Utilisation des schématisations de termes en déduction automatique

Hicham Bensaid

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Hicham BENSAID

Thèse dirigée par **Ricardo CAFERRA** et **Nicolas PELTIER**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

UTILISATION DES SCHÉMATISATIONS DE TERMES EN DÉDUCTION AUTOMATIQUE

Thèse soutenue publiquement le **17 juin 2011**,
devant le jury composé de :

Dominique DUVAL
Professeure à l'Université Joseph Fourier, Présidente
Denis LUGIEZ
Professeur à l'Université de Provence, Rapporteur
Christopher LYNCH
Professeur à la Clarkson University, Rapporteur
Bernhard GRAMLICH
Associate Professor à la Technische Universität Wien, Examinateur
Ricardo CAFERRA
Maître de conférences (HDR) à l'ENSIMAG, Directeur de thèse
Nicolas PELTIER
Chargé de recherche (HDR) au CNRS, Directeur de thèse

J'en viens à ceci, que les travaux d'écolier sont des épreuves pour le caractère, et non point pour l'intelligence. Que ce soit orthographe, version ou calcul, il s'agit de surmonter l'humeur, il s'agit d'apprendre à vouloir.

Émile-Auguste Chartier, dit Alain
Propos sur l'éducation

Résumé

Les schématisations de termes permettent de représenter des ensembles infinis de termes ayant une structure similaire de manière finie et compacte. Dans ce travail, nous étudions certains aspects liés à l'utilisation des schématisations de termes en déduction automatique, plus particulièrement dans les méthodes de démonstration de théorèmes du premier ordre par saturation.

Après une brève étude comparée des formalismes de schématisation existants, nous nous concentrons plus particulièrement sur les termes avec exposants entiers (ou I-termes). Dans un premier temps, nous proposons une nouvelle approche permettant de détecter automatiquement des régularités dans les espaces de recherche. Cette détection des régularités peut avoir plusieurs applications, notamment la découverte de lemmes nécessaires à la terminaison dans certaines preuves inductives. Nous présentons DS3, un outil qui implémente ces idées. Nous comparons notre approche avec d'autres techniques de généralisation de termes. Notre approche diffère complètement des techniques existantes car d'une part, elle est complètement indépendante de la procédure de preuve utilisée et d'autre part, elle utilise des techniques de généralisation inductive et non déductives. Nous discutons également les avantages et les inconvénients liés à l'utilisation de notre méthode et donnons des élements informels de comparaison avec les approches existantes.

Nous nous intéressons ensuite aux aspects théoriques de l'utilisation des I-termes en démonstration automatique. Nous démontrons que l'extension aux I-termes du calcul de résolution ordonnée est réfutationnellement complète, que l'extension du calcul de superposition n'est pas réfutationnellement complète et nous proposons une nouvelle règle d'inférence pour restaurer la complétude réfutationnelle.

Nous proposons ensuite un algorithme d'indexation (pour une sous-classe) des I-termes, utile pour le traitement des règles de simplification et d'élimination de la redondance.

Finalement nous présentons DEI, un démonstrateur automatique de théorèmes capable de gérer directement des formules contenant des I-termes. Nous évaluons les performances de ce logiciel sur un ensemble de benchmarks.

Mots clés : démonstration automatique de théorèmes, schématisation de termes, I-termes, analyse de l'espace de recherche, complétude réfutationnelle, indexation de termes.

Abstract

Term schematisations allow one to represent infinite sets of terms having a similar structure by a finite and compact form. In this work we study some issues related to the use of term schematisation in automated deduction, in particular in saturation-based first-order theorem proving.

After a brief comparative study of existing schematisation formalisms, we focus on terms with integer exponents (or I-terms). We first propose a new approach allowing to automatically detect regularities (obviously not always) on search spaces. This is motivated by our aim at extending current theorem provers with qualitative improvements. For instance, detecting regularities permits to discover lemmata which is mandatory for terminating in some kinds of inductive proofs. We present DS3, a tool which implements these ideas. Our approach departs from existing techniques since on one hand it is completely independent of the proof procedure used and on the other hand it uses inductive generalization techniques instead of deductive ones. We discuss advantages and disadvantages of our method and we give some informal elements of comparison with similar approaches.

Next we tackle some theoretical aspects of the use of I-terms in automated deduction. We prove that the direct extension of the ordered resolution calculus is refutationally complete. We provide an example showing that a direct extension of the superposition calculus is not refutationally complete and we propose a new inference rule to restore refutational completeness.

We then propose an indexing algorithm for (a subclass of) I-terms. This algorithm is an extension of the perfect discrimination trees that are are employed by many efficient theorem provers to implement redundancy elimination rules.

Finally we present DEI, a theorem prover with built-in capabilities to handle formulae containing I-terms. This theorem-prover is an extension of the E-prover developed by S. Schulz. We evaluate the performances of this software on a set of benchmarks.

Keywords: automated theorem proving, term schematisation, I-termes, search space analysis, refutation completeness, term indexing.

Remerciements

Je tiens en premier à remercier vivement tous les membres du jury qui m'ont fait l'honneur d'évaluer ce travail et d'y apporter de leur notoriété. Ainsi, je remercie Dominique Duval Professeure à l'Université Joseph Fourier d'avoir bien voulu présider le jury. Je remercie également Denis Lugiez professeur à l'Université de Provence et Christopher Lynch Professeur à la Clarkson University d'avoir bien voulu accepter la lourde tâche de rapporter cette thèse. Mes remerciements s'adressent aussi à Bernhard Gramlich Associate Professor à la Technische Universität Wien pour avoir bien voulu faire partie du jury et évaluer ce travail.

À tout seigneur tout honneur. Je tiens à exprimer ma gratitude la plus profonde à Ricardo Caferra et Nicolas Peltier, mes directeurs de thèse pour leur amitié, pour leur sympathie, pour leurs valeurs humaines, pour leur disponibilité, pour leur implication totale dans ce travail, pour leur aide et leur soutien illimités même durant les moments les plus difficiles, pour leurs lectures minutieuses et leurs corrections répétées des différentes versions intermédiaires de ce manuscrit, enfin pour tout ce qu'ils ont fait pour moi. Qu'ils puissent trouver dans ces quelques lignes l'expression de mon admiration et de ma reconnaissance les plus profondes.

Une pensée particulière à l'Institut National des Postes et Télécommunications de Rabat qui m'a offert le cadre et les moyens pour un bon déroulement de cette thèse.

Enfin, je tiens à remercier chaleureusement ma famille et mes amis pour leurs encouragements et leur soutien durant ces années de thèse.

Table des matières

Table des figures

Liste des algorithmes

Chapitre 1

Introduction

Le commencement est la moitié du tout.
Platon

Pour situer toute recherche dans un certain domaine, il est naturel de vouloir commencer en donnant une définition dudit domaine. Dans [CLP04], les quelques définitions existantes de l'expression *raisonnement automatique* sont rappelées. Nous choisissons celle qui nous semble la plus pertinente :

> The subject of automated reasoning is concerned with using computers to help the humans discover and write formal proofs. R. Constable [Con93].

Bien entendu, la vision du domaine que l'on peut avoir à partir d'une telle définition dépend essentiellement de l'imagination et des connaissances préalables du lecteur. Un plus grand éclairage est obtenu en analysant la notion extrêmement subtile de *preuve* (essentiellement en mathématiques) et son évolution historique, et tout particulièrement les conséquences théoriques, pratiques et philosophiques de la possibilité de découvrir et de vérifier, de façon assistée ou automatique, des preuves par ordinateur. La notion de *preuve* et son évolution est naturellement liée à celle de la logique, et bien qu'il y a consensus sur le fait que la notion de *preuve* est due essentiellement aux Grecs, certains auteurs font remarquer que les babyloniens avaient déjà reconnu son importance (pour plus de détails voir e.g. [BW05, CM04, CCM04, CLP04]) :

> ... More important than the technical algebra of these ancient Babylonians is their recognition -as shown by their work- of the necessity of *proof* in mathematics.
>
> Until recently it has been supposed that the Greeks were the first to recognize that proof is demanded for mathematical propositions. This was one of the most important steps ever taken by human beings. Unfortunately it was taken so long ago that it led nowhere in particular so far as our own civilization is concerned -unless the Greeks followed conciously, which they may well have done. They were not particularly generous to their predecessors. [Bel86], page 18.

L'importance pratique de la notion de 'preuve' en Informatique est bien connue : par exemple pour la vérification et la preuve de programmes et du point de vue des fondements, le paradigme « proofs-as-programs » (correspondance de Curry-Howard).

Notre travail est orienté vers l'amélioration *qualitative* des démonstrateurs, en particulier dans le but de les munir de caractéristiques les rapprochant de ce qu'un utilisateur peut raisonnablement attendre. Ces attentes raisonnables grandissent naturellement avec l'évolution du domaine. Notre vision est très bien synthétisée dans [BW05] :

> The challenge is to make the systems more mathematician-friendly, by building libraries and tools. The eventual goal is to help humans to learn, develop, communicate, referee and apply mathematics.

Dans un travail de thèse, évidemment orienté vers l'avenir, les thèmes traités à la section de [BW05] intitulée « The nature of the challenge » sont particulièrement importants. Nous résumons brièvement les sous-sections qui nous semblent les plus intéressantes dans le cadre de notre travail :

- State of the art : systems.
  Les principaux systèmes existants, ainsi que les théorèmes les plus importants qu'ils ont permis de prouver sont répertoriés.
- What is needed ?
  (i) Mathematical style
  Les langages formels utilisés par la plupart des démonstrateurs automatiques et assistants de preuves actuels, continuent de constituer un frein à leur adoption et à leur utilisation à grande échelle par les mathématiciens. En effet, à cause des limites théoriques inhérentes aux logiques supportées par ces outils, les formalisations sont parfois loin d'être intuitives. Un effort dans ce sens est nécessaire.
  (ii) Library
  La présence d'une bibliothèque de lemmes et théorèmes préalablement prouvés serait selon les auteurs un facteur clé de la performance d'un assistant de preuves : Plus de résultats (déjà prouvés) réutilisables permet de résoudre plus de problèmes.
  (iii) Decision procedures
  Et les auteurs de conclure : « However *automated theorem proving* is surprisingly weak when it comes to finding proofs that are interesting to human mathematics. Worse, if one takes an existing informal textbook proof, and considers the gaps between the steps in that proof as *proof obligations*, then a general purpose theorem prover often will not even be able to find proofs for those ». Notre travail est une contribution dans ce sens (détection de saturation avec DEI) même s'il est vrai que les exemples traités dans le cadre de cette thèse restent très simples comparés à ceux qu'on pourrait trouver dans un livre de mathématique même de difficulté moyenne.
  (iv) Support for reasoning with gaps
  Sur ce point aussi nous apportons quelque chose dans l'aide à *la découverte de lemmes* (voir section 4.3.4). Un autre article important qui nous permet de situer notre travail est [CCM04] (voir aussi [CM04]). En partant du constat que *raison* et *expérience* sont deux façons d'acquérir des connaissances, les auteurs classifient les preuves mathématiques en 8 périodes, dont celles qui concernent le plus notre travail sont le 7ème et 8ème :

  > The seventh period belongs to the second half of the 20th century, when algorithmic proofs become acceptable only when there complexity where not too high. ... We are now living in this period. An important event of this period was the 1976 proof of the Four-Colour Problem (4CP) : it marked the reconciliation of empirical-experimental mathematics with deductive mathematics, ...
  >
  > With the eight stage, proofs are no longer exclusively based on logic and deduction, but also empirical and experimental.

Pour atteindre ou même approcher notre objectif d'apporter des améliorations qualitatives aux démonstrateurs automatiques existants nous avons étudié dans le cadre de cette thèse l'utilisation de la *schématisation de termes* en déduction automatique. Dans le même sens, Vincent Aravantinos étudie dans le cadre de sa thèse un autre type de schématisation ([Ara10, ACP10b, ACP10c, ACP10a, ACP09a, ACP09b, ACP09c, ACP08]), la *schématisation de preuves et de for-*

mules. Le lecteur pourra consulter [Ara10] pour plus de détails et notamment une discussion sur la notion de schématisation.

Dans ce mémoire nous adoptons le plan suivant : au chapitre 2, les définitions et notations de base sont introduites. Le chapitre 3 contient une rapide synthèse des différents résultats sur la schématisation de termes. Nous introduisons une définition de ce concept et rappelons la hiérarchie des différentes schématisations existantes ainsi que leurs principales propriétés. Au chapitre 4, nous présentons une approche fondée sur la détection de régularités dans l'espace de recherche pour permettre de *découvrir* des lemmes. La divergence du calcul est souvent liée à la présence de régularités dans l'espace de recherche. Cette régularité peut justement être formalisée en utilisant des schémas de termes. Nous décrivons également le logiciel DS3 qui permet, en analysant les formules engendrées par un démonstrateur, de détecter des régularités et d'induire sous forme de schématisation de termes, une description *structurelle* de certains ensembles de formules potentiellement déductibles à partir des axiomes. Ces descriptions peuvent être considérées comme des lemmes. Notre approche est inductive et non déductive, c'est-à-dire que la validité de ces lemmes n'est pas garantie. Nous montrons que la détection de lemme permet dans certains cas d'assurer la terminaison, ce qui est particulièrement utile dans le cas de preuve par consistance pour la démonstration de théorèmes inductifs. Naturellement, afin de valider ces lemmes, il faut être capable de *raisonner* de manière automatique sur les représentations ainsi engendrées. Bien que les propriétés des langages de schématisation soient à présent bien connues, leur intégration aux procédures de preuve existantes n'a reçu que peu d'attention. Nous y consacrons la troisième partie du mémoire. Ainsi, au chapitre 5 nous choisissons un formalisme particulier de schématisation de termes - les *I-termes* - et nous introduisons une extension des calculs de *résolution ordonnée* et de *superposition* intégrant des schématisations de termes. Les méthodes de *résolution* et de *superposition* sont les deux procédures les plus couramment utilisées pour la preuve automatisée de théorèmes en logique du premier ordre, respectivement sans et avec égalité. En particulier, nous abordons le problème de la *complétude réfutationnelle* du calcul étendu. La *complétude réfutationnelle* est la propriété assurant qu'un démonstrateur procédant par contradiction (preuve par l'absurde) est capable (au moins en théorie) de prouver toutes les formules valides d'un point de vue logique. Nous montrons que le calcul de superposition n'est pas complet lorsqu'il est appliqué sur les schématisations de termes et proposons une nouvelle règle d'inférence permettant de garantir la complétude. Au chapitre 6, nous présentons un algorithme d'indexation des I-termes. Ceci, dans l'optique de permettre des simplifications des clauses redondantes dans les calculs. Le chapitre 7 décrit le démonstrateur automatique DEI, que nous avons développé dans le cadre de cette thèse. Il s'agit d'une extension du démonstrateur E [Sch04] capable de raisonner avec des I-termes. À notre connaissance, DEI est le premier démonstrateur automatique utilisant des schématisations de termes. Le chapitre 8 conclut ce mémoire et présente quelques perspectives de recherche.

Du point de vue des techniques et concepts en déduction automatique, leur évolution historique, ainsi que sur l'état de l'art et des regards prospectifs, le lecteur peut faire un tour d'horizon très complet en consultant [RV01b, SW83a, SW83b, BL84].

Les travaux présentés dans ce manuscrit ont fait l'objet de publications. Ainsi, le chapitre 4 est fondé sur [BCP07], le chapitre 5 sur [BCP10a], le chapitre 6 sur [BCP10b] et le chapitre 7 sur [BCP09]. Toutes ces publications ont été partiellement financées par le projet ASAP de l'Agence Nationale de la Recherche (ANR-09-BLAN-0407-01). Ce projet (http ://membres-lig.imag.fr/peltier/ASAP) est mené conjointement entre l'équipe CAPP du Laboratoire d'Informatique de Grenoble et l'équipe Theoretische Informatik und Logik de l'Université Technique de Vienne.

Première partie

Définitions

Chapitre 2

Préliminaires

Ma cohabitation passionnée avec les mathématiques m'a laissé un amour fou pour les bonnes définitions, sans lesquelles il n'y a que des à-peu-près.

Gustave Flaubert

Nous commençons par introduire les définitions et notions usuelles utilisées dans ce mémoire. Ainsi, nous présenterons brièvement la syntaxe et la sémantique de la logique du premier ordre, nous donnerons ensuite un bref aperçu sur la démonstration automatique de théorèmes, en particulier les techniques de preuve par saturation utilisées dans le cadre de cette thèse. Les définitions suivantes sont adaptées de [Pel01b]. Pour une présentation plus détaillée de la logique du premier ordre, le lecteur pourra consulter par exemple [Bar77] ou [Caf09] qui contient plusieurs détails historiques et philosophiques sur le sujet. Il pourra également consulter [RV01b] pour un aperçu plus complet des différentes techniques utilisées en déduction automatique.

2.1 Termes du premier ordre

Pour définir le langage de la logique du premier ordre, nous considérons trois ensembles dénombrables et disjoints de symboles : Un ensemble infini de *variables ordinaires* V_X, un ensemble de *symboles de prédicats* $\mathcal{P}$ et un ensemble de *symboles de fonctions* $\mathcal{F}$. Nous supposons donnée une fonction *arité* qui associe à chaque élément f de $\mathcal{F}$ un entier (unique) indiquant le nombre d'arguments de f et à chaque élément P de $\mathcal{P}$ un entier indiquant le nombre d'arguments du prédicat P. Nous considérons un symbole de prédicat particulier, $\approx$ représentant l'égalité entre termes. Les variables seront dénotées par les lettres minuscules $x, y, \ldots$, les symboles de fonctions par $f, g, h, \ldots$ et les symboles de prédicats par $P, Q, R, \ldots$. Nous considérons également un ensemble de symboles appelés *symboles logiques* : $\Rightarrow, \neg, \vee, \wedge, \Leftrightarrow, \forall, \exists$. Nous notons $\mathcal{L}$ l'ensemble $\mathcal{L} = \mathcal{F} \cup \mathcal{P}$. Une ($\mathcal{L}$-)*expression* est une combinaison de symboles logiques, de symboles de V_X, de symboles de $\mathcal{F}$ et de symboles de $\mathcal{P}$. Nous nous intéressons à des formes particulières d'expressions, respectant des contraintes sur la combinaison des symboles (nous parlons d'*expression bien formée*).

2.1.1 Syntaxe

Il existe plusieurs formalismes possibles pour décrire des concepts, objets, relations, ... dans un domaine donné. Parmi les plus utilisés pour leur pouvoir d'expression et leur rigueur, le langage des prédicats ou langage (de la logique) du premier ordre. Dans ce langage, les *termes* sont utilisés comme unité de base pour représenter et décrire les objets d'un domaine de discours donné.

Définition 1 (Terme)**.** L'ensemble T des *termes du premier ordre* (ou simplement *termes*) est le plus petit ensemble vérifiant :
- Si $x \in V_X$ alors $x \in T$.
- Si $f \in \mathcal{F}$ et $arité(f) = 0$ alors $f \in T$. f est appelée *constante* dans ce cas.
- Si $f \in \mathcal{F}$, $arité(f) = n \geq 1$ et $t_1, \ldots, t_n \in T$ alors $f(t_1, \ldots, t_n) \in T$.

Un terme qui ne contient aucune variable est dit *fermé* (ou *clos*). ■

Les notions d'atome et de littéral permettent d'introduire des relations entre les termes (et d'infirmer ces relations). En effet, en logique du premier ordre, les *connaissances* sont représentées (entre autres) avec des termes reliés entre eux par des *relations* représentées elles mêmes par des prédicats.

Définition 2 (Atome,Littéral)**.** Un *atome* A est une expression telle que :
- Ou bien A est un symbole de prédicat d'arité 0. A est appelé dans ce cas *variable propositionnelle*.
- Ou bien A est de la forme $P(t_1, \ldots, t_n)$ où $P \in \mathcal{P}$, $arité(P) = n \geq 1$ et $t_1, \ldots, t_n \in T$.
- Ou bien A est de la forme $t \approx s$ où $t, s \in T$.

Un *littéral* est soit un atome A, on parle dans ce cas d'un *littéral positif*, soit la *négation* d'un atome A notée $\neg A$ et on parle dans ce cas d'un *littéral négatif*. Nous utilisons la notation $t \not\approx s$ comme raccourci pour $\neg(t \approx s)$. ■

La logique du premier ordre utilise un autre élément de langage très puissant, la *quantification*, déclinée sous deux formes : *universelle* et *existentielle*. La quantification universelle permet de construire des expressions représentant une infinité de possibilités, en faisant parcourir une variable, un domaine fixé. La quantification existentielle, permet d'exprimer l'*existence* d'un terme (objet) vérifiant certaines propriétés. Pour une discussion sur quelques implications philosophiques de l'existence, le lecteur pourra consulter par exemple [Caf09] et d'un point de vue technique, e.g. [Bee85].

La notion de *formule* (*du premier ordre*) capture le concept d'*énoncé logique* basé sur les *relations* entre termes (en utilisant les prédicats), les *connecteurs logiques* entre énoncés et l'introduction des quantificateurs pour augmenter le pouvoir d'expression.

Définition 3 (Formule)**.** L'ensemble des *formules du premier ordre* (ou $\mathcal{L}$*-formules*) (définissant également les *expressions bien formées*) est le plus petit ensemble d'expressions tel que :
- Si L est un atome alors L est une formule.
- Si ϕ est une formule alors $\neg\phi$ est une formule.
- Si ϕ et ψ sont deux formules alors, $\phi \Rightarrow \psi$, $\phi \Leftrightarrow \psi$, $\phi \vee \psi$ et $\phi \wedge \psi$ sont des formules.
- Si ϕ est une formule et x une variable alors $\forall x.\phi$ et $\exists x.\phi$ sont des formules.

■

Définition 4 (Variables libres)**.** L'ensemble des *variables libres* apparaissant dans un terme ou une formule noté $Vars$ est inductivement défini de la manière suivante :
- $Vars(a) \stackrel{def}{=} \emptyset$ si a est une constante.
- $Vars(x) \stackrel{def}{=} \{x\}$ si x est une variable.

- $Vars(f(t_1, \ldots, t_n)) \stackrel{def}{=} \bigcup_{i=1}^{n} Vars(t_i)$ si $f \in \mathcal{F}$.
- $Vars(P) \stackrel{def}{=} \emptyset$ si P est une variable propositionnelle.
- $Vars(P(t_1, \ldots, t_n)) \stackrel{def}{=} \bigcup_{i=1}^{n} Vars(t_i)$ si $P \in \mathcal{P}$.
- $Vars(\neg\phi) \stackrel{def}{=} Vars(\varphi)$.
- $Vars(\phi \bowtie \psi) \stackrel{def}{=} Vars(\phi) \cup Vars(\psi)$ où $\bowtie \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$.
- $Vars(\forall x.\phi) \stackrel{def}{=} Vars(\phi) \backslash \{x\}$.
- $Vars(\exists x.\phi) \stackrel{def}{=} Vars(\phi) \backslash \{x\}$.

