitch : Démonstration Automatique, Technique de réécriture. InterEditions, Paris, 1989.

[ST86] D.Sannella, A.Tarlecki : Extended ML : an Institution-independent Framework for Formal Program Development. LFCS report-86-16, December 1986.

[ST86] D.Sannella, A.Tarlecki : Toward Formal Development of Programs from Algebraic Specifications : Implementation Revisited. LFCS report-86-17, 1986.

[ST90] D.Sannella, A.Tarlecki : Program Development : What Are the Real Problems?. Bulletin of The EATCS. Number 41, june 1990, 135-137.

[SS88] M.Schmidt-Schauβ, J.H.Siekmann : Unification Algebras : An Axiomatic Approach to Unification, Equation Solving and Constraint Solving. SR-88-23.

[Sch88] M.Schmidt-Schauβ : Implication of Clauses is Undecidable. TCS 59 (1988) 287-296.

[Sla74] J.R.Slagle : *Automated Theorem Proving for Theories with Simplifiers, Commutativity, and Associativity*. JACM, Vol. 21, n° 4, October 1974, 622-642.

[Smo88] G. Smolka : TEL (Version 0.9) Report and User Manual. SEKI Report SR-87-11, Universität Kaiserslautern, West Germany, 1988.

[Sti81] M.E.Stickel : A Unification Algorithm for Associative-Commutative Functions. JACM for Computing Machinery, vol 28, n° 3, July1981, 423-434.

[Sti85] M.E.Stickel : Automated Deduction by Theory Resolution. J. Automated Reasoning 1, 1985, 333-355.

[Sie89] J.H.Siekmann : Universal Unification. J. Symbolic Computation, vol 7, n° 3 & 4. 1989, 207-274.

[SS82] J. Siekmann, P. Szabo : Universal Unification and a Classification of Equational Theories. Universität Karlsrue. Report 7/82.

[Sto77] J.E. Stoy : Denotational Semantics : The Scott-Strachey Approach to Programming Language Theory. MIT Press, 1977.

[TW**] A.Tarlecki, M.Wirsing : Continuous Abstract Data Types : Basic Machinery and Results. Proc. of Fundamental of Computation Theory, LNCS 199, 431-441.

[TWW82] J.W. Thatcher, E.G. Wagner, and J.B. Wright : Data Type Specification : parameterization and the Power of Specification Techniques. TOPLAS 4,1982,711-732.

[Tha87] S.Thatte : A Refinement of Strong Sequentiality for Term Rewriting with Constructors. J. Information and Computation 72, 1987, 46-65.

[Tha88] S.Thatte : Implementing First-Order Rewriting with Constructor Systems. J. TCS 61, 1988, 83-92.

[Thi84] J.J.Thiel : Stop Losing Sleep over Incomplete Data Type Specifications. Proc. 11th POPL, Salt Lake City, 1984, 76-82.

[Tim88] M. Tiomkin : *Proving Unprovability*. Proc. of Logic in Computer Science,Edinburgh, Scotland, July 1988, 22-26.

[Tur81] D.A. Turner : The Semantic Elegance of Applicative Languages. Proc. of Conf. on Functional Prog. Languages and Computer Architecture. ACM, 1981, 85-92.

[Tur85] D.A. Turner : Miranda : A non-strict Functional Language with Polymorphic types. Proc. of Functional Programming Languages and Computer Architecture, LNCS 201,1985, 1-16.

[VP86] T.Vasak, J. Potter : Characterisation of Terminating Logic Programs. Proc. of SLP'86, Salt lake city, Utah, 1986, 140-147.

[War77] D.H.D. Warren : Implementing Prolog - Compiling Logic Programs, 1 and 2. D.A.I. Research report, Univ. of Edinburgh, 1977.

[Wir89] M. Wirsing : Algebraic Specifications. Report MIP-8914, Universtät Passau, Juin 1989.

[WPPDB83] M. Wirsing, P. Pepper, H. Partsch, W. Dosch, M. Broy : On Hierarchies of Abstract Data Types. Acta Informatica 20, 1983, 1-33.