■

Définition 5 (Variables liées)**.** L'ensemble des *variables liées* (ou *quantifiées*) dans une formule ϕ noté $V_q(\phi)$ est défini inductivement par :

- $V_q(P) \stackrel{def}{=} \emptyset$ si P est une variable propositionnelle.
- $V_q(P(t_1, \ldots, t_n)) \stackrel{def}{=} \emptyset$ si $P \in \mathcal{P}$ et $arité(P) = n$.
- $V_q(\neg\phi) \stackrel{def}{=} V_q(\phi)$.
- $V_q(\phi \bowtie \psi) \stackrel{def}{=} V_q(\phi) \cup V_q(\psi)$ où $\bowtie \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$.
- $V_q(\forall x.\phi) \stackrel{def}{=} V_q(\phi) \cup \{x\}$.
- $V_q(\exists x.\phi) \stackrel{def}{=} V_q(\phi) \cup \{x\}$.

■

Une expression (terme ou formule) t est dite *fermée* si $Vars(t) = \emptyset$ (*ouverte* sinon). Dans la suite nous ne considérons que des formules fermées. Si une formule ϕ est ouverte (c'est à dire contenant des variables libres, $Vars(\phi) \neq \emptyset$), elle est implicitement remplacée par sa *fermeture universelle* définie par $\forall x_1, \ldots, x_n.\phi$ où $Vars(\phi) = \{x_1, \ldots, x_n\}$.

Comme expliqué plus haut, l'utilité des variables est de fournir une abstraction permettant de représenter une infinité d'objets (termes) par une seule forme syntaxique. En effet, comme cela sera introduit dans la définition 9, une variable (quantifiée universellement) est destinée à parcourir un domaine donné quelconque en prenant toutes les valeurs de ce domaine. La notion de *substitution* permet de formaliser l'opération d'assignation d'une valeur (objet) à une variable donnée.

Définition 6 (Substitution)**.** Une *substitution* est une fonction qui associe à chaque variable un terme. L'image d'une variable x par une substitution σ est notée $x\sigma$ (au lieu de $\sigma(x)$).

Une substitution peut être étendue d'une manière directe aux termes et formules de la manière suivante :

- $a\sigma \stackrel{def}{=} a$ si a est une constante.
- $f(t_1, \ldots, t_n)\sigma \stackrel{def}{=} f(t_1\sigma, \ldots, t_n\sigma)$ si $f \in \mathcal{F}$ et $arité(f) = n$.
- $P\sigma \stackrel{def}{=} P$ si P est une variable propositionnelle.
- $(\neg\phi)\sigma \stackrel{def}{=} \neg(\phi\sigma)$.
- $(\phi \bowtie \psi)\sigma \stackrel{def}{=} (\phi\sigma) \bowtie (\psi\sigma)$ où $\bowtie \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$.
- $(\forall x.\phi)\sigma \stackrel{def}{=} \forall x.(\phi\sigma')$ où σ' est la substitution définie par : $y\sigma' = y\sigma$ si $y \neq x$ et $x\sigma = x$.
- $(\exists x.\phi)\sigma \stackrel{def}{=} \exists x.(\phi\sigma')$ où σ' est la substitution définie par : $y\sigma' = y\sigma$ si $y \neq x$ et $x\sigma = x$.

Le *domaine* $dom(\sigma)$ d'une substitution σ est l'ensemble des variables x telles que $x\sigma \neq x$. Une substitution de domaine $\{x_1, \ldots, x_n\}$ est parfois notée $\{x_1 \rightarrow x_1\sigma, \ldots x_n \rightarrow x_n\sigma\}$.

Si σ, θ sont deux substitutions, nous notons $\sigma\theta$ la *composition* de σ et θ. Une substitution σ est appelée une *instance* d'une substitution σ' si et seulement si il existe θ telle que $\sigma' = \sigma\theta$.

Une substitution σ est un *unificateur* de deux termes t, s si et seulement si $t\sigma = s\sigma$. Dans ce cas t et s sont dit *unifiables*.

Une substitution σ est dite *fermée* si et seulement si pour toute variable x, le terme $x\sigma$ est fermé. ■

Proposition 1. *Soient deux termes t, s unifiables. Il existe un unique unificateur σ (à un renommage des variables près) appelé* unificateur le plus général *de t et s et noté $\sigma = mgu(t, s)$ tel que tout unificateur θ de t, s est une instance de σ.* ■

Pour identifier les sous-termes d'un terme donné, nous introduisons la notion de *position* où chaque position d'un terme t correspond à un sous-terme de t.

Définition 7 (Position)**.** Une *position* est une suite d'entiers séparés par le symbole '.' . Λ désigne la *position vide* (une suite vide d'entiers).

L'*ensemble des positions* d'un terme t noté $Pos(t)$ est défini par :
- $Pos(t) = \{\Lambda\}$ si t est une variable ou une constante.
- $Pos(f(t_1, \ldots, t_n)) = \{\Lambda\} \cup \{i.q | q \in Pos(t_i), i \in [1..n]\}$.

■

Soient $t \in T$ et $p \in Pos(t)$. La notation $t|_p$ dénote le sous-terme de t apparaissant à la position p. Plus formellement nous avons :
- $t|_p = t$ si $p = \Lambda$.
- $f(t_1, \ldots, t_n)|_{i.q} = t_i|_q$ si $i \in [1..n]$ et $q \in Pos(t_i)$.

La notation $t[s]_p$ est utilisée pour désigner le terme obtenu à partir de t en remplaçant le sous-terme apparaissant à la position $p \in Pos(t)$ par s. Formellement, nous avons :
- $t[s]_p = s$ si $p = \Lambda$.
- $f(t_1, \ldots, t_n)[s]_{i.q} = f(t_1, \ldots, t_{i-1}, t_i[s]_q, t_{i+1}, \ldots, t_n)$ si $i \in [1..n]$ et $q \in Pos(t_i)$.

Il est possible d'étendre cette notation à un ensemble fini strictement ordonné (par une relation d'ordre stricte et totale $\prec$) de positions. Ainsi, $t[s]_{\mathcal{E}}$ peut être défini formellement par :
- $t[s]_{\emptyset} = t$.
- $t[s]_{\{q\}\cup\mathcal{E}} = (t[s]_q)[s]_{\mathcal{E}}$ si $q \in Pos(t)$ et $\forall p \in \mathcal{E}.q \prec p$.
- $t[s]_{\{q\}\cup\mathcal{E}} = t[s]_{\mathcal{E}}$ si $q \notin Pos(t)$ et $\forall p \in \mathcal{E}.q \prec p$.

Dans le cas où il n'y a pas de confusion possible, la même notation $t[s]_p$ peut être utilisée pour assurer que $t|_p$ et s sont syntaxiquement égaux, c'est à dire $t|_p = s$.

2.1.2 Sémantique

Nous présentons maintenant comment donner une *valeur* (ou *sémantique*) aux constructions syntaxiques introduites plus haut.

Définition 8 (Interprétation)**.** Une *interprétation* $\mathcal{I}$ est un couple $\mathcal{I} = (D, F)$ où D est un ensemble non vide appelé le *domaine* de $\mathcal{I}$ et F une fonction définie sur $\mathcal{F} \cup \mathcal{P}$ telle que :
- Si $c \in \mathcal{F}$ est un symbole de constante ($c \in \mathcal{F}$ et $arité(c) = 0$) alors $F(c)$ notée également $\mathcal{I}_c$ est un élément de D ($F(c) \in D$).
- Si $f \in \mathcal{F}$, $arité(f) = n$ et $n \geq 1$ alors $F(f)$ notée également $\mathcal{I}_f$ est une fonction de D^n dans D.
- Si P est une variable propositionnelle alors $F(P) \in \{vrai, faux\}$. $F(P)$ est notée également $\mathcal{I}_P$.
- Si $P \in \mathcal{P}$ et $arité(P) = n$ alors $F(P)$ notée également $\mathcal{I}_P$ est une fonction de D^n dans $\{vrai, faux\}$.

Nous étendons F aux variables en fixant pour chaque variable x une valeur unique dans D correspondant à $F(x)$ que nous notons également $\mathcal{I}_x$. ■

La notion d'interprétation permet d'assigner une *valeur* (*sémantique*) aux éléments syntaxiques (termes, formules,...).

Définition 9. La *valeur* d'un terme t ou d'une formule ϕ dans une interprétation $\mathcal{I}$, notée $\mathcal{I}(t)$ est définie inductivement par :

- Si t est une constante a, alors $\mathcal{I}(t) \stackrel{def}{=} \mathcal{I}_a$.
- Si t est de la forme $f(t_1, \ldots, t_n)$ alors $\mathcal{I}(t) \stackrel{def}{=} \mathcal{I}_f(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n))$.
- Si ϕ est un symbole propositionnel P alors $\mathcal{I}(\phi) \stackrel{def}{=} \mathcal{I}_P$.
- Si ϕ est de la forme $t = s$ alors $\mathcal{I}(\phi) = vrai$ si $\mathcal{I}(t) = \mathcal{I}(s)$ et $\mathcal{I}(\phi) = faux$ sinon.
- Si ϕ est un atome $P(t_1, \ldots, t_n)$ alors $\mathcal{I}_P(t) \stackrel{def}{=} \mathcal{I}_P(\mathcal{I}(t_1), \ldots, \mathcal{I}(t_n))$.
- Si $\phi = \phi_1 \vee \phi_2$ alors $\mathcal{I}(\phi) = vrai$ ssi $\mathcal{I}(\phi_1) = vrai$ ou $\mathcal{I}(\phi_2) = vrai$.
- Si $\phi = \phi_1 \wedge \phi_2$ alors $\mathcal{I}(\phi) = vrai$ ssi $\mathcal{I}(\phi_1) = vrai$ et $\mathcal{I}(\phi_2) = vrai$.
- Si $\phi = \phi_1 \Rightarrow \phi_2$ alors $\mathcal{I}(\phi) = vrai$ ssi $\mathcal{I}(\phi_1) = faux$ ou $\mathcal{I}(\phi_2) = vrai$.
- Si $\phi = \phi_1 \Leftrightarrow \phi_2$ alors $\mathcal{I}(\phi) = vrai$ ssi $\mathcal{I}(\phi_1) = \mathcal{I}(\phi_2)$.
- Si $\phi = \neg\psi$ alors $\mathcal{I}(\phi) = vrai$ ssi $\mathcal{I}(\psi) = faux$.
- Si $\phi = \forall x.\psi$ alors $\mathcal{I}(\phi) = vrai$ ssi pour tout élément v du domaine de $\mathcal{I}$ nous avons $\mathcal{I}[x \rightarrow v] \models \psi$, où $\mathcal{I}[x \rightarrow v]$ est l'interprétation coïncidant avec $\mathcal{I}$ sauf pour l'interprétation de x qui vaut v.
- Si $\phi = \exists x.\psi$ alors $\mathcal{I}(\phi) = vrai$ ssi il existe un élément v du domaine de $\mathcal{I}$ tel que $\mathcal{I}[x \rightarrow v] \models \psi$.

■

Une interprétation $\mathcal{I}$ est appelée *modèle* (resp. *contre-exemple*) d'une formule ϕ si $\mathcal{I}(\phi) = vrai$ (resp. $\mathcal{I}(\phi) = faux$). Dans le cas d'un modèle nous notons $\mathcal{I} \models \phi$, dans le cas contraire nous notons $\mathcal{I} \not\models \phi$. Une interprétation est *modèle d'un ensemble de formules* E si elle est modèle de toute formule de E.

Une formule ϕ est *satisfaisable* si elle admet au moins un modèle, *insatisfaisable* si elle n'a pas de modèles et *valide* si toute interprétation est un modèle de ϕ.

Une formule ϕ est une *conséquence logique* d'une formule ψ si et seulement si tout modèle de ψ est également un modèle de ϕ. Nous notons alors $\psi \models \phi$. Cette notion peut être étendue à des ensembles de formules de la façon suivante : un ensemble de formules F est une *conséquence logique d'un ensemble de formules* E si et seulement si toute interprétation qui valide l'ensemble des formules de E valide également toutes les formules de F.

2.2 Démonstration automatique et saturation

L'objectif de la démonstration automatique (de théorèmes) est de développer des *méthodes de preuve* (ou *calculs*) permettant de *prouver* (par ordinateur) qu'une ou plusieurs formules (les *conclusions*) sont des conséquences logiques d'un ensemble de formules appelées *prémisses*. Les techniques utilisées diffèrent bien entendu en fonction du type des formules considérées (et de leur pouvoir d'expression). Pour les formules (du premier ordre) introduites à la section précédente et que nous utilisons tout au long de ce travail, il existe deux grandes approches de techniques de preuve. La première est représentée par la méthode des *tableaux sémantiques*. Pour plus de détails concernant la méthode des tableaux, le lecteur pourra consulter par exemple [Fit90] et [Smu68]. Dans la deuxième approche que nous utilisons dans le cadre de ce travail, la procédure de preuve consiste à effectuer des manipulations (traitements) basées sur la forme des formules en utilisant des *règles d'inférence* qui permettent, étant données des formules (les *prémisses*), de construire de nouvelles formules (les *conclusions* conséquences directes (immédiates)) à partir de

ces prémisses jusqu'à obtenir une formule ayant une forme spéciale. En particulier, la technique de preuve la plus utilisée est la *preuve par réfutation*. Ainsi, pour prouver qu'une formule ϕ (ou ensemble de formules) est une conséquence logique d'un ensemble de formules E, il suffit de montrer que l'ensemble $\{\neg\phi\} \cup E$ est un ensemble insatisfaisable (remarquons que ce problème est à la base un problème sémantique). Le processus de recherche de preuve (réfutation) consiste alors à appliquer les règles d'inférence jusqu'à l'obtention d'une formule spéciale représentant la contradiction (la clause vide, voir définition 10). L'application des règles d'inférence dépend de la syntaxe des prémisses (en plus d'autres facteurs) (remarquons que ce deuxième problème est quant à lui syntaxique).

La preuve par réfutation nécessite un type particulier de formules appelées *clauses*.

Définition 10 (Clause)**.** Une *clause* est une disjonction finie éventuellement vide de littéraux $L_1 \vee \ldots \vee L_n$. Les variables d'une clause sont implicitement universellement quantifiées.

Une clause *vide* (ne contenant aucun littéral) est par convention équivalente à $faux$. La clause vide est généralement notée $\square$.

Une clause *positive* est une clause qui ne contient que des littéraux positifs.

Une clause *négative* est une clause qui ne contient que des littéraux négatifs.

Une clause est dite de *Horn* si et seulement si elle contient au plus un littéral positif. ■

Il est possible de transformer d'une manière totalement automatique, n'importe quelle formule (du premier ordre) ϕ en un ensemble (ou conjonction) de clauses S tel que ϕ est satisfaisable si et seulement si S l'est aussi. S n'est pas en général une conséquence logique de ϕ. Pour plus de détails sur cette opération dite de *normalisation au format FNC* (sigle de Forme Normale Conjonctive), le lecteur pourra consulter par exemple [BEL01].

Dans le reste de ce mémoire et sauf mention du contraire nous ne considérons que des clauses et des ensembles de clauses à l'exception de tout autre type de formules.

L'approche de *preuve par saturation*, consiste à appliquer les règles d'inférences sur un ensemble de clauses jusqu'à ce que la clause vide $\square$ soit générée, ou jusqu'à ce qu'aucune règle d'inférence ne puisse être appliquée. Cette procédure peut ne pas se terminer.

Définition 11 (Correction)**.** Une méthode de preuve (ou calcul) a la propriété de *correction* (ou tout simplement est dite *correcte*) si et seulement si toute clause déductible d'un ensemble de clauses E est une conséquence logique de E. ■

Définition 12 (Complétude réfutationnelle)**.** Une méthode de preuve (ou calcul) a la propriété de *complétude réfutationnelle* si et seulement si l'application des règles du calcul sur tout ensemble de clauses E insatisfaisable, permet de générer la clause vide. ■

Il est trivial de définir une procédure complète pour la réfutation (il suffit de déduire toujours la clause vide) ou correcte (il suffit de ne rien déduire). Ce qui est difficile est de définir une procédure ayant les deux propriétés à la fois (correction et complétude).

2.2.1 Cas non équationnel

La méthode de preuve par saturation de référence utilisée en l'absence du prédicat d'égalité est la méthode de *résolution* [Rob65b] définie par les règles d'inférence :

Résolution

$$\frac{C \vee A \qquad D \vee \neg B}{(C \vee D)\sigma}$$

où $\sigma = mgu(A, B)$.

Factorisation

$$\frac{C \vee A \vee B}{(C \vee B)\sigma}$$

où $\sigma = mgu(A, B)$.

L'utilisation directe du calcul de résolution engendre énormément de clauses ce qui rend la méthode pas très efficace. C'est pourquoi, plusieurs *stratégies* ont été proposées pour contrôler la taille de l'espace de recherche (l'ensemble des clauses générées par la procédure de preuve) et atténuer son explosion. Intuitivement, une *stratégie* est une restriction de l'application des règles d'inférence. Plusieurs types de stratégies ont été proposées. Plusieurs de ces stratégies parmi les plus efficaces utilisent un *ordre* sur les termes, littéraux et clauses pour instaurer les restrictions : si les conditions construites à partir de cet ordre ne sont pas satisfaites, les règles d'inférence sur lesquelles s'appliquent ces conditions ne sont pas appliquées.

Définition 13 (Relation d'ordre)**.** Une *relation d'ordre* $\succ$ sur un ensemble E est une relation :
- Irréflexive : $\forall t \in E, t \not\succ t$.
- Transitive : $\forall t, s, r \in E$, $(t \succ s) \wedge (s \succ r) \Rightarrow (t \succ r)$.

Une relation (d'ordre) est dite *totale* (sur un ensemble E) si et seulement si pour tous t et s, $t \succ s$ ou $s \succ t$ ou $t = s$.

Elle est *bien fondée* si et seulement s'il n'existe pas de suite infinie $t_1, \ldots, t_n, \ldots$ telle que $\forall i, t_i \succ t_{i+1}$. ■

Nous supposons donnée une relation d'ordre $\succ$ sur les atomes vérifiant les propriétés suivantes :

1. $\succ$ est totale sur les atomes fermés.
2. $\succ$ est une relation bien fondée.
3. $\succ$ est *fermée par substitution*, i.e. pour toute substitution σ et tous atomes A, B : $A \succ B \Rightarrow A\sigma \succ B\sigma$.

Un ordre sur les atomes peut être étendu aux littéraux en ignorant les symboles de négation comme formalisé dans la définition 14 ci-dessous pour le cas non équationnel (absence du prédicat $\approx$) :

Définition 14. Soit $\succ$ un ordre sur les atomes. L'*extension de* $\succ$ *aux littéraux* également notée $\succ$ est définie par :
- $A \succ \neg B \Leftrightarrow A \succ B$.
- $\neg A \succ B \Leftrightarrow A \succ B$.
- $\neg A \succ \neg B \Leftrightarrow A \succ B$.

■

Un ordre sur les littéraux permet de définir des *littéraux maximaux dans une clause* comme formalisé par la définition suivante :

Définition 15. Un littéral L_i est dit *maximal* dans une clause $C = L_1 \vee \ldots \vee L_n$ si et seulement si il n'existe pas de littéral L_j tel que $L_j \succ L_i$. ■

Un autre outil utilisé pour contrôler l'application des inférences est la notion de *fonction de sélection* que la définition suivante formalise.

Définition 16 (Fonction de sélection)**.** Soit $\succ$ un ordre sur les littéraux. Une *fonction de sélection sel* (par rapport à $\succ$) est une fonction qui associe à chaque clause $C = L_1 \vee L_2 \ldots \vee L_n$ un sous-ensemble de ses littéraux appelé *littéraux sélectionnés* tels que :

– Ou bien $sel(C)$ contient un littéral négatif.
– Ou bien $sel(C)$ contient *tous* les littéraux de C maximaux par rapport à $\succ$.

■

La stratégie de résolution la plus utilisée (et la plus efficace) est la stratégie dite de *résolution ordonnée* [Lei97, BG01].

Définition 17 (Résolution ordonnée $\mathcal{OR}$)**.** Le calcul de *résolution ordonnée* $\mathcal{OR}$ est défini par les deux règles d'inférence suivantes :

Résolution ordonnée

$$\frac{C \vee A \qquad D \vee \neg B}{(C \vee D)\sigma}$$

où $\sigma = mgu(A, B)$, $A\sigma$ et $\neg B\sigma$ sont sélectionnés dans $(C \vee A)\sigma$ et $(D \vee \neg B)\sigma$ respectivement.

Factorisation ordonnée

$$\frac{C \vee A \vee B}{(C \vee B)\sigma}$$

où $\sigma = mgu(A, B)$ et $B\sigma$ est sélectionné dans $(C \vee A \vee B)\sigma$. ■

Théorème 1 ([Lei97, BG01])**.** *Le calcul $\mathcal{OR}$ a la propriété de complétude réfutationnelle.* ■

Les fonctions de sélection permettent de décrire facilement des stratégies très diverses comme par exemple la *stratégie positive* qui consiste à prendre comme fonction de sélection une fonction retournant tous les littéraux négatifs (s'il existe des littéraux négatifs) ou tous les littéraux sinon. Une caractérisation de la stratégie positive est que l'une des deux prémisses de la règle de résolution doit obligatoirement être une clause positive. De manière duale, la *stratégie négative de résolution* est caractérisée par la nécessité que l'une des deux prémisses soit une clause négative (en plus des contraintes d'ordre).

2.2.2 Cas équationnel

L'égalité peut être traitée en ajoutant les axiomes de l'égalité :

1	$\forall x.$	$x \approx x$
2	$\forall x \forall y.$	$(x \approx y) \Rightarrow (y \approx x)$
3	$\forall x \forall y \forall z.$	$(x \approx y \wedge y \approx z) \Rightarrow (x \approx z)$
4	$\forall x_1, \ldots, x_n \forall y_1, \ldots, y_n.$	$(x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n) \Rightarrow$ $(f(x_1, \ldots, x_n) \approx f(y_1, \ldots, y_n))$
5	$\forall x_1, \ldots, x_n \forall y_1, \ldots, y_n.$	$(x_1 \approx y_1 \wedge \ldots \wedge x_n \approx y_n \wedge P(x_1, \ldots, x_n)) \Rightarrow$ $P(y_1, \ldots, y_n)$

où f et P désignent respectivement dans les axiomes 4 et 5 un symbole de fonction et un symbole de prédicat arbitraires d'arité n. 4 et 5 s'appliquent sur tous les symboles de fonction et de prédicat apparaissant dans la formule considérée.

Les axiomes $1, 2, 3$ indiquent que l'égalité est une relation d'équivalence, l'axiome 4 spécifie que l'égalité a la propriété de substitutivité de fonctions (ou tout simplement qu'elle est compatible avec les symboles de fonctions) et l'axiome 5 qu'elle a la propriété de substitutivité de prédicats (ou tout simplement qu'elle est compatible avec les symboles de prédicats (autres que l'égalité)). Il est facile d'établir que l'ajout de ces axiomes à un calcul ayant la propriété de complétude réfutationnelle conserve cette propriété.

En pratique, cette représentation de l'égalité pose des problèmes d'efficacité car un grand nombre de clauses *redondantes* (voir la définition 24) peut être engendré à partir de ces axiomes

(e.g. il est possible d'engendrer $f^n(a) \approx f^n(b)$ à partir de $a = b$). C'est pourquoi il est préférable de considérer le prédicat d'égalité comme appartenant à la syntaxe du langage. Il est alors nécessaire de disposer de règles d'inférence intégrant directement ce prédicat en particulier la substitution d'égal par égal dans toutes les positions (même les plus profondes) des clauses. La version la plus restrictive (par rapport aux applications possibles de la règle d'inférence) de ces calculs est le calcul de *superposition* [BG94]. Pour simplifier la présentation, il est possible de supposer que l'égalité est l'unique prédicat considéré (les atomes de la forme $P(t_1, \ldots, t_n)$ peuvent être codés par $P(t_1, \ldots, t_n) \approx true$).

Avant de présenter le calcul de superposition, nous avons besoin de formaliser les relations d'ordre sur les termes, littéraux et clauses dans le cas équationnel. Comme pour pour le cas non équationnel avec les atomes, nous supposons donnée une relation d'ordre $\succ$ sur les termes vérifiant les propriétés suivantes :

1. $\succ$ est totale sur les termes fermés.
2. $\succ$ est une relation bien fondée.
3. $\succ$ est *fermée par substitution*, i.e. pour toute substitution σ et tous termes $t, s : t \succ s \Rightarrow t\sigma \succ s\sigma$.

Pour étendre un ordre sur les termes aux littéraux et clauses dans le cas équationnel, nous avons besoin d'introduire la notion de *multi-ensemble*.

Définition 18 (Multi-ensemble)**.** Un *multi-ensemble* $\mathcal{M}$ sur un ensemble E est une fonction $\mathcal{M} : E \to \mathbb{N}$.

Un élément x *appartient* à $\mathcal{M}$ (ou x est un *élément* de $\mathcal{M}$) ssi $\mathcal{M}(x) > 0$.

Un multi-ensemble $\mathcal{M}$ est *vide* ssi $\forall x.\mathcal{M}(x) = 0$.

Un multi-ensemble $\mathcal{M}$ est *fini* ssi l'ensemble $\{x | \mathcal{M}(x) \neq 0\}$ est fini.

L'union de deux multi-ensembles $\mathcal{M}_1, \mathcal{M}_2$ notée $\mathcal{M}_1 \cup \mathcal{M}_2$ est un multi-ensemble $\mathcal{M}$ défini par $\forall x.\mathcal{M}(x) = \mathcal{M}_1(x) + \mathcal{M}_2(x)$.

L'intersection de deux multi-ensembles $\mathcal{M}_1, \mathcal{M}_2$ notée $\mathcal{M}_1 \cap \mathcal{M}_2$ est un multi-ensemble $\mathcal{M}$ défini par $\forall x.\mathcal{M}(x) = \min(\mathcal{M}_1(x), \mathcal{M}_2(x))$, où $\min(x, y)$ est le minimum de x, y.

Deux multi-ensembles $\mathcal{M}_1, \mathcal{M}_2$ sont *égaux* et nous notons $\mathcal{M}_1 = \mathcal{M}_2$ si et seulement si $\forall x.\mathcal{M}_1(x) = \mathcal{M}_2(x)$. ■

Intuitivement, un multi-ensemble est une extension de la notion d'ensemble où le nombre d'occurrences d'un élément est tenu en compte. En effet si $\mathcal{M}$ est un multi-ensemble sur un ensemble E, alors $\mathcal{M}(x)$ désigne le nombre d'occurrences de x dans E.

Exemple 1. Le multi-ensemble $\mathcal{M}$ sur $\{a, a, b, f(a), f(b)\}$ est défini par : $\mathcal{M}(a) = 2$, $\mathcal{M}(b) = \mathcal{M}(f(a)) = \mathcal{M}(f(b)) = 1$. ■

Parfois, la notation ensembliste ($\{a, a, b, f(a), f(b)\}$ comme dans l'exemple précédent) est utilisée pour représenter un multi-ensemble.

Définition 19. Soient $\mathcal{M}_1, \mathcal{M}_2$ deux multi-ensembles sur respectivement E_1, E_2 et soit $\succeq$ une relation d'ordre non stricte définie sur $E_1 \cup E_2$. L'*extension multi-ensemble* $\succeq_{mul}$ est la plus petite relation transitive telle que pour tous multi-ensembles $\mathcal{M}_1, \mathcal{M}_2$ et pour tout $a \in E_1 \cup E_2$, si pour tout $b \in \mathcal{M}_2$, $a \succeq b$ alors $\mathcal{M}_1 \cup \{a\} \succeq_{mul} \mathcal{M}_1 \cup \mathcal{M}_2$.

■

Pour plus de détails sur les relations d'ordre et les multi-ensembles, le lecteur pourra consulter par exemple [DM79, Der82].

Un littéral équationnel peut être représenté par un multi-ensemble. La définition suivante formalise cette représentation :

Définition 20. Soit $L = l \bowtie r$ un littéral. La *représentation multi-ensemble* de L notée $\mathcal{M}_L$ est un multi-ensemble défini par :
- $\mathcal{M}_L = \{l, l, r, r\}$ si L est un littéral positif (i.e. $\bowtie = \approx$).
- $\mathcal{M}_L = \{l, r\}$ si L est un littéral négatif (i.e. $\bowtie = \not\approx$).

■

L'extension multi-ensemble des littéraux permet de définir un ordre sur les littéraux dans le cas équationnel :

Définition 21. Soit $\succeq$ un ordre sur les termes. L'*extension de* $\succeq$ *aux littéraux* notée également $\succeq$ est définie par :

$$L_1 \succeq L_2 \Leftrightarrow \mathcal{M}_{L_1} \succeq_{mul} \mathcal{M}_{L_2}$$

pour tout littéraux L_1, L_2. ■

Une clause peut être représentée par un multi-ensemble dont les éléments sont les représentations multi-ensembles de ses littéraux. Par exemple, une clause $L_1 \vee L_2 \ldots \vee L_n$ sera représentée par le multi-ensemble $\{\mathcal{M}_{L_1}, \mathcal{M}_{L_2}, \ldots, \mathcal{M}_{L_n}\}$. Cette représentation permet d'étendre les relations d'ordre sur les termes et les littéraux introduites plus haut aux clauses. La définition suivante formalise cette extension.

Définition 22 (Clauses et relation d'ordre)**.** Soient $C_1 = L_1^1 \vee L_2^1 \ldots \vee L_n^1$, $C_2 = L_1^2 \vee L_2^2 \ldots \vee L_n^2$ deux clauses dont les représentations multi-ensembles sont respectivement $\mathcal{M}_1 = \{\mathcal{M}_{L_1^1}, \mathcal{M}_{L_2^1}, \ldots, \mathcal{M}_{L_n^1}\}$ et $\mathcal{M}_2 = \{\mathcal{M}_{L_1^2}, \mathcal{M}_{L_2^2}, \ldots, \mathcal{M}_{L_n^2}\}$. Soit $\succeq$ une relation d'ordre sur les littéraux. L'extension de $\succeq$ aux clauses également notée $\succeq$ est définie par :

$$C_1 \succeq C_2 \Leftrightarrow \mathcal{M}_1 \succeq_{mul} \mathcal{M}_2$$

■

Les notions de littéral maximal et de fonction de sélection introduites dans la définition 15 et la définition 16 respectivement peuvent naturellement être utilisée dans le cas équationnel.

Nous présentons maintenant le calcul de superposition [BG94].

Définition 23 (Le calcul de superposition $\mathcal{SC}$)**.** Le calcul de *superposition* $\mathcal{SC}$ est défini par les règles d'inférence suivantes :

Superposition (S)

$$\frac{Q \vee (l \approx r) \qquad D \vee (w[t]_p \bowtie u)}{(Q \vee D \vee (w[r]_p \bowtie u))\sigma}$$

où $\bowtie \in \{\approx, \not\approx\}$, $\sigma = mgu(l, t)$, $l\sigma \not\preceq r\sigma$, $w\sigma \not\preceq u\sigma$, $(l \approx r)\sigma$, $(w \bowtie u)\sigma$ sont sélectionnés dans $(Q \vee l \approx r)\sigma$ et $(D \vee w \bowtie u)\sigma$ respectivement, et t n'est pas une variable.

Résolution équationnelle (ER)

$$\frac{Q \vee (l \not\approx r)}{Q\sigma}$$

où $\sigma = mgu(l, r)$ et $(l \not\approx r)\sigma$ est sélectionné dans $(Q \vee l \not\approx r)\sigma$.

Factorisation équationnelle (EF)

$$\frac{Q \vee (t \approx s) \vee (l \approx r)}{(Q \vee (s \not\approx r) \vee (l \approx r))\sigma}$$

où $\sigma = mgu(l, t)$, $l\sigma \not\preceq r\sigma$, $t\sigma \not\preceq s\sigma$ et $(t \approx s)\sigma$ est sélectionné dans $(Q \vee l \approx r \vee t \approx s)\sigma$.

■

Théorème 2 ([BG94])**.** *Le calcul* $\mathcal{SC}$ *a la propriété de complétude réfutationnelle.* ■

2.2.3 Élimination de la redondance et saturation

Malgré les restrictions apportées à l'application des règles d'inférence par l'introduction des stratégies, la taille de l'espace de recherche augmente très souvent de manière incontrôlable ce qui peut entraîner la divergence des calculs dans certains cas. Plus important encore, certaines clauses produites ne sont pas nécessaires pour dériver la clause vide. Éliminer ce type de clauses permet d'alléger considérablement l'espace de recherche. La caractérisation de ces clauses inutiles est formalisée par la définition suivante :

Définition 24 (Redondance)**.** Une clause C est dite *redondante* par rapport à un ensemble de clauses E si pour chaque instance fermée $C\sigma$ de C, il existe n clauses $D_1, \ldots, D_n \in E$ et n substitutions fermées $\theta_1, \ldots, \theta_n$ telles que $D_i\theta_i \preceq C\sigma$ et $\{D_i\theta_i \mid i \in [1..n]\} \models C\sigma$. ■

La notion d'*ensemble de clauses saturé* est formalisée par la définition suivante.

Définition 25 (Ensemble saturé)**.** Un ensemble de clauses E est *saturé* (par rapport à un ensemble de règles d'inférence Γ) si et seulement si chaque clause dérivée à partir de E (en utilisant Γ) est redondante par rapport à E. ■

La notion de *subsomption* est un exemple de redondance de clauses.

Définition 26 (Subsomption)**.** Une clause C *subsume* une clause D si et seulement s'il existe une substitution σ telle que $C\sigma$ est une sous-clause de D. ■

Ainsi, toute clause subsumée par une clause D est redondante par rapport à $\{D\}$. De même, toute clause valide est redondante par rapport à n'importe quel ensemble de clauses.

La définition 24 permet d'enrichir le calcul par des règles de *simplification* ou d'*élimination de la redondance* visant à réduire l'espace de recherche. Ainsi, plusieurs règles de ce type ont été proposées dans la littérature. Ces règles sont des cas particuliers de l'élimination des clauses redondantes (puisqu'il est impossible en général de décider si une clause est redondante ou pas). Parmi les plus utilisées nous pouvons citer par exemple :

Règle de subsomption :

$$\frac{S \cup \{C\}}{S}$$

s'il existe une clause $D \in S$ qui subsume C en considérant C et D comme des multi-ensembles.

Règle d'élimination des tautologies :

$$\frac{S \cup \{P \vee \neg P \vee R\}}{S}$$

La clause $P \vee \neg P \vee R$ est trivialement valide et donc tout modèle de S est également modèle de $S \cup \{P \vee \neg P \vee R\}$ qui est donc redondante par rapport à S.

Règle de démodulation :

$$\frac{S \cup \{L[t]_p \vee R, t' \approx s' \vee R'\}}{S \cup \{L[s'\sigma]_p \vee R, t' \approx s' \vee R'\}}$$

si $t'\sigma = t$, $t'\sigma \succ s'\sigma$, $R'\sigma \subseteq R$ et $(t' \approx s')\,\sigma$ est maximal dans $(t' \approx s' \vee R')\,\sigma$.

Dans la règle de démodulation, la clause $L[t]_p \vee R$ est redondante par rapport à l'ensemble $\{L[s'\sigma]_p \vee R, t' \approx s' \vee R'\}$.

Pour ces règles de simplification, la conclusion remplace la prémisse dans l'espace de recherche.

La complétude réfutationnelle des calculs $\mathcal{OR}$ et $\mathcal{SC}$ établie par les théorèmes 1 et 2 est conservée en intégrant l'élimination de la redondance. Les théorèmes suivants établissent ce résultat.

Théorème 3 ([Lei97, BG01]). *Tout ensemble de clauses insatisfaisable E saturé par rapport à $\mathcal{OR}$ contient la clause vide.* ■

Théorème 4 ([BG94]). *Tout ensemble de clauses insatisfaisable E saturé par rapport à $\mathcal{SC}$ contient la clause vide.* ■

2.3 Preuves inductives

Les définitions inductives sont utilisées intensivement en mathématiques et en informatique. Ainsi, les nombres entiers, les listes, les arbres, ... sont tous formalisés par des définitions inductives. Il est plus difficile dans ce cas de faire des preuves sur les propriétés de ce type d'objets que dans le cas standard (introduit à la section 2.2). Ceci est du au fait que la validité des formules logiques dans ce cas doit être considérée par rapport à des interprétations spéciales comme par exemple les modèles de Herbrand minimaux (les définitions formelles seront introduites plus tard) et non par rapport à l'ensemble de toutes les interprétations possibles. Pour plus de détails sur la notion de définition inductive, le lecteur pourra consulter [Acz77].

Définition 27 (Interprétation de Herbrand). Une interprétation $\mathcal{I}$ est dite de *Herbrand* si :
- Le domaine de $\mathcal{I}$ est un quotient par la relation d'égalité de l'ensemble des termes fermés sur le langage $\mathcal{L}$.
- Pour tout symbole de fonction f d'arité n et pour tous termes fermés $t_1, \ldots, t_n$, $\mathcal{I}(f(t_1, \ldots t_n)) \stackrel{def}{=} f(t_1, \ldots t_n)$.

Une interprétation $\mathcal{I}$ est un *modèle de Herbrand* d'une formule ϕ si $\mathcal{I} \models \phi$ et $\mathcal{I}$ est une interprétation de Herbrand. ■

Définition 28 (Modèle minimal de Herbrand). À chaque interprétation de Herbrand $\mathcal{I}$ correspond un ensemble (potentiellement infini) de littéraux $(L_i)_i$ tel que pour chaque i, $L_i = P_i(t_1, \ldots, t_n)$, $P_i \in \mathcal{P}$, pour chaque $j \in [1..n]$, t_j est un terme fermé construit sur les symboles de $\mathcal{F}$ et $\mathcal{I} \models L_i$. Un tel ensemble est noté $\mathcal{I}_H$. Un modèle de Herbrand $\mathcal{I}$ d'un ensemble S est *minimal* si est seulement si pour tout modèle de Herbrand $\mathcal{J}$ de S $\mathcal{I}_H \subseteq \mathcal{J}_H$. ■

Le modèle minimal n'est pas toujours unique. Cependant pour certains type de formules il l'est. Par exemple, tout ensemble de clauses de Horn satisfaisable admet un modèle minimal unique.

Nous formalisons maintenant la notion de conséquence inductive.

Définition 29. Une formule du premier ordre φ est une *conséquence inductive* d'un ensemble de formules du premier ordre E si et seulement si pour chaque interprétation de Herbrand $\mathcal{H}$, $\mathcal{H} \models E$ implique que $\mathcal{H} \models \varphi\sigma$ pour toute substitution fermée σ. ■

Exemple 2. Soit la définition suivante de la concaténation de listes où *nil* représente la liste vide, *cons* est l'opérateur de construction et *append* désigne la concaténation de deux listes :

$$L = \left\{ \begin{array}{lrcl} \forall l & append(nil, l) & \approx & l \\ \forall x, l, l' & append(cons(x, l), l') & \approx & cons(x, append(l, l')) \end{array} \right.$$

$append(l, nil) \approx l$ n'est pas une conséquence logique de L. En effet soit l'interpétation $\mathcal{I}$ dont le domaine est $\{nil, c\}$ où c est une constante différente de *nil* et dont l'interprétation de *cons* désignée par $cons_{\mathcal{I}}$ est définie par : $\forall x, l. cons_{\mathcal{I}}(x, l) = l$ et dont l'interprétation de *append* notée $append_{\mathcal{I}}$ est définie par : $\forall l, l'. append_{\mathcal{I}}(l, l') = l'$.

$\mathcal{I}$ est un modèle de L : $append_{\mathcal{I}}(nil, l) = l$ et $append_{\mathcal{I}}(cons(x, l), l') = append_{\mathcal{I}}(l, l') = cons(x, append_{\mathcal{I}}(l, l')) = l'$ mais $\mathcal{I}$ n'est pas un modèle de $append_{\mathcal{I}}(l, nil) = l$ puisque $append_{\mathcal{I}}(l, nil) = nil \neq l$.

$append(l, nil) \approx l$ est en revanche une conséquence inductive de L. Ceci peut être démontré par induction sur la structure des listes. ■

Nous pouvons distinguer deux approches pour la démonstration de théorèmes inductifs. La première approche est basée sur des schémas d'induction explicites (voir [Bun01] pour un tour d'horizon sur le sujet). Par exemple, pour démontrer qu'une propriété P est inductivement valide sur les entiers, il suffit de démontrer que la formule $P(0) \wedge (P(n) \Rightarrow P(n+1))$ est valide. Le second type d'approche englobe toutes les procédures de preuve où le schéma inductif n'est pas explicite. Ces procédures vont de l'induction implicite [BKR92, BR95] à la méthode dite « induction sans induction » [Com94, CN98, Com01] (appelée également *preuve par consistance*). Dans la suite de cette section, nous détaillons cette méthode que nous utilisons notamment au chapitre 4.

Induction sans induction

Dans la suite de cette section, E désigne un ensemble de clauses appelées *axiomes*, C désigne un ensemble de clauses appelées *conjectures*. L'objectif est de prouver que C est une conséquence inductive de E. Pour simplifier la présentation, nous ne considérons que le cas où E ne contient que des clauses de Horn avec l'égalité comme seul prédicat. Ainsi, si E est satisfaisable alors son modèle de Herbrand minimal existe et est unique. Pour le traitement des cas où E contient des formules autres que les clauses de Horn avec potentiellement d'autres prédicats, le lecteur pourra consulter [CN98] dont nous reprenons les notations et définitions, en particulier, $\mathcal{I}$ désigne le modèle de Herbrand minimal de E.

Dans la preuve par induction sans induction nous nous intéressons au problème suivant : Si $\mathcal{I}$ est le modèle de Herbrand minimal d'un ensemble de clauses (de Horn dans notre cas) E et si C est un ensemble de clauses (également de Horn), est ce que $\mathcal{I} \models C$?

L'idée de base dans cette méthode est d'utiliser un ensemble A tel que $A \cup E \cup C$ est consistant si et seulement si $\mathcal{I} \models C$. Un tel ensemble est appelé *I-axiomatisation* (la définition 30 formalise cette notion).

Définition 30. Soit E un ensemble de clauses de Horn satisfaisable et soit $\mathcal{I}$ son modèle de Herbrand minimal. Un ensemble de formules du premier ordre A est une *I-axiomatisation* de E si :

- A ne contient que des formules purement universelles (ou quantifiées universellement).
- $\mathcal{I}$ est l'unique modèle de Herbrand de $E \cup A$ à un isomorphisme prêt.

■

Pour plus d'informations sur la manière de calculer une I-axiomatisation dans différents cas, le lecteur pourra consulter [CN98].

La proposition suivante permet de ramener le problème de savoir si C a pour modèle $\mathcal{I}$ à un problème de vérification de la consistance d'un ensemble de clauses :

Proposition 2 ([CN98])**.** *Soit E un ensemble de clauses de Horn satisfaisable, $\mathcal{I}$ son modèle de Herbrand minimal et soit A une I-axiomatisation de E. $A \cup E \cup C$ est consistant si et seulement si $\mathcal{I} \models C$.* ■

Ainsi, le problème (est ce que $\mathcal{I} \models C$?) est ramené à un problème de vérification de consistance. Ceci fait la force de cette méthode puisque il suffit d'utiliser les méthodes de preuves par

saturation standards (en utilisant des stratégies spécifiques). Il faut noter que pour établir la consistence, il est nécessaire que les méthodes de preuve terminent – ce qui rend crucial le choix de la stratégie.

Induction sans induction et preuves inductives.

Le problème traité par l'induction sans induction est différent de celui de l'établissement de preuves inductive. En effet, une clause C qui a pour modèle $\mathcal{I}$ n'est pas forcément conséquence inductive de E.

Exemple 3 ([Com01])**.** Nous considérons la définition de l'opération d'addition + sur les entiers :

$$E = \left\{ \begin{array}{rcl} 0 + x & \approx & x \\ s(x) + y & \approx & s(x + y) \end{array} \right.$$

Le modèle de Herbrand minimal $\mathcal{I}$ de E est l'ensemble $\mathbb{N} = \{0, s(0), s(s(0)), \ldots\}$. La clause $C = (x \not\approx s(x))$ n'est pas une conséquence inductive de E (un modèle de Herbrand ne contenant que l'élément 0 n'est pas un modèle de C), en revanche, $\mathcal{I} \models C$. ■

Cependant, si C ne contient que des clauses positives, les deux problèmes sont équivalents comme le montre le lemme suivant :

Lemme 3 ([CN98])**.** *Soient E un ensemble de clauses (de Horn) satisfaisable, $\mathcal{I}$ son modèle de Herbrand minimal et c une clause positive. c est une conséquence inductive de E si et seulement si $\mathcal{I} \models c$.* ■

Si en revanche, C contient des clauses avec des littéraux négatifs, le deux problèmes ne coïncident pas comme illustré par l'exemple 3.

Ainsi, l'induction sans induction permet de définir une procédure de preuve ayant ce qui peut être appelée la propriété de *complétude réfutationnelle inductive* (selon la terminologie utilisée dans [CN98]), i.e. pour tout ensemble C tel que $\mathcal{I} \not\models C$ (et donc C n'est pas une conséquence inductive de E), l'induction sans induction termine.

Chapitre 3

Schématisation de termes

qui dans sa main tient une même boîte sur le couvercle de laquelle sa même image se répète, comme dans ces jeux de miroir sans fin.

Claude Simon

Les termes du premier ordre permettent de représenter une infinité d'objets (en général, on s'intéresse exclusivement aux termes fermés) avec une notation finie en utilisant les variables. Une variable permet de réaliser une abstraction d'un ensemble infini (l'ensemble de tous les termes fermés). Le but des *schématisations de termes* est de représenter en utilisant une abstraction plus précise que celle avec des variables et pouvant tenir compte de la *structure* des termes un ensemble infini d'objets (les clauses par exemple) d'une manière finie. Par exemple, soit l'ensemble d'axiomes S suivant :

$$S = \left\{ \begin{array}{cc} P(0) & \\ \forall x.P(x) & \Rightarrow \quad P(s(s(x)) \end{array} \right.$$

Le calcul de résolution appliqué à S peut engendrer la suite (infinie) de clauses $\{P(0), P(s(s(0))), \ldots, P(s^{2n}(0)), \ldots\}$ ce qui entraîne la divergence.