[Yel85] K.Yelick : Combining Unification Algorithms for Confined Regular Equational Theories. Proc. of RTA'85, LNCS n°202, 1985, 365-380.

[You88] J-H. You : Outer Narrowing for Equational Theories Based on Constructors. Proc. of ICALP'88.

Grenoble, le 19 Novembre 1990

DÉPARTEMENT DES ÉTUDES DOCTORALES

Affaire suivie par
Tél : 76.57.

N/Réf. :

Objet :

AUTORISATION de SOUTENANCE

Vu les dispositions de l'arrêté du 23 Novembre 1988 relatif aux Etudes Doctorales
Vu les rapports de présentation de :

- Monsieur DAUCHET
- Monsieur KIRCHNER

Monsieur ECHAHED Rachid

est autorisé(e) à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR de l'INSTITUT NATIONAL POLYTECHNIQUE de GRENOBLE, spécialité :

"Informatique

Pour le Président de l'I.N.P.-G.
et par délégation,
le Vice-Président
M. GARNIER

Abstract :

This thesis presents an approach, based on Horn clause logic with equality, to the integration of logic and functional programming languages. We start our investigation by defining the programs we consider. They are syntactically defined as theory presentations in the framework of Horn clause logic with equality. The semantic of a program is its least Herbrand E-model.

We then focus on the computational aspects of this language, and propose a deductive system consisting of a unique rule we call SLDEI-resolution. We prove the soundness, the completeness and the strong completeness of this rule.

Implementations of SLDEI-resolution lean on algorithms of resolution of equations. We consider such algorithms based on narrowing relationship and improve them by using narrowing strategies. These strategies are not complete in general, so we propose sufficient conditions on these strategies that ensure the completeness of the algorithms we consider.

We characterize afterwards a class of programs, called uniform programs, so that any narrowing strategy leads to a complete algorithm of resolution of equations. We propose then a procedure for deciding whether or not a program is uniform. Furthermore, we give syntactical criteria that warrant the uniformity of programs.

Finally, we give an overview of main features of a programming language based on the approach presented in this thesis, and sketch our implementation.

Keywords : Logic Programming, Algebraic Specification, Horn Clause Logic with Equality, SLDEI-resolution, EI-unification, Narrowing.

Résumé :

Ce mémoire présente l'étude d'une approche particulière des langages de programmation logico-fonctionnels, fondée sur la logique des clauses de Horn avec égalité. Nous définissons d'abord la syntaxe et la sémantique des programmes que nous considérons. La syntaxe est celle de la logique des clauses de Horn avec égalité. La sémantique est donnée par le plus petit E-modèle de Herbrand associé à un programme.

Nous nous intéressons ensuite au calcul dans ce langage. Nous proposons pour cela une nouvelle règle appelée SLDEI-résolution comme seule règle de calcul. Nous montrons sa cohérence, sa complétude ainsi que sa complétude forte.

La mise en œuvre de la règle SLDEI-résolution nécessite un algorithme de résolution d'équations. Nous étudions de tels algorithmes fondés sur la relation de surréduction, et améliorons ces algorithmes par l'utilisation de stratégies de surréduction. Cependant, ces stratégies ne sont pas complètes dans le cas général. Nous proposons alors des conditions suffisantes sur ces stratégies afin de préserver la complétude des algorithmes considérés.

Nous caractérisons ensuite une classe de programmes, dits uniformes, pour lesquels l'utilisation de n'importe quelle stratégie de surréduction donne un algorithme complet de résolution d'équations. Nous donnons de plus une méthode de vérification de l'uniformité d'un programme. Par ailleurs, nous proposons des conditions syntaxiques pour qu'un programme soit uniforme.

Enfin, nous décrivons les principaux traits d'un langage de programmation fondé sur l'approche présentée dans ce mémoire, et l'implantation que nous avons réalisée.

Mots-clés : Programmation Logique, Spécification Algébrique, Logique des Clauses de Horn avec Egalité, SLDEI-résolution, EI-unification, Surréduction.