Dans cet exemple, nous remarquons que les éléments de la suite générée « se ressemblent » de par leur structure, ainsi, nous pouvons remplacer cette liste infinie par *une seule* formule compacte : $\forall N.P(s^{2N}(0))$ où N est une variable arithmétique (à valeur dans $\mathbb{N}$) et $s^{2N}(0)$ est un terme défini à l'aide d'une extension de la syntaxe de la logique du premier ordre usuelle. Nous parlons alors de *schématisation de termes*.

Il est vrai que dans ce cas très simple, l'utilisation de restrictions induites par une relation d'ordre judicieusement choisie (ou par une stratégie de résolution négative), permet de limiter l'application de la règle de résolution et par conséquent restaure la convergence. Cependant, il est facile de construire des exemples plus complexes pour lesquelles le choix de l'ordre n'est pas évident, voire pour lequel aucun ordre ne convient :

Exemple 4. Soit l'ensemble de clauses $\{Q(x, f(y)) \Leftrightarrow Q(f(x), y)$, $Q(x, x)$, $\neg Q(f(x), x)$, $\neg Q(x, f(x))\}$. Les clauses engendrées sont suivant la stratégie : $Q(f^{2n}(x), x)$, $Q(x, f^{2n}(x))$, $\neg Q(f^{2n+1}(x), f(x))$ ou $\neg Q(f(x), f^{2n+1}(x))$. ■

D'autre part, même dans le cas très simple ci-dessus (l'ensemble S), l'utilisation des schématisations peut avoir un intérêt : si nous considérons des stratégies basées sur des raffinements positifs (ou bien stratégies positives) -très utilisées par ailleurs en construction de modèles (voir

e.g. [CLP04])- comme l'hyper-resolution[1] [Rob65a] ou l'hyper-tableaux [Bau98], la convergence n'est plus assurée.

Plusieurs formalismes de schématisation de termes on été proposés dans la littérature. Ces formalismes ont été introduits justement pour empêcher la divergence dans les procédures de calcul symbolique particulièrement dans le contexte de la réécriture. Une notion clé dans la schématisation de termes est celle de *patron inductif* (ou *contexte inductif*) appliqué à un *terme de base*. Par exemple, pour chaque entier n, le terme $\underbrace{s(s(\ldots(0)\ldots))}_{n} = s^{2n}(0)$ est obtenu en *répétant* le *contexte inductif* $s(s(.))$ n fois et en l'appliquant au *terme de base* 0. Les définitions formelles seront introduites plus loin dans ce chapitre. La différence principale entre ces formalismes réside dans le type du patron (contexte) inductif autorisé. Ainsi le concept de termes récurrents a été initialement introduit dans [CHK90]. Plusieurs extensions et améliorations ont ensuite été proposées à partir de ce formalisme instaurant une hiérarchie de langages de schématisation de termes, avec différents pouvoirs d'expression. Nous retrouvons par ordre croissant du pouvoir d'expression les termes inductifs ou ρ-termes, les termes avec exposants entiers ou I-termes de H. Comon [Com95], les R-termes de G. Salzer [Sal92] et les grammaires primales de M. Hermann et R. Galvabý [HG97]. L'utilité de ce type de formalismes en déduction automatique paraît évidente comme pour l'exemple précédent d'introduction, où il est possible de remplacer une infinité de clauses par une seule formule compacte comme indiqué plus haut. Plus encore, les longueurs des preuves et leur lisibilité peuvent nettement être améliorées grâce aux schématisations de termes. Ainsi, toujours à partir de l'exemple précédent, Nous pouvons prouver la formule $P(s^{2k}(0))$, étant donné un entier k en une seule étape, alors qu'il aurait fallu k inférences sans utilisation de la schématisation.

Dans ce chapitre nous présentons formellement cette hiérarchie. En particulier, nous présentons les ρ-termes, les I-termes, les R-termes et les grammaires primales.

3.1 Définitions préliminaires

Nous considérons un nouvel ensemble dénombrable V_N disjoint de V_X, de $\mathcal{F}$ et de $\mathcal{P}$. V_N est appelé l'*ensemble des variables arithmétiques*. Nous considérons également un symbole spécial $\diamond$ n'appartenant pas à l'ensemble $V_X \cup V_N \cup \mathcal{F}$. Le symbole $\diamond$ est appelé un *trou*. Il est utilisé pour définir les contextes inductifs. Les variables arithmétiques seront dénotées par les lettres majuscules $M, N, \ldots$ tandis que les variables ordinaires seront dénotées par les lettres minuscules $x, y, \ldots$.

Nous considérons également l'ensemble E_N qui désigne l'ensemble des *expressions arithmétiques* construites (de manière usuelle) sur la signature 0, s, $+$ et V_N. 0, s et $+$ sont interprétés comme dans l'arithmétique de Presburger.

Définition 31. L'ensemble E_N des *expressions arithmétiques* est le plus petit ensemble tel que :
- $\mathbb{N} \subset E_N$.
- $V_N \subset E_N$.
- Si $k \in \mathbb{N}, n \in E_N$ alors $k \times n \in E_N$ (parfois s'il n'y a pas d'ambiguïté, $k \times n$ sera notée simplement $k.n$ ou kn).
- Si $n, m \in E_N$ alors $n + m \in E_N$.

■

[1] Le calcul d'*hyper-resolution* est une extension du calcul de résolution qui permet d'effectuer en un seul pas plusieurs inférences réalisées par résolution simple. Dans ce calcul il est possible d'avoir plusieurs (clauses) prémisses (plus de deux).

Exemple 5. Les expressions suivantes sont des expressions arithmétiques : 2, N, $3.N$, $1+2.N+3.N$ où $N, M \in V_N$.

Les expressions suivantes ne sont pas des expressions arithmétiques : $N.M$, N^2, $1+2.N+M^2$, t où t est un terme du premier ordre (en particulier une constante ou une variable non arithmétique). ■

Dans l'exemple précédent, les expressions $N.M$, N^2 et $1+2.N+M^2$ ne sont pas considérées comme expressions arithmétiques car elle contiennent un produit de deux variables arithmétiques (le produit est interdit par la définition). Les termes du premier ordre sont également interdits, en particulier les variables (standards).

3.2 ρ-termes

Ces termes ont été introduit initialement par Chen, Hsiang et Kong dans [CHK90] sous l'appellation de *hyper-termes*, puis repris et redéfinis dans [CH91] sous le nom ρ-termes (ou ω-termes).

3.2.1 Syntaxe

La définition suivante présente formellement l'ensemble des ρ-termes. Nous considérons comme pour le cas des termes standards, un ensemble $\mathcal{F}$ de symboles de fonctions et un ensemble (infini dénombrable) V_X de variables standards.

Définition 32 (ρ-termes)**.** L'ensemble des *ρ-termes* T_ρ et l'ensemble des ρ-termes à un seul trou $T_\rho^\diamond$ sont les plus petits ensembles vérifiant :

- $\diamond \in T_\rho^\diamond$ et $V_X \subset T_\rho$.
- Si $t_1, \ldots, t_k \in T_\rho$, pour tout $i \in [1..k]$, $f \in \mathcal{F}$ et $arité(f) = k$ alors $f(t_1, \ldots, t_k) \in T_\rho$.
- Si $f \in \mathcal{F}$, $arité(f) = k > 0$, $t_j \in T_\rho^\diamond$ pour $j \in [1..k]$ et t_i est un terme standard fermé pour tout $i \neq j$,$i \in [1..k]$,alors $f(t_1, \ldots, t_k) \in T_\rho^\diamond$.
- Si $t \in T_\rho^\diamond$, $t \neq \diamond$, s est un terme standard fermé et $N \in E_N$ alors $t^N.s \in T_\rho$. Un ρ-terme de cette forme est appelé un N-terme. t est appelé dans ce cas un *contexte inductif* (ou itératif) et s un *terme de base*. Ces appellations (contexte inductifs et terme de base) sont utilisées également pour les I-termes et R-termes introduits respectivement aux sections 3.3 et 3.4.

Soit t un ρ-terme ($t \in T_\rho$). $var(t)$ désigne l'*ensemble des variables* de t appartenant aussi bien à V_X que V_N et $NVar(t)$ désigne l'*ensemble des variables arithmétiques* de t (appelées *variables exposant* dans la terminologie des ρ-termes). Un ρ-terme t est dit *fermé* si $var(t) = \emptyset$. ■

Exemple 6 (ρ-termes)**.** $f(a, g(\diamond))^N.b$ et $f(\diamond)^3.a$ sont des ρ-termes. ■

3.2.2 Sémantique

Comme c'est le cas pour les autres types de schématisation de termes, les ρ-termes permettent de représenter de manière finie un ensemble (éventuellement) infini de termes (standards) partageant une même structure itérée. Par définition, les termes standards (fermés) sont bien évidemment aussi des (cas particulier de) ρ-termes, par exemple, $f(a, b)$ est aussi bien un terme fermé standard qu'un ρ-terme. Si nous prenons par exemple un ρ-terme de la forme $t = f(\diamond)^N.a$ où $N \in V_N$ alors t désigne en fait un ensemble infini correspondant à toutes les itérations possibles (il y en a une infinité puisque N doit parcourir l'ensemble $\mathbb{N}$) du contexte $f(.)$, soit l'ensemble (infini) $\{a, f(a), f(f(a)) \ldots\}$.

Si t est un contexte inductif, nous dénotons par $t[s]_\diamond$ le terme obtenu à partir de t en remplaçant l'unique occurrence du trou dans t par s. Formellement, la notation $t[s]_\diamond$ est définie inductivement :

- $\diamond[s]_\diamond = s$.
- $f(t_1, \ldots, t_n)[s]_\diamond = f(t_1, \ldots, t_{i-1}, t_i[s]_\diamond, t_{i+1}, \ldots, t_n)$ où t_i contient le trou.

Si t est un contexte inductif et $n \in \mathbb{N}$, t^n désigne le contexte inductif $t^n = \underbrace{t[t[\ldots[\diamond]_\diamond \ldots]_\diamond]_\diamond}_{n}$. Il est clair (par récurrence) que si t est un contexte inductif, alors t^n est également un contexte inductif pour tout $n \in \mathbb{N}$.

Formellement, la sémantique des ρ-termes est définie par le système de réécriture R_ρ suivant :

$$R_\rho = \left\{ \begin{array}{rcll} t^0.s & \rightarrow & s & t \in T_\rho^\diamond, s \text{ est un terme standard fermé} \\ t^{n+1}.s & \rightarrow & t[t^n.s]_\diamond & n \in \mathbb{N}, t \in T_\rho^\diamond, s \text{ est un terme standard fermé} \end{array} \right.$$

Proposition 4. *Le système de réécriture R_ρ est convergent. De plus, il réduit avec les règles de l'arithmétique de Presburger, chaque ρ-terme sans variables arithmétiques en un terme standard.* ■

Preuve. Par induction sur la structure des ρ-termes. □

Comme le système R_ρ est convergent, nous pouvons définir pour chaque ρ-terme t une forme normale unique que nous notons $t \downarrow_\rho$.

La notion de substitution introduite pour les termes standards peut être étendue en une fonction qui associe à chaque variable standard un ρ-terme et à chaque variable arithmétique, une expression arithmétique. Comme pour le cas standard, pour chaque expression (terme, clause, variable arithmétique, ...) $\mathcal{E}$, $\mathcal{E}\sigma$ désigne l'expression obtenue en remplaçant chaque variable $x \in V_X \cup V_N$ apparaissant dans $\mathcal{E}$ par son image par σ. $\mathcal{E}\sigma$ est appelée *instance* de $\mathcal{E}$ comme pour le cas standard.

Définition 33. Une substitution σ est dite $\mathbb{N}$-*clôturante* si et seulement si $dom(\sigma) = V_N$ et $\forall N \in V_N, N\sigma \in \mathbb{N}$. Elle est *fermée* si et seulement si $\forall x \in dom(\sigma) \cap V_X$, $x\sigma$ est un terme fermé et $\forall N \in dom(\sigma) \cap V_N$ $N\sigma \in \mathbb{N}$. En pratique, le domaine de σ n'a pas à être infini : il peut être restreint uniquement aux variables apparaissant dans les termes considérés. L'ensemble des substitutions $\mathbb{N}$-clôturantes d'un (ρ-)terme t est noté $\Sigma_g(t)$. ■

Une $\mathbb{N}$-instance d'un ρ-terme est obtenue en instanciant toutes les variables entières apparaissant dans t par des entiers. Les variables standards *ne sont pas* instanciées.

Définition 34 ($\mathbb{N}$-instances)**.** Un terme s est dit $\mathbb{N}$-instance d'un ρ-terme t si et seulement s'il existe une substitution $\mathbb{N}$-clôturante σ telle que $s = t\sigma \downarrow_\rho$. ■

Par exemple, $g(a, x)$, $g(f(a), x)$ et $g(f(f(a)), x)$ sont des $\mathbb{N}$-instances du I-terme $t = g(f(\diamond)^N.a, x)$, alors que les termes $g(f(a), b)$ et $g(f(\diamond)^{N+1}.a, b)$ ne le sont pas même si se sont des instances au sens usuel de t.

La définition suivante formalise la sémantique des ρ-termes :

Définition 35 (Sémantique des ρ-termes)**.** L'ensemble des termes standards *représentés* par un ρ-terme t est l'ensemble $\{t\sigma \downarrow_\rho | \sigma \in \Sigma_g(t)\}$. ■

Exemple 7. $f(a, g(\diamond))^N.b$ représente l'ensemble infini $\{b, f(a, g(b)), f(a, g(f(a, g(b)))), \ldots\}$ et $f(\diamond)^3.a$ représente le terme standard $f(f((f(a))))$. ■

L'unification des ρ-termes est décidable et il existe un algorithme d'unification introduit dans [CHK90].

3.3 I-termes

Le pouvoir d'expression des ρ-termes présentés à la section précédente a des limites que l'on voudrait pouvoir surmonter. En effet, par définition, le terme de base d'un ρ-terme ne peut être qu'un terme (standard) fermé et le contexte inductif ne peut contenir que des termes fermés. Ainsi, par exemple ni $f(\diamond)^3.x$ ni $f(\diamond)^3.f(\diamond)^2.a$ ne sont des ρ-termes puisque le terme de base du premier est une variable et celui du deuxième est un ρ-terme. De même $f(x,\diamond)^3.a$ et $f(g(\diamond)^2.a,\diamond)^3.a$ ne sont pas non plus des ρ-termes puisque dans le premier cas le contexte inductif contient une variable et dans le deuxième il contient un ρ-terme (imbrication de ρ-termes). Le formalisme des termes à exposants entiers (I-termes)[Com95] a été donc introduit pour surmonter ces limites.

Nous commençons par définir la syntaxe des I-termes ainsi que des termes avec un (seul) trou qui formalisent les contextes inductifs des I-termes. Ces formalismes ont été introduits initialement dans [Com95]. Pour plus de clarté et de lisibilité, les exemples sont donnés avant les définitions.

Exemple 8 (I-termes)**.** $f(a,\diamond,b)^N.g(c)$ est un I-terme dénotant l'ensemble infini de termes standards $\{g(c), f(a,g(c),b), f(a,f(a,g(c),b),b),\ldots\}$. ■

Définition 36. L'ensemble des *termes à exposants entiers* T_I (ou I-termes) et l'ensemble des termes à un seul trou $T_\diamond$ sont les plus petits ensembles vérifiant :

- $\diamond \in T_\diamond$ et $V_X \subset T_I$.
- Si $t_1,\ldots,t_k \in T_I$, $f \in \mathcal{F}$ et $arité(f) = k$ alors $f(t_1,\ldots,t_k) \in T_I$.
- Si $f \in \mathcal{F}$, $arité(f) = k > 0$, $t_j \in T_\diamond$ pour $j \in [1..k]$ et $t_i \in T_I$, $i \neq j$ alors $f(t_1,\ldots,t_k) \in T_\diamond$.
- Si $t \in T_\diamond$, $t \neq \diamond$, $s \in T_I$ et $N \in E_N$ alors $t^N.s \in T_I$. Comme pour les ρ-termes, un I-terme de cette forme est appelé un *N-terme*, t est appelé *contexte inductif* et s *terme de base.*

■

Soit t un I-terme ($t \in T_I$). Comme pour les ρ-termes, $var(t)$ désigne l'*ensemble des variables* de t appartenant aussi bien à V_X que V_N et $NVar(t)$ désigne l'*ensemble des variables arithmétiques* de t. Un I-terme t est dit *fermé* si $var(t) = \emptyset$.

Nous étendons les notations $t[s]_\diamond$ et t^n introduites pour les ρ-termes aux I-termes. Ainsi, si t est un terme avec un trou ($t \in T_\diamond$), nous dénotons par $t[s]_\diamond$ le terme obtenu à partir de t en remplaçant l'unique occurrence du trou n'apparaissant pas dans un N-terme, par s. Formellement, la notation $t[s]_\diamond$ est définie inductivement :

- $\diamond[s]_\diamond = s$.
- $x[s]_\diamond = x$.
- $f(t_1,\ldots,t_n)[s]_\diamond = f(t_1,\ldots,t_{i-1},t_i[s]_\diamond,t_{i+1},\ldots,t_n)$ où $t_i \in T_\diamond$.

Si $t \in T_\diamond$ et $n \in \mathbb{N}$, t^n désigne le terme avec un trou $t^n = \underbrace{t[t[\ldots[\diamond]_\diamond\ldots]_\diamond]_\diamond}_{n}$. Il est également clair (par récurrence) que si $t \in T_\diamond$, alors $t^n \in T_\diamond$ pour tout $n \in \mathbb{N}$.

3.3.1 Sémantique des I-termes

Les I-termes sont utilisés pour décrire de manière finie un ensemble infini dont les éléments partagent une même logique de construction syntaxique. Ainsi, chaque terme fermé *instance* d'un I-terme t peut être obtenu en itérant un nombre donné de fois le contexte inductif de t et en remplaçant le trou par le terme de base. La sémantique des I-termes peut être décrite par un système de réécriture identique au système R_ρ introduit pour les ρ-termes mais étendu aux

I-termes. Nous notons ce système R_I qui peut être formalisé par :

$$R_I = \left\{ \begin{array}{lll} t^0.s & \rightarrow_I & s \qquad t \in T_\diamond, s \in T_I \\ t^{n+1}.s & \rightarrow_I & t[t^n.s]_\diamond \quad n \in \mathbb{N}, t \in T_\diamond, s \in T_I \end{array} \right.$$

Proposition 5. *Le système de réécriture R_I est convergent. De plus, il réduit avec les règles de l'arithmétique de Presburger, chaque I-terme sans variables arithmétiques en un terme standard.* ■

Preuve. Par induction sur la structure des I-termes. □

Dans la suite, $t\downarrow$ désignera la forme normale d'un I-terme t par rapport à $\rightarrow_I$ et $=_I$ désignera l'égalité modulo le dépliage, ainsi, $t =_I s$ si et seulement si $t\downarrow = s\downarrow$ pour tous I-termes t et s.

Les substitutions peuvent être étendues aux I-termes de la même façon que pour les ρ-termes. De même, les notions de substitution $\mathbb{N}$-clôturante et de $\mathbb{N}$-instance peuvent également être étendues d'une façon directe aux I-termes. L'ensemble des substitutions $\mathbb{N}$-clôturantes d'un (I-)terme t est noté $\Sigma_g(t)$ (comme c'est le cas pour les ρ-termes).

La définition suivante formalise la sémantique des ρ-termes :

Définition 37 (Sémantique des I-termes)**.** L'ensemble des termes standards représentés par un I-terme t est l'ensemble $\{t\sigma\downarrow | \sigma \in \Sigma_g(t)\}$. ■

L'unification des I-termes est décidable et il existe un algorithme d'unification introduit dans [Com95].

3.4 R-termes

3.4.1 Syntaxe des R-termes

Les contextes inductifs des I-termes ne contiennent qu'une seule position itérative (un seul trou). Il est ainsi impossible par exemple de représenter l'ensemble infini $\{c, f(a,a), f(f(a,a), f(a,a)), \ldots\}$ avec un I-terme ($f(\diamond,\diamond)^N.a$ n'est pas un I-terme). Les R-termes [Sal92] développés en parallèle et indépendamment du développement des I-termes permettent de résoudre ce problème en autorisant plusieurs trous dans le contexte inductif. Ceci est la seule différence entre les R-termes et les I-termes.

Dans la terminologie des R-termes, les variables arithmétiques sont appelées N-variables. et la forme des exposants considérés est définie par l'ensemble E_N.

La définition suivante formalise la syntaxe des R-termes.

Définition 38 (R-termes)**.** L'ensemble T_R des *R-termes* et l'ensemble des termes à plusieurs trous $T_R^\diamond$ sont les plus petits ensembles vérifiant :

- $\diamond \in T_R^\diamond$ et $V_X \subset T_R^\diamond$.
- Si $t_1, \ldots, t_k \in T_R$, $f \in \mathcal{F}$ et $arité(f) = k$ alors $f(t_1, \ldots, t_k) \in T_R$.
- Si $f \in \mathcal{F}$, $arité(f) = k > 0$, $\{i_1, \ldots i_n\} \subseteq [1..k]$, $n > 0$, $t_{i_1}, \ldots, t_{i_n} \in T_R^\diamond$, $t_j \in T_R$ pour tout $j \in [1,k]$, $j \notin \{i_1, \ldots i_n\}$ alors $f(t_1, \ldots, t_k) \in T_R^\diamond$.
- Si $t \in T_R^\diamond$, $t \neq \diamond$, $s \in T_R$ et $N \in E_N$ alors $t^N.s \in T_R$. Comme pour les ρ-termes et les I-termes, t est appelé contexte inductif, s terme de base et $t^N.s$ terme de base. $T_R^\diamond$ définit les *contextes itérés* des R-termes.

■

Exemple 9. $f(\diamond,\diamond)^{2.N}.a$ (plusieurs trous), $f(x,\diamond)^{2.N+1}.g(\diamond)^M.b$ (le terme de base est un R-terme), $f(\diamond, f(\diamond,c)^M.b, \diamond)^N.a)$ (le contexte inductif contient un R-terme) sont des R-termes. ■

La notation $t[s]_\diamond$ introduite pour les I-termes à la section 3.3 pour décrire l'opération de remplacement (de l'unique occurrence) d'un trou par un terme peut facilement être étendue aux R-termes où un contexte peut contenir *plusieurs* trous. Cette opération peut être formalisée avec le système de réécriture suivant :

$$\begin{array}{rcl} \diamond[s]_\diamond & \to & s \\ x[s]_\diamond & \to & x \\ f(t_1 \dots t_n)[s]_\diamond & \to & f(t_1[s]_\diamond, \dots, t_n[s]_\diamond) \end{array}$$

3.4.2 Sémantique des R-termes

Les R-termes ont un pouvoir d'expression strictement supérieur à celui des I-termes : dans [AHL93], les auteurs démontrent que l'ensemble des I-termes est strictement inclus dans celui des R-termes. Par exemple, la suite infinie $\{a, f(a), f(f(a)), \dots\}$ peut être représentée par $f(\diamond)^n.a$ qui est aussi bien un I-terme qu'un R-terme. En revanche, la suite de termes $\{a, f(a,a), f(f(a,a), f(a,a)), \dots\}$ peut être spécifiée par le R-terme $f(\diamond,\diamond)^N.a$ mais ne peut pas être représentée par un I-terme (comme expliqué plus haut).

Comme pour les I-termes, un N-terme $t^e.s$ définit un contexte itératif t à *déplier* un certain nombre de fois. Le nombre de dépliages est défini par l'expression e. Le système de réécriture R_I qui définit la sémantique des I-termes peut être étendu d'une façon directe aux R-termes. Nous notons ce système R_R qui peut être formalisé par :

$$R_R = \left\{ \begin{array}{lll} t^0.s & \to_R \; s & t \in T_R^\diamond, s \in T_R \\ t^{n+1}.s & \to_R \; t[t^n.s]_\diamond & n \in \mathbb{N}, t \in T_R^\diamond, s \in T_R \end{array} \right.$$

Proposition 6. *Le système de réécriture R_R est convergent. De plus, il réduit avec les règles de l'arithmétique de Presburger, chaque I-terme sans variables arithmétiques en un terme standard.* ■

Preuve. Par induction sur la structure des R-termes. □

Comme le système R_R est convergent, nous pouvons définir pour chaque R-terme t une forme normale unique que nous notons $t\downarrow_R$.

Les substitutions peuvent être étendues aux R-termes de la même façon que pour les (ρ, I)-termes. De même, les notions de substitution $\mathbb{N}$-clôturante et de $\mathbb{N}$-instance peuvent également être étendues d'une façon directe aux R-termes. L'ensemble des substitutions $\mathbb{N}$-clôturantes d'un (R-)terme t est noté $\Sigma_g(t)$ (comme c'est le cas pour les (ρ, I)-termes).

La définition suivante formalise la sémantique des R-termes :

Définition 39 (Sémantique des R-termes)**.** L'ensemble des termes standards représentés par un R-terme t est l'ensemble $\{t\sigma\downarrow \,|\sigma \in \Sigma_g(t)\}$. ■

Exemple 10. $f(\diamond, g(\diamond))^N.a$ est un R-terme qui représente l'ensemble infini $\{a, f(a, g(a)), f(f(a, g(a)), g(f(a, g(a)))), \dots\}$. ■

L'unification des R-termes est décidable et il existe un algorithme d'unification introduit dans [Sal92].

3.5 Grammaires primales

Nous présentons dans cette section le formalisme des *grammaires primales*. La différence majeure par rapport aux R-termes (et I-termes) est que ce formalisme permet de définir des

contextes inductifs paramétrés par des compteurs, ainsi, le contexte inductif peut être déplié à chaque fois d'une manière différente en fonction de la valeur de l'exposant. Cela permet de définir des séquences de la forme $f(g^n(a), f(g^{n-1}(a), \ldots, f(g(a)^0, a)\ldots))$ par exemple (où le contexte inductif varie en fonction du rang d'itération). Dans [Her94], il est démontré que ce formalisme est strictement plus expressif que celui des R-termes, dans le sens où tout ce qu'il est possible de représenter avec des R-termes peut également être représenté avec des grammaires primales (la réciproque n'est pas vraie). Cette section est adaptée de [HG97]. Pour plus de détails, le lecteur pourra consulter [Her94, HG97]. [ACP08] présente une extension plus flexible des grammaires primales qui a le même pouvoir d'expression.

3.5.1 Syntaxe

Nous considérons un nouvel ensemble de symboles de fonction, appelées *symboles définis*, et noté $\mathcal{D}$. Nous considérons également un ensemble $\mathcal{N}$ contenant la constante 0 et le symbole s (s comme successeur). Les symboles de $\mathcal{F}$ sont appelés des *symboles constructeurs*.

Les symboles définis (appartenant à $\mathcal{D}$) sont surmontés d'un accent circonflexe pour les distinguer des autres symboles. Par exemple un symbole défini sera noté $\hat{f}$. Les symboles définis sont implicitement typés et acceptent deux types de sortes. Les arguments des deux sortes sont séparés par le symbole ';'. Les arguments qui apparaissent avant le ';' sont appelés des *compteurs* et seront notés en gras pour les distinguer des autres termes (comme dans [ACP08]). Le premier compteur est appelé un *compteur principal*, les autres compteurs sont appelés des *compteurs secondaires*. Pour chaque symbole défini $\hat{f}$, $car(\hat{f})$ désigne le nombre d'arguments compteurs de $\hat{f}$.

Dans [HG97], les compteurs sont des expressions construites sur une algèbre dite l'*algèbre des expressions compteurs*. De manière équivalente, et par souci d'homogénéité, nous définissons les compteurs comme des expressions de E_N puisque dans les deux cas les expressions considérées sont en fait des combinaisons linéaires d'entiers et de variables arithmétiques pondérées par des entiers.

Définition 40 (Terme primal)**.** L'ensemble des termes primaux $\mathcal{TP}$ est le plus petit ensemble tel que :

- $V_X \subset \mathcal{TP}$.
- Si $t_1, \ldots t_n \in \mathcal{TP}$ et $f \in \mathcal{F}$ et $arité(f) = n$ alors $f(t_1, \ldots t_n) \in \mathcal{TP}$.
- Si $\boldsymbol{c_1}, \ldots, \boldsymbol{c_k} \in E_N$, $t_1, \ldots t_n \in \mathcal{TP}$, $\hat{f} \in \mathcal{D}$, $car(\hat{f}) = k$ et $arité(\hat{f}) = n + k$ alors $t = \hat{f}(\boldsymbol{c_1}, \ldots \boldsymbol{c_k}; t_1, \ldots t_n) \in \mathcal{TP}$. Les variables arithmétiques qui apparaissent dans $\boldsymbol{c_1}, \ldots \boldsymbol{c_k}$ sont appelées des *variables arithmétiques* (ou *variables compteurs*) de t. Comme pour les (ρ, I, R)-termes, $NVar(t)$ désigne l'ensemble des variables arithmétiques de t. De plus, un terme de ce type $(\hat{f}(\boldsymbol{c_1}, \ldots \boldsymbol{c_k}; t_1, \ldots t_n))$ est appelé un *N-terme* (comme pour les (ρ, I, R)-termes).

■

En particulier, tous les termes standards sont des termes primaux. Les termes construits à partir d'un symbole défini, d'un ensemble d'expressions compteurs et d'un ensemble de termes primaux sont également des termes primaux.

Exemple 11. Soient $\mathcal{F} = \{f, g, a, b\}$, $N, M \in V_N$, $\mathcal{D} = \{\hat{h}, \hat{j}\}$. Les termes suivants sont des termes primaux :

x , $f(a, x)$, $\hat{h}(\boldsymbol{c_1}; a, b)$, $\hat{h}(\boldsymbol{2}, \boldsymbol{c_1} + \boldsymbol{3}; g(x, b), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b))$, $g((\hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b), x)$. ■

La notion de position introduite dans la définition 7 page 10 est utilisée de la même façon pour les termes primaux.

Définition 41. Un *redex* r de t est un sous-terme r de t tel que r est un N-terme. L'ensemble des redex d'un terme t est noté $Rdx(t)$.

L'*ensemble des positions redex* d'un terme t noté $\mathcal{D}pos(t)$ est l'ensemble de toutes les positions p de t telles que $t|_p$ est un redex. ■

Exemple 12. Soit le terme primal $t = g(\hat{h}(\mathbf{2}, \boldsymbol{c_1} + \mathbf{3}; \hat{j}(\mathbf{0}, \mathbf{2}; a), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b)), x)$. Les redex de t sont :
- $\hat{h}(\mathbf{2}, \boldsymbol{c_1} + \mathbf{3}; \hat{j}(\mathbf{0}, \mathbf{2}; a), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b))$ à la position 1,
- $\hat{j}(\mathbf{0}, \mathbf{2}; a)$ à la position 1.3,
- $\hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b)$ à la position 1.4.

Nous avons donc $Rdx(t) = \{\hat{h}(\mathbf{2}, \boldsymbol{c_1} + \mathbf{3}; \hat{j}(\mathbf{0}, \mathbf{2}; a), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b)), \hat{j}(\mathbf{0}, \mathbf{2}; a), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b)\}$ et $\mathcal{D}pos(t) = \{1, 1.3, 1.4\}$. ■

Définition 42. Un terme primal t est dit *régulier* (ou *non imbriqué*) si l'ensemble $\mathcal{D}pos(t)$ est parallèle, i.e. il n'existe pas de positions $p, q \in \mathcal{D}pos(t)$ telles que p est un préfixe de q ou q est un préfixe de p. . t est *imbriqué* s'il existe deux positions redex p, q de t telles que p est un préfixe de q. ■

Exemple 13. $\hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b)$ est un terme primal régulier, en revanche $t = g(\hat{h}(\mathbf{2}, \boldsymbol{c_1} + \mathbf{3}; \hat{j}(\mathbf{0}, \mathbf{2}; a), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b)), x)$ est imbriqué car il contient un redex $\hat{h}(\mathbf{2}, \boldsymbol{c_1} + \mathbf{3}; \hat{j}(\mathbf{0}, \mathbf{2}; a), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b))$ à la position 1 qui est un préfixe de 1.3 correspondant à la position du redex $\hat{j}(\mathbf{0}, \mathbf{2}; a)$.

Intuitivement, les N-termes correspondent aux parties gauches d'un système de réécriture (qui sera introduit ci-dessous). Un redex est donc un sous-terme qui sera réécrit et remplacé par un autre terme. Pour définir ce système de réécriture, nous avons besoin d'introduire la notion d'*approximation d'un terme primal* qui utilise elle même la notion de *sous-vecteur*. Les deux définitions suivantes introduisent ces deux concepts. ■

Définition 43 (Sous-vecteur)**.** Soient $u = (u_1, \ldots, u_n)$ et $v = (v_1, \ldots, v_m)$ deux vecteurs. u est un *sous-vecteur* de v et nous notons $u \Subset v$ si et seulement si :
- $n \leq m$
- Il existe une fonction croissante $\eta : [1..n] \to [1..m]$ telle que $\forall i \in [1..n], u_i = v_{\eta(i)}$.

■

Exemple 14. $(2, 5, a) \Subset (0, 1, x, 2, y, 5, z, a, b)$. En revanche $(2, 5, a) \not\Subset (0, 1, x, 2, y, 5)$ ($a \notin (0, 1, x, 2, y, 5)$) et $(1, 2) \not\Subset (2, 3, 1)$ (1 est avant 2 dans $(1, 2)$ mais 2 est avant 1 dans $(2, 3, 1)$). ■

Dans la suite de cette section, $\succ$ désigne une relation d'ordre (non nécessairement totale) stricte (ou relation de précédence) sur les symboles de $\mathcal{D}$.

Définition 44 (Approximation d'un terme primal)**.** L'*approximation* d'un terme primal $t = \hat{f}(\boldsymbol{c_1}, \ldots \boldsymbol{c_k}; t_1, \ldots t_n)$ par rapport à la relation de précédence $\succ$ notée $\mathcal{A}px(t)$ est un ensemble de termes primaux défini par :

$$\mathcal{A}px(\hat{f}(\boldsymbol{c_1}, \ldots, \boldsymbol{c_m}; t_1 \ldots t_n)) = \{\hat{g}(\boldsymbol{z_1}, \ldots, \boldsymbol{z_k}; s_1, \ldots s_l) | \hat{f} \succ \hat{g}, \overrightarrow{z} \Subset \overrightarrow{c}, \overrightarrow{s} \Subset \overrightarrow{t}\}$$

où $\overrightarrow{z} = (\boldsymbol{z_1}, \ldots, \boldsymbol{z_k})$, $\overrightarrow{c} = (\boldsymbol{c_1}, \ldots, \boldsymbol{c_m})$, $\overrightarrow{s} = (s_1, \ldots s_l)$ et $\overrightarrow{t} = (t_1 \ldots t_n)$. ■

Exemple 15 ([HG97])**.** Soient $\hat{f}, \hat{g}, \hat{h}, \hat{k}$ tels que $\hat{f} \succ \hat{g} \succ \hat{h} \succ \hat{k}$ et $arité(\hat{f}) = 4$, $arité(\hat{g}) = 1$ $arité(\hat{h}) = 2$, $arité(\hat{k}) = 4$.

$$\mathcal{A}px(\hat{f}(\boldsymbol{c_1}, \boldsymbol{c_2}; x, y)) = \{\hat{g}(\boldsymbol{c_1}), \hat{g}(\boldsymbol{c_2}), \hat{h}(\boldsymbol{c_1}; x), \hat{h}(\boldsymbol{c_2}; x), \hat{h}(\boldsymbol{c_1}; y), \hat{h}(\boldsymbol{c_2}; y), \hat{k}(\boldsymbol{c_1}, \boldsymbol{c_2}; x, y)\}$$

■

Définition 45 (Enveloppe d'un terme primal)**.** L'*enveloppe* d'un terme primal t est le terme obtenu à partir de t en remplaçant toute occurrence d'un redex par un nouveau symbole spécial $\bullet$ appelé *trou noir*. En reprenant les notations du chapitre 2 l'enveloppe d'un terme t est égal à $t[\bullet]_{\mathcal{D}pos}$. ■

Une enveloppe ne contient par définition que des symboles constructeurs, des variables standards ou des trous (noirs).

Exemple 16. L'enveloppe du terme primal $t = g(\hat{j}(\mathbf{0}, \mathbf{2}; a), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b)), x)$ est $g(\bullet, \bullet, x)$.
Celle de $r = g(\hat{h}(\mathbf{2}, \boldsymbol{c_1} + \mathbf{3}; \hat{j}(\mathbf{0}, \mathbf{2}; a), \hat{j}(\boldsymbol{c_1}, \boldsymbol{c_2}; b)), x)$ est $g(\bullet)$. ■

3.5.2 Sémantique

Les termes primaux permettent de définir comme pour le cas des autres schématisations récurrentes, des contextes itératifs. La différence en revanche est que le contexte itératif peut être paramétré par un ou plusieurs compteurs. La sémantique des grammaires primales est calculée par un système de réécriture. En réalité, il n'existe pas un seul système de réécriture, puisque contrairement aux formalismes précédents, dans le cas des grammaires primales, le système de réécriture dépend du terme à représenter. La définition suivante formalise ce(s) système(s) de réécriture :

Définition 46 (Système de réécriture de Presburger)**.** Un système de réécriture R_P est dit de *Presburger* pour les symboles définis $\mathcal{D}$ si et seulement si R_P contient pour chaque symbole défini $\hat{f} \in \mathcal{D}$ exactement deux règles de réécriture :

– Une règle appelée *règle basique* de la forme :

$$\hat{f}(\mathbf{0}, \boldsymbol{c_1}, \dots \boldsymbol{c_k}; x_1 \dots x_m) \to r_1^{\hat{f}}$$

– Une règle appelée *règle inductive* qui peut avoir l'une des deux formes suivantes :

$$\begin{array}{lcll} \hat{f}(\boldsymbol{n}+\mathbf{1}, \boldsymbol{c_1}, \dots \boldsymbol{c_k}; x_1 \dots x_m) & \to & r_2^{\hat{f}}[\hat{f}(\boldsymbol{n}, \boldsymbol{c_1}, \dots \boldsymbol{c_k}; x_1 \dots x_m)]_{\mathcal{P}} & (1) \\ \hat{f}(\boldsymbol{n}+\mathbf{1}, \boldsymbol{c_1}, \dots \boldsymbol{c_k}; x_1 \dots x_m) & \to & r_2^{\hat{f}}[\hat{f}(\boldsymbol{n}, \boldsymbol{c_1}, \dots, \boldsymbol{c_{i-1}}, \boldsymbol{c_i}+\mathbf{1}, \boldsymbol{c_{i+1}} \dots \boldsymbol{c_k}; x_1 \dots x_m)]_{\mathcal{P}} & (2) \end{array}$$

où :
- $r_1^{\hat{f}}$ et $r_2^{\hat{f}}$ sont deux termes primaux associés par R_P au symbole $\hat{f}$.
- $\boldsymbol{c_1}, \dots, \boldsymbol{c_n}$ sont des variables arithmétiques.
- $\mathcal{P}$ est un ensemble fini de positions parallèles de $Pos(r_2^{\hat{f}})$ tel que $\Lambda \notin \mathcal{P}$.
- La partie droite de chaque règle inductive est un terme primal *régulier*.
- Tous les redex de $r_1^{\hat{f}}$ et $r_2^{\hat{f}}$ appartiennent à $\mathcal{A}px(\hat{f}(\boldsymbol{n}, \boldsymbol{c_1}, \dots \boldsymbol{c_k}; x_1 \dots x_m))$: $Rdx(r_1^{\hat{f}}) \cup Rdx(r_2^{\hat{f}}) \subset \mathcal{A}px(\hat{f}(\boldsymbol{n}, \boldsymbol{c_1}, \dots \boldsymbol{c_k}; x_1 \dots x_m))$.
- Le symbole de tête de $r_2^{\hat{f}}$ est un *constructeur*.
- Chaque variable x_i appartient à l'enveloppe de $r_1^{\hat{f}}$ ou celle de $r_2^{\hat{f}}$.

Ce système de Presburger est dit *fondé* sur l'ensemble des symboles constructeurs $\mathcal{F}$. ■

Proposition 7 ([HG97])**.** *Le système de réécriture R_P est convergent et permet de calculer pour chaque terme primal t une forme normale unique notée $t \downarrow_{R_P}$.* ■

Les substitutions peuvent être étendues aux termes primaux de la même façon que pour les (ρ, I, R)-termes. De même, la notion de substitution $\mathbb{N}$-clôturante peut également être étendue d'une façon directe aux termes primaux. L'ensemble des substitutions $\mathbb{N}$-clôturantes d'un terme t est noté $\Sigma_g(t)$ (comme c'est le cas pour les (ρ, I, R)-termes).

La définition suivante formalise la sémantique des grammaires primales :

Définition 47 (Sémantique des grammaires primales [HG97])**.** Une *grammaire primale* est un quadruplet $G = (\mathcal{K}, \mathcal{D}, \mathcal{R}_P, t)$, où $\mathcal{K}$ est un ensemble de symboles constructeurs, $\mathcal{D}$ un ensemble de symboles définis, R_P un système de réécriture de Presburger défini sur $\mathcal{D}$ et fondé sur $\mathcal{K}$ et t est un terme primal.

Le langage généré par une grammaire primale $G = (\mathcal{K}, \mathcal{D}, \mathcal{R}_P, t)$ noté $L(G)$ est l'ensemble de termes standards $L(G) = \{t\sigma \downarrow_{\mathcal{R}_P} | \sigma \in \Sigma_g(t)\}$. ■

Exemple 17. L'exemple suivant donne une grammaire primale formalisant une séquence qui ne peut pas être décrite par les autres formalismes (ρ, I, R)-termes. Soit la grammaire primale $G = (\mathcal{K}, \mathcal{D}, R_P, t)$ avec $\mathcal{K} = \{f, g, a, b\}, \mathcal{D} = \{\hat{f}, \hat{g}\}, t = \hat{f}(n, 0)$ et

$$R_P \begin{cases} \hat{f}(n+1, m) & \rightarrow & f(\hat{g}(n), \hat{g}(m), \hat{f}(n, m+1)) \\ \hat{f}(0, m) & \rightarrow & a \\ \hat{g}(n+1) & \rightarrow & g(\hat{g}(n)) \\ \hat{g}(0) & \rightarrow & b \end{cases}$$

Cette grammaire primale schématise le langage L :

$$\left\{a, f\left(b, b, a\right), \ldots, f\left(g^{n-1}\left(b\right), b, f\left(g^{n-2}\left(b\right), g\left(b\right), f\left(\ldots, f\left(g^{n-1-i}\left(b\right), g^{i}\left(b\right), f\left(\ldots, f\left(b, g^{n-1}\left(b\right), a\right)\ldots\right)\right)\ldots\right)\right)\right), \ldots\right\}$$

Nous montrons que L ne peut pas être représenté par un R-terme. Soit $n \in \mathbb{N}$. Le terme $\hat{f}(n, 0) \downarrow_{R_P}$ contient n occurrences du symbole f aux positions $\Lambda, 3, 3.3, 3.3.3, \ldots, \underbrace{3.\ldots.3}_{n-1}$. De plus le sous-terme à la position $\underbrace{3.\ldots.3}_{n}$ est a. Donc s'il existait un R-terme t qui représente L, alors t serait de la forme $t = f\left(t_1, t_2, \diamond\right)^N.a$. D'un autre côté, $\hat{f}(n, 0) \downarrow_{R_P}$ contient $g^{n-1}\left(b\right)$ et b comme sous-termes aux positions 1 et 2 respectivement. Ainsi, $t_1 = g^{n-1}\left(b\right)$ et $t_2 = b$ et donc $t = f\left(g^{N-1}\left(b\right), b, \diamond\right)^N.a$. Or t représente l'ensemble $L' = \{a, f(b, b, a), f(g(b), b, f(g(b), b, a)), \ldots\}$ qui est différent de L puisque par exemple le terme $r = f\left(g\left(b\right), f\left(b, g\left(b\right), a\right)\right) \in L$ mais $r \notin L'$. ■

Finalement, l'unification des grammaires primales est décidable et il existe un algorithme d'unification introduit dans [HG97].

3.6 Choix du formalisme

Une étude comparée du pouvoir d'expression de ces différents formalismes est disponible dans [Her94, AHL93]. Ainsi, ces langages de schématisation constituent une hiérarchie en terme de pouvoir d'expression allant du moins expressif (les ρ-termes) au plus expressif (les grammaires primales). Malheureusement, comme nous pourrions nous y attendre, la complexité des opérations liées à ces langages et particulièrement l'unification est liée au pouvoir d'expression. Or même si pour chacun de ces langages le problème d'unification est décidable et il existe un algorithme d'unification qui termine toujours, pour pouvoir espérer intégrer la schématisation de termes en démonstration automatique, il est primordial que le coût de ses opérations soit raisonnable (surtout l'unification). Nous pensons que le formalisme des I-termes offre un excellent compromis entre pouvoir d'expression et complexité et de ce fait nous avons choisi d'étudier l'utilisation des I-termes en déduction automatique. En particulier, l'unification, bien que beaucoup plus complexe que pour les termes standards, peut être implémentée assez facilement, ce qui n'est pas le cas pour les autres formalismes qui font appel à des mécanismes plus sophistiqués pour détecter les boucles et garantir la terminaison lors du parcours des positions cycliques. L'extension des résultats et des systèmes présentés dans cette thèse à d'autres formalismes pourrait bien entendu constituer une suite naturelle à notre travail.

Nous nous focalisons donc dans cette section sur ce formalisme et présentons quelques résultats qui nous seront utiles dans la suite.

3.6.1 Forme simplifiée

Dans le formalisme des I-termes présenté à la section 3.3, l'exposant peut être la somme d'un entier et d'une combinaison linéaire de variables arithmétiques. Pour simplifier la présentation et sans perte de généralité, nous pouvons supposer que les termes considérés sont tous normalisés par rapport au système R_I (introduit à la sous-section 3.3.1) et que chaque exposant (forcément non entier par application de R_I) apparaissant dans un N-terme est une variable.

En effet il est possible de transformer n'importe quel I-terme t en un I-terme équivalent s tel que chaque exposant apparaissant dans un N-terme dans s est une variable. Nous définissons pour cela les opérateurs de transformation $\mathcal{T}$ et $\mathcal{T}_\diamond$ inductivement :

$$\mathcal{T} = \begin{cases} \mathcal{T}(x) = x & \text{si } x \in V_X \\ \mathcal{T}(f(t_1,\ldots,t_n)) = f(\mathcal{T}(t_1),\ldots,\mathcal{T}(t_n)) & \text{si } f \in \mathcal{F}, t_i \in T_I \\ \mathcal{T}(f(t_1,\ldots t'_i \ldots,t_n)) = f(\mathcal{T}(t_1),\ldots \mathcal{T}_\diamond(t'_i),\ldots,\mathcal{T}(t_n)) & \text{si } f \in \mathcal{F}, \forall j \neq i, t_j \in T_I, t'_i \in T_\diamond \\ \mathcal{T}_\diamond(\diamond) = \diamond & \\ \mathcal{T}(t^N.s) = \mathcal{T}_\diamond(t)^N.\mathcal{T}(s) & \text{si } t \in T_\diamond, s \in T_I, N \in V_N \\ \mathcal{T}(t^n.s) = (\mathcal{T}_\diamond(t)^n)[\mathcal{T}(s)]_\diamond & \text{si } n \in \mathbb{N} \\ \mathcal{T}(t^{k\times N}.s) = (\mathcal{T}_\diamond(t)^k)^N.\mathcal{T}(s) & \text{si } k \in \mathbb{N}, N \in V_N \\ \mathcal{T}(t^{k_1N_1+\ldots+k_nN_n+m}.s) = (\mathcal{T}_\diamond(t)^m)[\mathcal{T}(t^{k_1N_1}.(t^{k_2N_2+\ldots+k_nN_n}.s))]_\diamond & \text{si } t \in T_\diamond, s \in T_I, N_i \in V_N, k_i \in \mathbb{N} \end{cases}$$

Proposition 8. *Si $t \in T_I$ et $t' \in T_\diamond$ alors $\mathcal{T}(t) \in T_I$ et $\mathcal{T}(t') \in T_\diamond$.* ■

Preuve. La preuve est par induction sur la structure des I-termes. □

Proposition 9. *La transformation $\mathcal{T}$ est correcte (i.e. préserve la sémantique des I-termes) et complète (le résultat de l'application de $\mathcal{T}$ n'engendre que des I-termes avec variables comme seul exposant en supposant que les exposants constants sont éliminés par l'application des règle de R_I).* ■

Preuve. La preuve est par induction sur la structure des I-termes. □

L'exemple suivant illustre la pertinence de notre proposition : Le terme $f(\diamond)^{2.N+1}.a$ est équivalent par dépliage au terme $f(f(f(\diamond))^N.a)$.

3.6.2 I-termes et preuves par saturation.

Un de nos objectifs dans ce travail est d'étudier comment intégrer les I-termes dans les procédures existantes de preuve par saturation (les procédures les plus efficaces). Dans ce sens, il est possible d'étendre les notions de bases de la logique du premier ordre introduites au chapitre 2 pour intégrer les I-termes.

Définition 48. Un *I-atome* est de la forme $P(t_1,\ldots,t_n)$ où $t_1,\ldots,t_n \in T_I$ et P est un prédicat de $\mathcal{P}$ (en particulier si P correspond à $\approx$ alors nous avons $t_1 \approx t_2$).

Un *I-littéral* est soit un I-atome soit la négation d'un I-atome.

Une *I-clause* est une disjonction de I-littéraux. En particulier, la clause vide (ne contenant aucun I-littéral) est notée □. Remarquons qu'une I-clause ne contenant aucun N-terme est en fait une clause (au sens standard du terme comme introduit au chapitre 2). ■

Les notions d'instance et de $\mathbb{N}$-instance peuvent évidemment être étendues aux I-clauses. Ainsi, toutes les $\mathbb{N}$-instances d'une I-clause sont des clauses (standards). Les interprétations sont également définies comme dans le cas usuel. On dira qu'une interprétation I *valide* un ensemble d'I-clauses S et on écrira $I \models S$ si I valide *toutes* les $\mathbb{N}$-instances des I-clauses de S (au sens usuel du terme).

D'autres extensions seront présentées aux chapitres 5 et 6. Par ailleurs, il est indéniable que l'unification est une opération fondamentale pour les calculs de saturation. Ainsi, toute tentative d'extension de ces calculs a nécessairement besoin de pouvoir unifier (d'une manière aussi efficace que possible) les I-termes.

3.6.3 Unification

Nous présentons dans la suite un algorithme d'unification des I-termes ([Com95]). L'algorithme est décrit ci-dessous sous la forme d'un ensemble de règles en reprenant le formalisme et les notations de [Com95] :

Triviale	$s = s$	$\rightarrow$	$\top$
Décomposition	$f(\overrightarrow{s}) = f(\overrightarrow{t})$	$\rightarrow$	$\overrightarrow{s} = \overrightarrow{t}$
Contradiction	$f(\overrightarrow{s}) = g(\overrightarrow{t})$	$\rightarrow$	$\bot$
		si	$f \neq g$
Élimination	$x = s \wedge P$	$\rightarrow$	$x = s \wedge P\{x \rightarrow s\}$
de variables		si	$x \notin vars(s), x \in vars(P),$
			$(s \in V_X \Rightarrow s \in vars(P))$
Test d'occurrence	$x = s$	$\rightarrow$	$\bot$
		si	$x \not\equiv s, x \in vars(s)$

Cet ensemble de règles est appelé R_1. Il correspond aux règles usuelles de l'unification standard. Ces règles ne peuvent pas s'appliquer si l'un des deux termes à unifier s, t est un N-terme. Les règles suivantes sont introduites pour traiter ce cas (t ou s est un N-terme).

La première règle permet de ramener une (partie de l')unification impliquant un N-terme en un ensemble d'unifications gérées par R_1, et ce par dépliage de t. Par exemple pour unifier $s = f(a)$ et $t = f(\diamond)^N.a$, R_1 ne peut pas être utilisé puisque t est un N-terme, cependant, le dépliage de t une fois donne ou bien $(N = 1 \wedge f(a) = f(a))$ ou bien $\exists M.N = M + 1 \wedge f(a) = f(f(\diamond)^M.a)$. Dans les deux cas, la règle de Décomposition du système R_1 peut s'appliquer (en simplifiant le problème initial).

Dépliage 1

$$s = t[\diamond]_p^N.u \quad \rightarrow \quad (N = 1 \wedge s = t[u]_p) \vee (\exists M.N = M + 1 \wedge s = t[t^M.u]_p)$$

Si :
- il n'y a pas de préfixe q de p tel que s_q est un N-terme.
- R_1 ne s'applique pas.

R_2 désigne l'ensemble de règles obtenu en ajoutant la règle Dépliage 1 à l'ensemble R_1.

Exemple 18. Soit par exemple les deux termes $s = f(\diamond)^N.g(a)$ et $t = f(g(\diamond))^M.a$ à unifier. Par définition, la règle Dépliage 1 ne peut pas s'appliquer ($s|_\Lambda$ est un N-terme et Λ est un préfixe de 1.1 qui est la position du trou dans t). ■

Nous avons donc besoin d'autres règles (de dépliage et décomposition) pour gérer ce type de cas. La première règle (Dépliage 2) permet de ramener les cas $s[t_1[\diamond]_{q_1}^{N_1}.u_1]_p = t_2[\diamond]_{q_2}^{N_2}.u_2$ où les positions des trous dans t_1 et t_2 n'ont pas les même longueur vers des problèmes où c'est le cas.

Exemple 19. Par exemple en considérant $s = f(g(f(\diamond)))^N.g(a)$ et $t = f(g(\diamond))^M.f(g(a))$ nous avons (en utilisant les notations de la règle Dépliage 2 ci-dessous) : $p = \Lambda, q_1 = 1.1.1, q_2 = 1.1, d = 1, \alpha_1 = 3, \alpha_2 = 2$. Dans ce cas, $3 = |q_1| \neq |q_2| = 2$. L'objectif est de ramener ce problème vers un problème similaire mais avec des positions des trous de longueurs égales. ■

Dépliage 2

$$
\begin{array}{rll}
s[t_1[\diamond]_{q_1}^{N_1}.u_1]_p = & t_2[\diamond]_{q_2}^{N_2}.u_2 & \rightarrow \\
\bigvee_{1 \leq r_1 < \alpha_2} & N_1 = r_1 \wedge s[t_1[\diamond]_{q_1}^{r_1}.u_1]_p = t_2[\diamond]_{q_2}^{N_2}.u_2 & \\
\bigvee_{1 \leq r_2 < \alpha_1} & N_2 = r_2 \wedge s[t_1[\diamond]_{q_1}^{N_1}.u_1]_p = t_2[\diamond]_{q_2}^{r_2}.u_2 & \\
\bigvee_{\substack{0 \leq r_1 < \alpha_2 \\ 0 \leq r_2 < \alpha_1}} & \exists M_1, M_2. N_1 = \alpha_2 \times M_1 + r_1 \wedge N_2 = \alpha_1 \times M_2 + r_2 & \\
& \wedge s[(t_1^{\alpha_2})^{M_1}.t_1^{r_1}.u_1]_p = ((t_2^{\alpha_1})^{M_2}.t_2^{r_2}.u_2) &
\end{array}
$$

Si $|q_1| \neq |q_2|$, R_2 ne peut pas s'appliquer, p est un préfixe de q_2, $d = pgcd(|q_1|, |q_2|)$, $\alpha_1 = \frac{|q_1|}{d}$, $\alpha_2 = \frac{|q_2|}{d}$, et $m = \alpha_1 \times \alpha_2 \times d$ (m est le *ppcm* de $|q_1|$ et $|q_2|$).

Nous remarquons que la position du trou dans $(t_1^{\alpha_2})^{M_1}.t_1^{r_1}.u_1]_p$ est de longueur égale à celle dans $(t_2^{\alpha_1})^{M_2}.t_2^{r_2}.u_2$ ($|q_1| \times \alpha_2 = \alpha_1 \times \alpha_2 \times d = |q_2| \times \alpha_1 = m$). La règle suivante Dépliage 3 permet de déplier des termes qui possèdent précisément cette propriété (i.e. la longueur de la position du trou dans des N-termes est la même des deux cotés de l'égalité) et sur lesquels la règle R_2 ne peut pas s'appliquer.

Dépliage 3

$$
\begin{array}{rll}
s[t_1[\diamond]_{q_1}^{N_1}.u_1]_p = t_2[\diamond]_{q_2}^{N_2}.u_2 & \rightarrow & (N_2 = 1 \wedge t_2[u_2]_{q_2} = s[t_1^{N_1}.u_1]_p) \\
& \vee & (N_1 = 1 \wedge s[t_1[u_1]_{q_1}]_p = t_2^{N_2}.u_2) \\
& \vee & (N_2 = 2 \wedge t_2[t_2[u_2]_{q_2}]_{q_2} = s[t_1^{N_1}.u_1]_p) \\
& \vee & (\exists M_1, M_2. N_1 = M_1 + 1 \wedge N_2 = M_2 + 2 \\
& & \quad \wedge s[t_1[t_1^{M_1}.u_1]_{q_1}]_p = t_2[t_2[t_2^{M_2}.u_2]_{q_2}]_{q_2})
\end{array}
$$

Si p est un préfixe de q_2 mais ou bien q_2 n'est pas un préfixe de $p.q_1$ ou bien $p.q_1$ n'est pas un préfixe de $q_2.q_2$.

Finalement, la règle Décomposition 2.

Décomposition 2

$$
\begin{array}{rll}
s[(v_1[w_1[\diamond]_p]_{q'})^{N_1}.u_1]_p & = (v_2[w_2[\diamond]_{q'}]_p)^{N_2}.u_2 & \rightarrow \\
& & s[a]_p = w_1[a]_p \wedge s[a]_p = v_2[a]_p \wedge v_1[a]_{q'} = w_2[a]_{q'} \\
& \wedge(& (N_1 = N_2 \wedge s[u_1]_p = u_2) \\
& \vee & (\exists M_1. N_1 = N_2.M_1 \wedge s[(v_1[w_1[\diamond]_p]_{q'})^{M_1}.u_1]_p = u_2) \\
& \vee & (\exists M_2. N_2 = N_1.M_2 \wedge w_1[u_1]_p = (v_2[w_2[\diamond]_{q'}]_p)^{M_2}.u_2) \quad)
\end{array}
$$

Théorème 5 ([Com95]). *Le système R_3 qui consiste à appliquer R_2 tant que c'est possible puis Dépliage 2, Dépliage 3 et Décomposition 2 sur les problèmes irréductibles par R_2 est correcte et termine sur n'importe quel problème d'unification (de I-termes). De plus, si une formule est irréductible par R_3 elle est nécessairement de la forme :*

$$\exists \overrightarrow{n}.\phi \wedge x_1 = t_1 \wedge x_n = t_n$$

où ϕ est une conjonction d'équations diophantiennes linéaires, $\overrightarrow{n}$ est un ensemble de variables arithmétiques et $x_1 \ldots, x_n$ sont des variables standards n'apparaissant pas dans les parties droites de la conjonction. ■

La solution d'un problème d'unification pour I-termes peut être obtenue après la résolution du système diophantien par un solveur de contraintes par exemple. Comme nous allons le voir au chapitre 7, il est possible d'éviter le recours à la résolution du système linéaire en procédant à des remplacements à la volée des variables concernées. Ceci signifie que la complexité de l'unification n'est pas liée à la résolution des contraintes arithmétiques.

La fonction mgu introduite au chapitre 7 qui retourne l'unificateur le plus général de deux termes (s'il existe) peut être étendue aux I-termes mais avec une différence importante : l'unicité de l'unificateur le plus général n'est plus assurée en cas de I-termes puisque pour deux I-termes t, s il peut exister plusieurs unificateur les plus généraux. Nous conservons la notation mgu, ainsi, si $t, s \in T_I$ alors $mgu(t, s)$ est l'ensemble de tous les unificateurs les plus généraux de t et s. Notons que pour tous t, s, $mgu(t, s)$ est un ensemble fini.

Deuxième partie

Prévention de la divergence

Chapitre 4

Analyse des espaces de recherche

4.1 Motivations

Même les stratégies les plus efficaces ne sont pas toujours satisfaisantes quand il s'agit de réaliser des tâches déductives. En effet, d'une part elles pèchent par leur *uniformité* dans le sens où elles traitent tous les problèmes de la même façon indépendamment de leurs spécificités[1] et d'autre part, elles sont très souvent incapables de contrôler l'explosion combinatoire due à des applications « aveugles et uniformes » des règles d'inférence. Cela est évidemment inévitable puisque le problème de la satisfaisabilité n'est pas décidable (en logique du premier ordre) et donc si un ensemble est satisfaisable, un calcul par saturation peut diverger (sur cet ensemble). Il est vrai que l'utilisation de stratégies bien choisies permet parfois de garantir la terminaison pour certaines classes particulières de formules (classes monadiques, Ackermann, E+ etc.) [FLTZ93]. Les règles d'élimination de la redondance telles que la subsomption ou le critère plus général proposé dans [BG94] ont précisément pour objet de limiter le nombre de clauses engendrées en supprimant toutes les clauses inutiles. Depuis les débuts historiques de la déduction automatique, le traitement des ensembles de très grandes tailles qui contiennent de nombreuses formules ou clauses générées par le calcul et *inutiles* pour la complétude de celui-ci a été reconnu comme étant d'une importance capitale. Ce problème est intimement lié à celui de l'analyse et de la structuration des espaces de recherche. À ce sujet, le lecteur pourra consulter par exemple [Wos88].

La structuration de l'espace de recherche engendré et par conséquent la structuration des preuves est une tâche que les stratégies actuelles sont incapables de réaliser. Doter alors les démonstrateurs de techniques qui permettent d'automatiser (ou même de semi-automatiser) l'*analyse* de l'espace de recherche est d'une importance capitale si nous voulons améliorer qualitativement ces systèmes. Le type d'analyse que nous considérons ici est principalement syntaxique. Elle consiste à étudier la *forme* des clauses générées dans le but de détecter si possible des régularités dans l'espace de recherche (pour aider à le structurer). Cette régularité peut se manifester par la présence de motifs répétés (plusieurs clauses ou termes ayant une structure similaire sont construits à partir de la répétition d'un motif comme c'est le cas par exemple pour les schémas de termes introduits au chapitre 3). Notre analyse doit également pouvoir détecter (dans certains cas) des cycles (c'est-à-dire des motifs qui se répètent) dans les inférences pour

[1] Cette assertion peut être tempérée par le fait que certains démonstrateurs intègrent une procédure capable d'analyser l'ensemble de clauses en entrée et de choisir la stratégie en fonction de la classe à laquelle il appartient (i.e. Horn avec égalité, Horn sans égalité, non Horn ...). Cela pourrait être vu comme une forme (au moins embryonnaire) de stratégie « non uniforme ».

élaguer les branches divergentes qu'ils induisent. Enfin, elle doit permettre de suggérer (générer) des lemmes nécessaires pour la terminaison dans certains cas. Dans la deuxième partie de notre travail nous étudions ce problème et nous proposons une méthode répondant à certains de ces objectifs. Notre méthode est inspirée par la *méthode expérimentale* et *inductive* (s'apparentant au raisonnement inductif, i.e. procéder du particulier au général - à ne pas confondre avec l'induction mathématique) utilisée par exemple pour *découvrir* des résultats en mathématiques ou en sciences naturelles. Il s'agit de conjecturer (induire) un résultat général en observant la réalisation de ce résultat pour des cas particuliers (qui doivent naturellement être *significatifs*). La valeur heuristique de ce type d'approche est indéniable comme mentionné dans [Caf09] qui rapporte par exemple que le mathématicien L. Euler affirmait que la plupart des propriétés des nombres qu'il avait établies avaient été découvertes à partir de l'observation d'exemples et de l'induction. Pour un traitement plus détaillé et plus philosophique de l'induction, le lecteur pourra consulter [Caf09] et pour un traitement plus historique et technique [Vic10]. Plus précisément, notre objectif est de procéder en étudiant les clauses produites (correspondant ici aux cas particuliers de l'induction) à une généralisation inductive qui fournit un résultat plus général dont certaines clauses de l'espace de recherche sont les cas particuliers. Le formalisme des I-termes décrit au chapitre précédent nous fournit un langage adéquat pour exprimer les propriétés induites, et permet une généralisation beaucoup plus fine (plus précise) que les termes du premier ordre.

Ce problème ressemble à ceux étudiés dans d'autres disciplines comme par exemple l'apprentissage automatique [Mit97], la programmation logique inductive [LD94], l'inférence grammaticale [AS83, Sak97], ... : il s'agit pour simplifier de produire une description synthétique ou une règle générale (ou plusieurs règles) qui explique (ou décrit) un ensemble d'éléments appelés *exemples positifs*, tout en excluant potentiellement d'autres éléments dits *négatifs*. À ce titre, le travail pionnier de Plotkin [Plo70, Plo71] qui a permis d'associer la généralisation inductive avec les systèmes de raisonnement doit être signalé.

Ce que nous recherchons, c'est de remplacer les descriptions *génétiques* ([Kle52]) des conséquences engendrées par le calcul et présentes dans l'espace de recherche par des descriptions *structurelles*. Les descriptions génétiques correspondent en général à la spécification du problème (en logique du premier ordre), alors que les descriptions structurelles permettent de décrire la *forme* (structure) des conséquences engendrées.

Cette description doit permettre de résoudre de manière effective certains problèmes sur les objets ainsi représentés : par exemple le problème de l'appartenance (tester si un objet donné appartient à la liste des conséquences engendrées), de l'intersection, ou encore de pouvoir effectuer des inférences sur les représentations ainsi construites. Il n'est évidemment pas possible en général de construire une description structurelle de la *totalité* de l'espace de recherche. En fait elle ne peut l'être en théorie que pour des ensembles de formules appartenant à des classes décidables. Cette limitation théorique ne doit pas empêcher d'étudier certains cas particuliers du problème, en particulier, la description de certains sous-ensembles de l'espace de recherche. Nous pouvons identifier trois types de problèmes où cette approche peut évidemment être utile (naturellement cette liste n'est pas exhaustive) :

– Le premier est immédiat : étant donné une liste de termes définis récursivement, nous voulons obtenir une formule qui caractérise ces objets en fonction de leur rang dans une liste. L'ordre d'apparition des éléments dans la liste (leur rang) peut naturellement être quelconque. Cependant, pour simplifier nous pouvons nous intéresser au cas où les éléments de la liste (les termes) sont triés selon l'ordre de génération des clauses les contenant dans une dérivation, i.e. une clause C générée avant une clause D aura un rang inférieur à celui de D. Ceci dit, il est évident que le rang n'est pas toujours pertinent et il pourrait être intéressant donc de considérer d'autres critères. Les deux autres usages sont fondés sur celui-ci.

- Les démonstrateurs de théorèmes manipulent et génèrent des objets structurés (les termes, les formules, les clauses, ...) et les combinent entre eux. Ces combinaisons engendrent des patrons (ou motifs) qui peuvent être représentés parfois par des objets utilisant des puissances entières comme par exemple la suite infinie $f(\ldots), f^3(\ldots), f^7(\ldots), \ldots$. Si nous arrivons à découvrir une loi (ou règle) capable de capturer intégralement et d'une manière structurelle l'ensemble des conséquences d'une formule, il est alors possible de décider si oui ou non la formule peut engendrer des contradictions, ce qui permet de statuer sur sa satisfaisabilité (la formule) et (dans certains cas) de spécifier des modèles (potentiellement infinis).
- Il est bien établi que l'introduction de lemmes peut fortement (même si parfois d'une façon non triviale) réduire la taille des preuves, ce qui permet de transformer certaines preuves *fastidieuses* en d'autres plus simples ou au moins plus faciles à lire et à comprendre (par un humain). La possibilité de décrire (abstraire, généraliser) à partir d'un ensemble (forcément fini) de cas particuliers peut s'avérer très utile, en particulier dans les preuves par induction. L'ajout de lemmes permet aussi dans certains cas de garantir la terminaison. Pour les preuves par induction, il est bien connu qu'il est parfois nécessaire d'effectuer l'induction sur une propriété plus générale que celle que l'on souhaite démontrer, appelée « lemme inductif »[2].

Ces idées sont appliquées dans le contexte de la démonstration automatique par saturation (particulièrement la résolution).

Notre approche bénéficie à la fois du pouvoir d'expression des langages de schématisation de termes et de la quantité énorme de travaux relatifs à la découverte de formes générales des suites d'entiers (voir par exemple l'encyclopédie en ligne des suites d'entiers (ou OEIS)[3]) et plus particulièrement des possibilités offertes dans ce sens par les logiciels de calcul symbolique comme Mathematica que nous utilisons pour calculer les séquences d'exposants. Ainsi, par exemple à partir de la suite $f(a)$, $f(a)^3$, $f(a)^5$, $f(a)^7$, ... nous nous intéressons à la suite entière des exposants $1, 3, 5, 7, \ldots$ OEIS ou Mathematica (dans notre cas) peuvent être utilisés pour trouver une forme générale de cette suite : $2N + 1$ où N est une variable dans $\mathbb{N}$ et la suite de termes peut être généralisée (généralisation inductive) par $f(\diamond)^{2N+1}.a$.

Deux points méritent néanmoins d'être soulignés :

- Le premier est que notre approche n'a certainement pas comme ambition de concurrencer le logiciel Mathematica ou d'autres outils excellents de calcul symbolique (en particulier l'encyclopédie OEIS), mais d'utiliser ce type d'outils pour *ajouter de nouvelles possibilités* (capacités) aux démonstrateurs automatiques de théorèmes.
- Le deuxième point est que même une complexité *élevée* de la procédure correspondante n'est pas gênante pour notre approche même si bien évidemment le temps de réponse doit rester raisonnable. En effet, notre objectif n'est pas de prouver mais de structurer les espaces de recherche, ce qui permet de fournir des informations utiles là où les démonstrateurs automatiques échouent ou ne donnent aucune information utile.

Une tâche importante dans cette partie a été d'implémenter un outil appelé DSSS (sigle de **D**escribing **S**tructurally **S**earch **S**paces ou plus simplement DS3) fondé sur ces idées pour illustrer l'intérêt de notre approche. En particulier, ce système a été capable de générer des lemmes utiles dans le cas de la technique de preuve par induction dite « induction sans induction » (voir la section 4.3).

Dans la suite de ce chapitre, nous décrivons DS3 qui est programmé en JAVA. Nous présentons également quelques applications de cet outil que nous espérons prometteuses, principalement dans la détection de satisfaisabilité, en particulier pour la construction de modèles et les preuves par

[2] Cette méthode s'apparente au paradoxe de l'inventeur [Pol73].

[3] http ://www.research.att.com/∼njas/sequences/

consistance.

4.2 Un nouvel outil pour analyser les espaces de recherche

Nous exploitons le pouvoir d'expression des I-termes dans le but de calculer automatiquement - quand c'est possible - une description structurelle des suites de termes standards qui sont générés pendant la procédure de recherche de preuves. Nous utilisons *uniquement* l'information contenue dans l'ensemble des clauses déduites (donc indépendance par rapport à la procédure de preuve utilisée). En fait, nous ne considérons dans notre implémentation que des N-termes ayant des contextes où le trou apparaît à la profondeur 1 (le trou est un sous-terme directe du symbole de tête du contexte inductif mais la profondeur du contexte n'est pas limitée). Ceci est une limitation évidente de notre approche qui ne permet pas par exemple de construire le N-terme $f\left(g\left(\diamond\right)\right)^2.a$ à partir du terme $f\left(g\left(f\left(g\left(a\right)\right)\right)\right)$. Mais cela rend d'un autre côté réaliste l'analyse de très grands ensembles de clauses, en particulier, l'application des deux règles **« Création de contexte »** et **« Normalisation »,** introduites ci-dessous à la page 43 et utilisées pour la construction des généralisations, n'a pas besoin de retour en arrière dans le traitement. Pour des contextes plus compliqués comme l'exemple cité ci-dessus, il est évident que l'usage d'heuristiques est important pour filtrer l'ensemble de clauses (termes) à considérer. Une conséquence immédiate de cette restriction est que tous les termes obtenus sont de la forme $f\left(t_1, \ldots t_n\right)^m.s$ où f est un symbole de fonction et $\forall i \in [1..n]$ t_i est soit le trou soit un I-terme (avec au maximum un trou).

Cette section propose une description de haut niveau de l'approche.

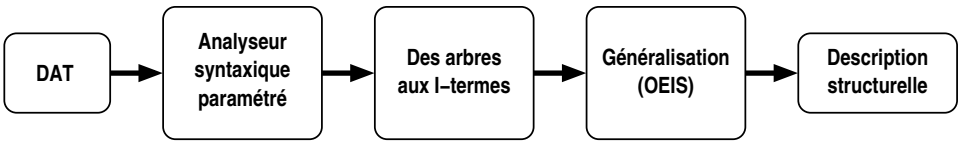

FIG. 4.1 – DS3 : architecture générale

4.2.1 L'algorithme

L'algorithme que nous proposons fonctionne comme illustré par la figure 4.1 en trois passes :

1. Une première passe d'**analyse syntaxique des clauses générées** par l'exécution d'un démonstrateur automatique sur un problème donné associée à une opération de simplification des clauses (ou compactage) qui vise à normaliser les clauses afin de pouvoir les comparer plus facilement et de mettre en évidence les structures communes.
2. Une deuxième passe de **classification des clauses simplifiées** produite lors de la première phase et qui produit des *classes* de clauses partageant une même structure.
3. Une troisième passe dont l'objectif est de proposer une **généralisation des classes produites** par la deuxième passe en utilisant un outil de calcul symbolique. La deuxième passe fournit la structure des termes avec exposants entiers, la troisième nous donne la valeur des exposants.

Nous décrivons à présent en détail chacune de ces passes.

Analyse syntaxique des clauses et construction des I-termes. Notre algorithme reçoit en entrée des clauses. Dans ce travail nous avons utilisé le démonstrateur automatique E-prover [Scha, Sch04] développé par S. Schulz. Notre analyseur syntaxique est donc lié au format des sorties produites par E-prover. La séparation entre la partie analyse syntaxique et

le reste des traitements, permet d'utiliser les mêmes idées avec d'autres démonstrateurs par saturation comme par exemple Vampire [RV01a], OTTER [McC95] ou son successeur prover9 (http ://www.cs.unm.edu/∼mccune/mace4/manual/June-2006C/) moyennant une adaptation de la partie analyse syntaxique uniquement. En fait, notre système permet d'analyser n'importe quel ensemble de termes ou formules dans la syntaxe du E-prover (évidemment le traitement d'autres types de formats pourrait être intégré de manière quasi immédiate).

L'analyseur syntaxique commence par construire une représentation abstraite des clauses lues dans l'objectif de dissocier le reste du traitement de la syntaxe des clauses. Ainsi, chaque terme est représenté par une structure de données sous la forme d'un arbre syntaxique (ou abstrait) où la racine contient le symbole de tête du terme et les fils représentent les sous-termes (s'ils existent). Un littéral est également représenté par un arbre dont la racine est le symbole de prédicat du littéral ainsi que son signe (littéral positif ou négatif). Une clause est représentée par une liste chaînée des représentations abstraites des littéraux de la clause et le résultat de cette passe est une liste chaînée (des représentations abstraites) des clauses lues. En même temps que les termes sont lus, nous procédons dans cette passe à une opération de compactage des termes. Cette opération permet de détecter qu'un terme est en fait une instance dépliée d'un N-terme. Par exemple, un terme du type $\underbrace{f(f(f(f(\ldots f(a)))))}_{n\,\text{fois}}$ est détecté comme étant une instance du N-terme $f(\diamond)^n.a$ et donc au lieu de construire un arbre syntaxique pour $\underbrace{f(f(f(f(\ldots f(a)))))}_{n\,\text{fois}}$, nous construisons un arbre pour $f(\diamond)^n.a$.

D'un point de vue abstrait, nous pouvons voir ce processus comme une application systématique (non déterministe) des règles suivantes :

- **Création de contexte.** $t \to t[\diamond]^1.s$ où s est un terme apparaissant dans t et $t[\diamond]$ désigne le contexte obtenu en remplaçant n'importe quelle occurrence de s dans t par $\diamond$ (suivant la restriction ci-dessus, $\diamond$ apparaît à une position de longueur 1 dans t).
- **Normalisation.** $t^n.t^m.s \to t^{n+m}.s$. Cette règle permet d'une part de fournir une représentation plus compacte du terme et d'autre part de détecter les séquences potentielles.

Par exemple, le terme $f(a, f(a, f(a, b)))$ est écrit par la première règle $f(a,\diamond)^1.f(a,\diamond)^1.f(a,\diamond)^1.b$. La règle de normalisation permet d'obtenir le terme $f(a,\diamond)^3.b$.

La règle de création de contexte peut clairement être appliquée sur n'importe quel terme et à n'importe quelle position dans ce terme. Ceci peut potentiellement engendrer un nombre très important de contextes différents. Pour éviter ce problème, cette règle n'est appliquée que sur les contextes itérés, c'est à dire que quand la règle de normalisation devient directement applicable sur les termes obtenus. Les termes sont analysés du bas vers le haut (les deux règles sont appliquées en utilisant une stratégie de réécriture du plus profond).

Classification des clauses. À la fin de l'étape précédente, les termes présents dans les clauses analysées sont représentés en utilisant des I-termes. Les clauses peuvent alors être classifiées en fonction des contextes itératifs identifiés (en associant les clauses/termes ayant un même contexte). La définition suivante formalise la notion de classe de clauses :

Définition 49. Pour chaque entier l, l'ensemble C des *classes de l-termes* et l'ensemble $C_\diamond$ des *classes de l-contextes* sont définis inductivement par :

- $V_X \in C$ et $\diamond \in C_\diamond$.
- Si $s_1, \ldots, s_k \in C$ et $arité(f) = k$ alors $f(s_1, \ldots, s_k) \in C$.
- Si $arité(f) = k$, $\forall i \in [1..k], s_1, \ldots, s_k \in C \cup C_\diamond$ et $\exists j \in [1..n], s_j \in C_\diamond$ alors $f(s_1, \ldots, s_k) \in C_\diamond$.
- Si $s \in C_\diamond$, $t \in C$, $n_1, \ldots, n_l \in E_N$ et $s \neq \diamond$ alors $t^{n_1,\ldots,n_l}.s \in C$.

■

Par définition, il est clair que chaque I-terme est également un terme de classe 1 correspondant au cas $l = 1$ dans le quatrième item.

Exemple 20. Si nous prenons $l = 4$, $x \in V_X$, $f, g, a, b \in \mathcal{F}$ alors $x \in C$, $f(a, b) \in C$, $g(\diamond)^{1,2,3,4}.a \in C$, $g(\diamond)^{2.N,4.N,6.N,8.N}.a \in C$, $f\left(a, g(\diamond)^{1,2,3,4}.a\right) \in C$. ■

Définition 50. Soient i, l des entiers tels que $i \leq l$. Un terme t est une *i-instance* d'une classe $\hat{t}$ de l-termes si une des conditions suivantes est vérifiée :
- $t = \hat{t}$.
- $t = f(t_1, \ldots, t_n), \hat{t} = f\left(\hat{t}_1, \ldots, \hat{t}_n\right)$ et pour chaque $j \in [1..n]$ t_j est une i-instance de $\hat{t}_j$.
- $\hat{t} = f\left(\hat{t}_1, \ldots, \hat{t}_n\right)^{n_1,\ldots,n_l}.\hat{s}$ et $t = f(t_1, \ldots, t_n)^{n_i}.s$ où $t_1, \ldots, t_n, s$ sont des i-instances de $\hat{t}_1, \ldots, \hat{t}_n, \hat{s}$ respectivement.

L'ensemble des i-instances d'une classe de termes $\hat{t}$ est noté $I(\hat{t})$. ■

Exemple 21. En reprenant les classes de l'exemple précédent, nous avons :
- x et $f(a, b)$ sont des instances des classes x et $f(a, b)$ respectivement.
- $g(\diamond)^1.a$, $g(\diamond)^2.a$, $g(\diamond)^3.a$ et $g(\diamond)^4.a$ sont des instances de la classe $g(\diamond)^{1,2,3,4}.a$ et $I\left(g(\diamond)^{1,2,3,4}.a\right) = \left\{g(\diamond)^i.a | i \in [1..4]\right\}$.
- $g(\diamond)^{2.N}.a$, $g(\diamond)^{4.N}.a$, $g(\diamond)^{6.N}.a$ et $g(\diamond)^{8.N}.a$ sont des instances de la classe $g(\diamond)^{2.N,4.N,6.N,8.N}.a$ et $I\left(g(\diamond)^{2.N,4.N,6.N,8.N}.a\right) = \left\{g(\diamond)^{i.N}.a | i \in \{2, 4, 6, 8\}\right\}$.
- $f\left(a, g(\diamond)^1.a\right)$, $f\left(a, g(\diamond)^2.a\right)$, $f\left(a, g(\diamond)^3.a\right)$ et $f\left(a, g(\diamond)^4.a\right)$ sont des instances de la classe $f\left(a, g(\diamond)^{1,2,3,4}.a\right)$ et $I\left(g(\diamond)^{1,2,3,4}.a\right) = \left\{f\left(a, g(\diamond)^i.a\right) | i \in [1..4]\right\}$.

■

Pour classer les I-termes obtenus, nous avons besoin d'un algorithme qui étant donné une classe de terme $\hat{t}$ et un terme s, trouve une nouvelle classe de termes notée $\hat{t} \star s$, telle que $I(\hat{t}) \cup \{s\} = I(\hat{t} \star s)$. Une telle classe n'existe pas toujours, mais quand elle existe elle est calculée comme suit (si N est une suite d'entiers et m un entier, alors $N.m$ désigne la concaténation de N et m) :
- $\hat{t} \star s \stackrel{def}{=} f(\hat{s}_1, \ldots, \hat{s}_n)^{N.m}.\hat{v}$ si $\hat{t} = f\left(\hat{t}_1, \ldots, \hat{t}_n\right)^N.\hat{u}$, $s = f(s_1, \ldots, s_n)^m.v$ et $\forall i \in [1..n], \hat{s}_i = \hat{t}_i \star s_i$ et $\hat{v} = \hat{u} \star v$.
- $\hat{t} \star s \stackrel{def}{=} f(\hat{s}_1, \ldots, \hat{s}_n)$ si $\hat{t} = f\left(\hat{t}_1, \ldots, \hat{t}_n\right)$, $s = f(s_1, \ldots, s_n)$ et $\forall i \in [1..n], \hat{s}_i = \hat{t}_i \star s_i$.
- Sinon $\hat{t} \star s$ n'existe pas.

Proposition 10. *Soient $\hat{t}$ une classe de termes et s un terme. S'il existe $\hat{s}$ telle que $\hat{s} = \hat{t} \star s$, alors $\hat{s}$ est une classe de k-termes (pour un entier k donné) et nous avons $I(\hat{s}) = I(\hat{t}) \cup \{s\}$.* ■

Preuve. La preuve est par induction sur la structure des classes de termes. □

Les littéraux et clauses sont traités de la même manière (en tenant compte que l'union de littéraux est associative et commutative). Les termes correspondant à une même classe (modulo le renommage des variables arithmétiques) peuvent être regroupés. L'algorithme 1 décrit la classification des clauses.

Généralisation des classes produites. Cette dernière passe permet d'analyser les suites (d'entiers ou de formules) contenues dans les classes obtenues pour trouver une formule mathématique qui généralise chaque suite (figurant dans un terme). En fait, notre implémentation est restreinte à des généralisations de suites polynomiales (les éléments de la suite sont des instances

Algorithm 1: ClauseClassification

input: Un ensemble de clauses
output: Δ, un ensemble de classes de clauses
foreach *Clause* C **do** `/* C a été transformée au préalable avec les règles «Création de contexte» et «Normalisation» */`
  foreach *classe de clause* $\hat{C}$ *dans* Δ **do**
    if $\hat{C'} = \hat{C} \star C$ *existe* **then**
      $\Delta = \Delta \backslash \{\hat{C}\} \cup \{\hat{C'}\}$;
  if *pas de classes trouvées* **then**
    $\Delta = \Delta \cup \{C\}$;

d'un polynôme) puisque ce type de suite est celui largement retrouvé en pratique. Beaucoup de travaux ont été mené pour étudier comment trouver le terme général d'une suite dont les premiers termes sont connus. L'encyclopédie OEIS fournit plus de 100000 suites dans ce sens. Plusieurs méthodes sont utilisées pour trouver la généralisation (polynomiale) d'une suite comme par exemple celle dite de transformation binomiale [BS95][4], ou la méthode de calcul avancé de déterminant [KA99] sur laquelle est fondé le programme `rate.m`[5] que nous utilisons dans notre implémentation.

Une limite de cette approche est qu'elle ne permet pas de séparer les suites imbriquées. En fait, il peut exister des suites d'entiers Σ pour lesquelles il est impossible de trouver une description polynomiale du terme général de Σ mais telles que si nous décomposons Σ en plusieurs sous-suites $\Sigma_1, \ldots \Sigma_k$ avec $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_k$, nous pouvons trouver une généralisation pour chaque sous-suite Σ_i. Le problème alors est que cette décomposition doit être la même pour toutes les suites d'une classe de clauses. Cette situation se produit si les clauses d'une classe donnée ne sont pas toutes générées par la même combinaison de règles d'inférence. L'exploitation des informations données par le démonstrateur à propos des règles utilisées pour générer chaque clause pourrait aider pour surmonter ce problème. Notre généralisation est « correcte » dans le sens où les I-termes obtenus sont toujours des généralisations strictes des suites de termes générées par le démonstrateur. Cependant, naturellement, comme nous utilisons une approche inductive, nous ne pouvons pas assurer que les formules obtenues sont des conséquences logiques des formules initiales. Elles doivent être traitées comme des lemmes potentiellement utiles pour éviter la divergence par exemple. Ainsi, une vérification a posteriori est nécessaire. Cette étape de vérification n'est pas intégrée actuellement dans notre implémentation.

4.2.2 Comparaison avec d'autres approches

Notre approche est - à notre connaissance - originale dans le sens où aucune autre approche basée sur des idées similaires n'a été proposée. Il en va de même pour notre implémentation. En effet, les approches qui ont été proposées avant notre travail essaient de générer automatiquement des I-termes à partir des axiomes, en détectant des cycles potentiels dans les inférences. En particulier [Sal92] se base essentiellement sur la détection de clauses *auto-résolvantes* comme par exemple la clause $\neg P(x) \vee P(f(x))$. L'application de la résolution entre cette clause et une variante d'elle même peut engendrer la clause $\neg P(x) \vee P(f(f(x)))$, puis l'application de la résolution entre

[4]Le lien http ://www.research.att.com/∼njas/sequences/seqtranslib.txt fournit fournit une implémentation Mathematica de la transformation binomiale en plus d'autres méthodes.
[5]http ://www.mat.univie.ac.at/∼kratt/rate/rate.html

cette dernière clause et une variante d'elle même engendre $\neg P(x) \vee P(f(f(f(x))))$ et ainsi de suite conduisant ainsi à la divergence. Dans certains cas, ces cycles peuvent être détectés en utilisant des critères syntaxiques relativement simples, puis les I-termes correspondants (qui peuvent être générés par ces cycles) sont introduits pour décrire intégralement l'ensemble des clauses déductibles ($\neg P(x) \vee P(f^n(\diamond).x)$ dans ce cas, où $n \in \mathbb{N}^*$)[6]. Une approche plus générale a été proposée dans [Pel01a]. Elle utilise une analyse plus sophistiquée de l'arbre de dérivation de la preuve pour détecter des cycles plus complexes qui ne peuvent pas être générés de manière explicite par la procédure de preuve.

Notre démarche dans ce travail est fondamentalement différente :

- Elle est *complètement indépendante* de la procédure de preuve considérée.
- Elle utilise des techniques de *généralisation inductive* au lieu de celles déductives utilisées dans les travaux cités plus haut.

Le fait d'utiliser des techniques inductives implique que les suites générées *ne sont pas* en général des conséquences logiques des axiomes comme c'est le cas avec les approches basées sur des méthodes déductives. Elles doivent être considérées plutôt comme des « conjectures » potentiellement utiles. Ainsi une phase de vérification et de validation de ces conjectures est nécessaire. D'un autre côté, en dépit de cette limitation, disposer de plus d'informations que celles fournies par les conséquences logiques des axiomes s'avère souvent utile comme évoqué plus haut, pour détecter la satisfaisabilité ou pour suggérer des lemmes. Plus de détails à propos de ce point seront fournis à la section 4.3.

Un travail dont l'esprit pourrait être comparé au nôtre est celui de [AHL97] où un algorithme de généralisation à la Plotkin [Plo71] est proposé pour obtenir un I-terme qui généralise un ensemble de termes standards. Dans leur approche, les auteurs se basent essentiellement sur une énumération de l'ensemble des I-termes. La disunification [Com91, Pel97] est utilisée pour rejeter les généralisations non minimales. Les I-termes sont inférés de la façon suivante : étant donnés deux termes t et s, la première étape consiste à décomposer t et s sous la forme de I-termes, c'est à dire trouver des termes u, v, w, z et deux entiers n et m non nuls tels que $t = u^n.v$ et $s = w^m.z$. Ensuite, dans la deuxième étape, il s'agit de trouver une généralisation (à la Plotkin) de u et w et de v et z respectivement. Il n'existe pas à notre connaissance d'implémentation de cette méthode, ainsi il ne nous a pas été possible d'effectuer des comparaisons expérimentales.

Cet algorithme convient à des ensembles (de termes) de taille réduite, mais quand il s'agit de traiter de grands ensembles (comme c'est le cas pour notre méthode), le coût de la vérification de toutes les décompositions possibles pour chaque terme risque de devenir exorbitant. Dans la méthode que nous proposons, nous utilisons des algorithmes de généralisation basés sur des outils de calcul symbolique. Ceci a trois avantages :

- Nous n'avons besoin de considérer que les décompositions minimales. Par exemple si l'exécution du démonstrateur génère la suite (de termes) $f(f(a))$,$f(f(f(f(a))))$, $f(f(f(f(f(f(a))))))$, ... qu'il faut généraliser, alors notre algorithme génère les termes (compactés) $f^2(a)$, $f^4(a), f^6(a), \ldots$, ce qui donne la suite $2, 4, 6, \ldots$ à généraliser. Finalement DS3 infère $f^{2.N}(\diamond).a$.

  En revanche, l'algorithme de [AHL97] (utilisant une généralisation à la Plotkin) génère :
  - soit la généralisation usuelle obtenue par décomposition et contradiction : $f(f(x))$.
  - soit $f(\diamond)^2.a, f(\diamond)^4.a, f(\diamond)^6.a, \ldots$ ce qui donne $f(\diamond)^N(a)$.
  - ou bien $f(f(\diamond))^1.a, f(f(\diamond))^2.a, f(f(\diamond))^3.a$ ce qui donne $f(f(\diamond))^N(a)$.
  - ou encore des décompositions mixtes telles que $f(\diamond)^2.a, f(f(\diamond))^2.a$, $f(f(\diamond))^3.a$ ce qui donne x (x est une variable).

  Les solutions sont filtrées ensuite pour ne garder que celles qui sont minimales (i.e. les plus instanciées, ici $f(f(\diamond))^N(a)$). Nous obtenons donc le même résultat même sans avoir à

[6] En toute rigueur, on peut même dire que $n \in \mathbb{N}$, puisque le cas $n = 0$ correspond à la tautologie $\neg p(x) \vee p(x)$

considérer avec notre algorithme tous les cas possibles, mais uniquement les décompositions minimales.
- Nous pouvons obtenir des caractérisations plus fines puisque nous ne sommes pas restreints aux suites linéaires. En effet, notre algorithme est capable d'inférer des généralisations du type $t^{P(N_1,\dots N_k)}.s$ par exemple où $P(N_1,\dots N_k)$ est un polynôme de degré quelconque en $N_1,\dots N_k$. L'algorithme de [AHL97] n'infère que des N-termes du type $t^N.s$, i.e. les exposants sont des polynômes de degré au plus 1. Voir l'exemple de la sous-section 4.3.2 pour plus de détails.
- Il est très difficile d'isoler parmi toutes les clauses générées celles qui correspondent à une suite donnée. Or l'application de l'algorithme de généralisation de [AHL97] à l'ensemble complet des conséquences ne permet pas en général de trouver une généralisation pertinente (autre qu'une variable). Notre approche permet justement de surmonter cette limitation mais ceci a malheureusement un prix : une seule variable arithmétique peut être introduite à la fois (puisque nous ne traitons que des *suites* de termes). Cette variable correspond en fait à une direction particulière dans l'espace de recherche. Ceci dit, il est tout à fait envisageable d'introduire plusieurs variables successivement en répétant l'application de notre algorithme (voir la sous-section 4.3.2).

Notre approche s'apparente aussi à la méthode dite *critique de divergence* (« divergence critic » dans le texte original) développée dans [Wal96] pour résoudre le problème de divergence dans le contexte des preuves par induction. Comme pour DS3, le critique de divergence est indépendant de la procédure de preuve. La méthode utilise des heuristiques intelligentes basées sur la technique dite de « rippling » introduite initialement dans [Aub76] puis reprise et détaillée dans [Bun88], pour générer des descriptions générales des séquences de termes inférés. La différence majeure avec notre approche est que le formalisme de schématisation de termes que nous utilisons pour décrire l'espace de recherche a plus de pouvoir d'expression que les termes standard (de premier ordre) utilisés dans le critique. Ceci nous permet d'obtenir *des généralisations plus précises.*

4.3 Études de cas

4.3.1 Syntaxe de la sortie de DS3

Nous donnons ici quelques précisions sur la syntaxe de sortie de DS3 pour une bonne lisibilité de la suite. DS3 affiche la liste des classes de clauses trouvées. Pour chaque classe, le nombre d'instances (c'est à dire les clauses appartenant à la classe de clauses) est imprimé.

Une classe de termes $f(t_1,\dots,t_n)^n.s$ (ou tout simplement un I-terme) avec $arité(f) > 1$ est représentée en sortie par la syntaxe $[f,n](s_1,\dots,s_n)$ avec $s_i = \$$ si t_i est le trou et $s_i = t_i$ dans le cas contraire. Si $arité(f) = 1$ alors un terme $f(\diamond)^n.x$ est représenté en sortie (simplement) par $[f,n](x)$.

Les expressions arithmétiques $x+y$, $x \times y$, $\frac{x}{y}$ et x^y sont respectivement représentées (suivant la syntaxe de Mathematica) par `Plus[x,y]`, `Times[x,y]`, `Rational[x,y]` et `Power[x,y]`.

Un littéral positif $P(X,Y)$ est noté `++P(X,Y)` et un littéral négatif $\neg P(X,Y)$ est noté `--P(X,Y)`.

4.3.2 Caractérisation de suites

Le premier exemple que nous présentons sert surtout à illustrer comment notre approche fonctionne en pratique. Nous considérons les clauses de Horn suivantes, qui correspondent à

l'énumération usuelle des nombres rationnels (représentés sous la forme de couples (x, y)) :

$$S = \left\{ \begin{array}{rcl} P(s(0), s(0), s(0)) & & \\ P(s(0), s(X), s(N)) & \vee & \neg P(X, s(0), N) \\ P(s(X), Y, s(N)) & \vee & \neg P(X, s(Y), N) \end{array} \right.$$

$P(x, y, z)$ signifie que le rang de $\frac{x}{y}$ dans l'ensemble des nombres rationnels (qui est dénombrable) est z. L'utilisation de la résolution positive unitaire (et/ou de l'hyper-résolution [Rob65a]), génère une suite infinie de clauses de la forme $P(1, 1, 1)$, $P(1, 2, 2)$, $P(2, 1, 3)$, $P(1, 3, 4)$, $P(2, 2, 5)$, $P(3, 1, 6)$, $P(1, 4, 7)$, ... où l'entier k désigne le terme $s^k(0)$ (dans l'arithmétique de Presburger). Nous ne pouvons pas trouver une généralisation pertinente *directement*, car la valeur de z dépend de *deux* entiers correspondant respectivement à x et à y.

Afin de contourner ce problème, nous fixons (interactivement) la valeur de x, par exemple nous prenons $x = 1$. L'application de DS3 aux sorties produites par la restriction de l'ensemble S au cas $x = 1$ permet de générer le résultat décrit par la figure 4.2.

```
Found 1 different form(s)
++p([s,1](0),[s,Plus[1,n]](0),[s,Plus[1,Times[Rational[1,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))

Current Form has 14 different instances found

Printing Instances of the current Form
++p([s,1](0),[s,1](0),[s,1](0))
++p([s,1](0),[s,2](0),[s,2](0))
++p([s,1](0),[s,3](0),[s,4](0))
++p([s,1](0),[s,4](0),[s,7](0))
++p([s,1](0),[s,5](0),[s,11](0))
% ... [skipped]
**********************
```

FIG. 4.2 – Application de DS3 au problème d'énumération usuelle des nombres rationnels : cas $x = 1$

La sortie de DS3 indique donc qu'il a trouvé une seule classe de clauses qui est $P(s(0), s(\diamond)^{n+1}.0, s(\diamond)^{1+\frac{1}{2}\times n+\frac{1}{2}\times n^2}.0)$ soit $P(s(0), s(s(\diamond)^n.0), s(s(\diamond)^{\frac{n.(n+1)}{2}}.0))$.

DS3 a trouvé 14 instances sur lesquelles il s'est basé pour inférer cette classe. Une partie de ces instances est affichée : $P(s(0), s(\diamond)^1.0, s(\diamond)^1.0)$, $P(s(0), s(\diamond)^2.0, s(\diamond)^2.0)$, $P(s(0), s(\diamond)^3.0, s(\diamond)^4.0)$, ...

L'algorithme de généralisation de [AHL97] en revanche n'est capable d'inférer que la suite $P(s(0), s(s^n(0)), s(s^m(0)))$ qui est clairement plus générale (c'est à dire moins précise) que le résultat retourné par DS3 (qui est $P(s(0), s(s^n(0)), s(s^{1+\frac{n.(n+1)}{2}}(0))))$.

Pour $x = 2$ nous obtenons le résultat décrit par la figure 4.3 :

```
Found 1 different form(s)
++p([s,2](0),[s,Plus[1,n]](0),[s,Plus[3,Times[Rational[3,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))
```

FIG. 4.3 – Application de DS3 au problème d'énumération usuelle des nombres rationnels : cas $x = 2$

La sortie de DS3 indique dans ce cas qu'il a trouvé une seule classe (pour $x = 2$) qui est $P(s(\diamond)^2.0, s(\diamond)^{n+1}.0, s(\diamond)^{3+\frac{3}{2}\times n+\frac{1}{2}\times n^2}.0)$. L'affichage des instances est omis dans cette figure.

En répétant cette opération plusieurs fois, nous obtenons une *suite de I-termes*. DS3 peut être appliqué récursivement à cette suite, ce qui donne le résultat décrit par la figure 4.4 :

```
==============================
Found 1 different form(s)
++p([s,Plus[1,m]](0),[s,Plus[1,n]](0),[s,Plus[1,Times[Rational[1,
2],n],Times[Rational[1, 2],Power[n,2]],Times[Rational[3,
2],m],Times[n,m],Times[Rational[1, 2],Power[m,2]]]](0))
Printing Instances of the current Form
++p([s,1](0),[s,Plus[1,n]](0),[s,Plus[1,Times[Rational[1,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))
++p([s,2](0),[s,Plus[1,n]](0),[s,Plus[3,Times[Rational[3,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))
++p([s,3](0),[s,Plus[1,n]](0),[s,Plus[6,Times[Rational[5,
2],n],Times[Rational[1, 2],Power[n,2]]]](0))
... [skipped]
**************************
End processing
```

FIG. 4.4 – Application de DS3 au problème d'énumération usuelle des nombres rationnels : cas général

Nous obtenons donc la caractérisation suivante des nombres rationnels :

$$P(s(\diamond)^{1+m}.0, s(\diamond)^{1+n}.0, s(\diamond)^{1+\frac{1}{2}\times n+\frac{1}{2}\times n^2+\frac{3}{2}\times m+n\times m+\frac{1}{2}\times m^2}.0)$$

4.3.3 Détection de la satisfaisabilité et construction de modèles

Nous montrons maintenant l'utilité de notre démarche dans la détection de la satisfaisabilité et la construction de modèles. Pour cela, nous considérons le problème suivant :

$$S = \left\{ \begin{array}{c} P(x,x,z) \\ P(f(x,g(z),x),y,z) \vee \neg P(x,f(y,g(z),y),z) \\ \neg P(x,f(x,a),z) \end{array} \right.$$

Cet exemple d'apparence simple est difficile à traiter car l'ensemble de clauses S n'a que des modèles de cardinalité infinie. Par conséquent, les constructeurs de modèles finis ne sont pas

utiles. De plus, les stratégies usuelles du calcul de résolution (aussi bien que d'autres méthodes de preuve comme la méthode des tableaux sémantiques [Häh01] ou celle des connexions [Bib81, CP00]) ne terminent pas sur S.

Le démonstrateur E produit - en utilisant un paramétrage standard - un nombre infini de clauses. Il est incapable donc de prouver que S est satisfaisable. Nous utilisons notre méthode pour analyser l'espace de recherche obtenu. DS3 génère la suite de clauses suivante (la sortie détaillée du programme est donnée dans l'annexe A.1) :

$$P(f(\diamond, g(z), \diamond)^{2\times n}.x, x, z)$$

La clause unitaire $\{P(f(\diamond, g(z), \diamond)^{2\times n}.x, x, z)\}$ définit un modèle de Herbrand de l'ensemble S : $P(u, v, w)$ est vrai si v est de la forme $P(f(\diamond, g(w), \diamond)^{2\times n}.u, u, w)$ pour un n donné ([CLP04, Pel97] fournissent plus de détails sur l'utilisation des I-termes pour représenter les interprétations de Herbrand). Ceci peut être automatiquement vérifié, puisque la théorie de premier ordre des I-termes est décidable [Pel97].

Nous donnons un autre exemple plus simple extrait de [DG79].

$$S' = \left\{ \begin{array}{rcl} Q(x, f(y)) & \vee & \neg P(x, y) \\ Q(x, f(y)) & \vee & \neg Q(x, y) \\ P(x, f(x)) & \vee & \neg Q(x, f(x)) \\ P(x, f(x)) & & \\ \neg Q(x, f(x)) & & \end{array} \right.$$

S' est satisfaisable, mais *n'a pas de modèle fini*. E-prover ne termine pas sur S'. En analysant les clauses générés, DS3 produit les suites suivantes : $\neg Q(f^n(x), x)$ et $\neg P(f^n(x), x)$. Là encore, ces deux clauses unitaires définissent un modèle de S' : $P(x, y)$ et $Q(x, y)$ sont vrais si et seulement si $x \neq f^n(y)$.

4.3.4 Génération de lemmes inductifs

La méthode de preuve par consistance ou « Induction sans induction » introduite au chapitre 2 illustre particulièrement bien l'utilité des techniques et idées à la base de notre approche.

Les démonstrateurs existants sont dans la plupart des cas incapables de terminer sur $A \cup E \cup C$ (en reprenant les notations de la section 2.3) comme expliqué dans [Wal96], ce qui empêche de vérifier la satisfaisabilité de l'ensemble. Pour pouvoir obtenir un ensemble de clauses saturé (et donc faire converger le calcul), il faut souvent ajouter des lemmes inductifs. Ces lemmes établissent des propriétés qui ne sont pas valides en général mais uniquement dans des interprétations de Herbrand. En général, ces lemmes sont fournis par l'utilisateur. Une application intéressante de notre approche est qu'elle permet de suggérer *automatiquement* des lemmes inductifs permettant dans certains cas de faire converger le calcul.

Nous illustrons ces propos à travers un exemple simple.

Exemple 22. Nous considérons une fonction $l0$ qui associe à chaque entier n une liste de taille n ne contenant que des '0'. En particulier, $l0(1) = (0), l0(2) = (0, 0), \ldots$. Nous considérons également une fonction *count* qui compte le nombre d'occurrences de l'élément 0 dans une liste l. Les fonctions $l0$ et *count* peuvent être définies par l'axiomatisation A suivante.

$$A = \left\{ \begin{array}{rcl} l0(0) & \approx & nil \\ count(l) & \approx & count'(l, 0) \\ count'(cons(0, l), n) & \approx & count'(l, s(n)) \\ l0(s(n)) & \approx & cons(0, l0(n)) \\ count'(nil, n) & \approx & n \\ count'(cons(s(n), l), m) & \approx & count'(l, m) \end{array} \right.$$

Nous souhaitons démontrer pour chaque liste l la propriété suivante : $P(n)$: $count(l0(n)) \approx n$. Il s'agit d'une propriété inductive. En effet, il est facile de vérifier que $count(l0(n)) \approx n$ *n'est pas* une conséquence logique de A, en revanche, toute instance fermée de $count(l0(n)) \approx n$ est une conséquence logique de A. La preuve par consistance peut être utilisée pour établir la propriété.

Pour cela, nous avons besoin d'ajouter à l'ensemble A à la fois le théorème à prouver ($count(l0(n)) \approx n$ dans ce cas) et une I-axiomatisation. Ensuite, il faut s'assurer que l'ensemble obtenu est insatisfaisable, ce qui assure qu'il existe au moins un modèle de Herbrand qui valide $count(l0(n)) \approx n$ (l'I-axiomatisation a pour rôle de garantir que le modèle est minimal).

L'exécution de E-prover sur cet exemple ($A + P(n) + I$-axiomatisation) ne termine pas et donc *échoue à détecter la satisfaisabilité*. Le démonstrateur SPIKE [BKR92] conçu pour traiter les problèmes inductifs diverge aussi sur cet exemple. DS3 en revanche génère la suite suivante : $s(\diamond)^n.X \approx count'(l0(x), n)$ (voir l'annexe A.2 pour plus de détails). Ceci suggère d'ajouter cette propriété aux hypothèses comme un lemme inductif. Comme E-prover n'est pas capable de manipuler des clauses construites à partir de I-termes, cette expression doit être traduite en logique du premier ordre. Nous proposons le codage F suivant (qui nous semble le plus naturel) :

$$F = \left\{ \begin{array}{rcl} f(x,0) & \approx & x \\ f(x,s(n)) & \approx & s(f(x,n)) \end{array} \right.$$

où f est un nouveau symbole de fonction et $f(x,n)$ est utilisé pour coder le I-terme $s(\diamond)^n.x$.

$s(\diamond)^n.X \approx count'(l0(x), n)$ devient alors $f(x,n) \approx count'(l0(x), n)$. En ajoutant cette expression plus les axiomes de F à l'ensemble initial ($A + P(n) + I$-axiomatisation), nous obtenons un nouvel ensemble de clauses sur lequel E-prover termine. Et donc, la satisfaisabilité est détectée et **le théorème inductif $P(n)$ est démontré**. ■

Il est clair que tout le processus peut être automatisé. Des exemples plus compliqués peuvent bien sûr demander une aide à l'utilisateur qui doit choisir le lemme « convenable » parmi une liste de lemmes candidats, mais ceci ne met pas en cause l'utilité de notre méthode. Naturellement le type de lemmes que nous pouvons engendrer grâce à cette technique est également limité par le pouvoir d'expression des I-termes.

4.3.5 DS3 et grammaires hors contexte

Nous proposons deux exemples du domaine de la théorie des langages pour illustrer d'autres aspects du champ d'application de notre approche.

Terme général d'un langage hors contexte

Dans le premier exemple, il s'agit de découvrir le terme général d'un langage défini par une grammaire hors contexte. Soit L le langage défini par la grammaire hors contexte G :

$$\begin{array}{rcl} S & \rightarrow & ab \\ S & \rightarrow & aSb \end{array}$$

Il est facile de démontrer que $L = \{a^n b^n | n \in \mathbb{N}\}$.

Nous utilisons les possibilités de DS3 pour suggérer ce résultat par le biais de la formulation en logique du premier ordre suivante :

$$W = \left\{ \begin{array}{c} w(f(a,b)) \\ w(f(a,f(x,b))) \vee \neg w(x) \\ f(x,f(y,z)) \approx f(f(x,y),z) \end{array} \right.$$

$w(t)$ signifie que t est un mot du langage L et la fonction f représente la concaténation de deux mots, ainsi $f(t_1, t_2)$ est le résultat de la concaténation de t_1 et de t_2.

En utilisant une stratégie négative sur le problème W, E diverge en générant une suite infinie de clauses. L'analyse de cette suite par DS3 permet de trouver les deux formes génériques $w(f(a, f(f(a, \diamond)^{N+2}.f(b, \diamond)^{N+1}.b, b)))$ et $w(f(a, \diamond)^{N+2}.f(b, \diamond)^{N+1}.b)$. Ces deux formules équivalentes décrivent donc le terme général du langage L, soit $L = \{a^n b^n | n \in \mathbb{N}\}$.

Ceci pourrait aider par exemple un étudiant qui voudrait connaître la forme des mots produits par une grammaire pour pouvoir la donner à prouver à un démonstrateur par induction. À ce titre, l'utilisation de notre démonstrateur DEI une fois que notre algorithme d'indexation des I-termes (cf. chapitres 6 et 7) sera implémentée (ce qui permettra de gérer la subsomption avec des I-termes) pourrait s'avérer intéressant en procédant par induction sans induction. Le problème donné à DEI contiendrait le système W plus les deux formules génériques découvertes par DS3.

L'intérêt de cet exemple est qu'il illustre des possibilités que l'on n'attend pas (encore) d'un démonstrateur automatique.

Description(s) partielle(s) d'un espace de recherche

Dans le deuxième exemple, nous présentons un cas à la limite de ce que permet de faire DS3. En effet et comme mentionné en introduction de ce chapitre, il n'est malheureusement pas toujours possible en général de structurer totalement un espace de recherche. Sur certains problèmes (comme ceux illustrés par les autres exemples de cette section) il est possible de le faire. En revanche, sur d'autres problèmes comme celui-ci, la structuration possible n'est que partielle.

Nous nous intéressons à un langage particulier, celui des *palindromes* sur l'alphabet $\{a, b\}$. Un palindrome sur un alphabet V est un mot $w = c_1.c_2.\ldots.c_n$ tel que $\{c\}_{1 \leq i \leq n} \subseteq V$ et $w' = w$ où $w' = c_n.\ldots.c_2.c_1$, i.e. un palindrome est un mot qui peut être lu de gauche à droite ou de droite à gauche de la même manière. Soit la grammaire hors contexte suivante G' [HMU00] :

$$\begin{array}{lcl} S & \rightarrow & a \\ S & \rightarrow & b \\ S & \rightarrow & \epsilon \\ S & \rightarrow & aSa \\ S & \rightarrow & bSb \end{array}$$

Il est simple à démontrer (par induction) que langage engendré par G' est celui des palindromes sur le vocabulaire $\{a, b\}$.

Notre objectif est d'essayer de retrouver ce résultat à l'aide de DS3. Pour cela, nous utilisons la formulation en logique du premier ordre suivante :

$$W' = \left\{ \begin{array}{rcl} p(a) & & \\ p(b) & & \\ p(e) & & \\ w(f(a, f(x, a))) & \vee & \neg p(x) \\ w(f(b, f(x, b))) & \vee & \neg p(x) \\ f(e, a) & \approx & a \\ f(e, b) & \approx & b \\ f(a, e) & \approx & a \\ f(b, e) & \approx & b \\ f(x, f(y, z)) & \approx & f(f(x, y), z) \end{array} \right.$$

e est une constante désignant le symbole epsilon, $p(t)$ signifie que t est un palindrome et f désigne comme pour l'exemple précédent la fonction de concaténation.

En utilisant une stratégie négative sur le problème W', E diverge en générant une suite infinie de clauses. L'analyse de l'espace de recherche par DS3 ne permet pas de trouver de formules générales (à base de I-termes) pour décrire la totalité de l'espace de recherche, ce qui est normal puisqu'il est possible d'établir que l'ensemble des palindromes sur le vocabulaire $\{a, b\}$ ne peut être représenté par aucun ensemble fini de I-termes. En revanche, DS3 fournit des descriptions structurelles pour des sous-ensembles de l'espace de recherche. Nous donnons des exemples de ces descriptions partielles inférées par DS3 :

$p(f(a,\diamond)^{N+1}.a)$	a^{N+2}	(notation compacte)
$p(f(b,\diamond)^{N+1}.b)$	b^{N+2}	
$p(f(a,f(f(a,\diamond)^{N+1}.a,a)))$	$a.a^{N+2}.a$	
$p(f(b,f(f(a,\diamond)^{N+1}.a,b)))$	$b.a^{N+2}.b$	
$p(f(a,f(b,f(f(a,\diamond)^{N+1}.b,a))))$	$a.b.a^{N+1}.b.a$	
$p(f(b,\diamond)^2.f(a,\diamond)^{N+1}.f(b,b))$	$b^2.a^{N+1}.b.b$	
$p(f(a,f(f(b,\diamond)^2.f(a,\diamond)^{N+1}.f(b,b),a)))$	$a.b^2.a^{N+1}.b.b.a$	
$\ldots$		

DS3 retrouve donc des descriptions partielles de certains types de palindromes.

4.3.6 Découverte du spectre d'une formule

Nous concluons cette section par un exemple relatif à la découverte du *spectre* d'une formule. Nous rappelons que le spectre d'une formule est l'ensemble des cardinalités des modèles finis de la formule. Nous considérons la proposition P suivante : « une fonction injective est surjective ». P est valide sur tout domaine fini mais elle peut avoir des contre-exemples (sur des domaines infinis). En effet, il suffit de prendre la fonction $f : \mathbb{N} \to \mathbb{N}$ telle que $\forall n \in \mathbb{N}$, $f(n) = n + 1$. f est injective mais elle n'est pas surjective puisque $0 \in \mathbb{N}$ et $\forall n \in \mathbb{N}$, $f(n) \neq 0$. Nous considérons la formulation en logique du premier ordre suivante :

$$\phi = \begin{cases} (x \approx y) \quad \vee \quad \neg(f(x) \approx f(y)) \\ \qquad\qquad\quad\ \ \neg(f(x) \approx c) \end{cases}$$

La première clause signifie que f est injective et la deuxième clause qu'elle n'est pas surjective. E converge rapidement et répond satisfaisable (puisque P n'est pas insatisfaisable). En revanche Prover9 [McC10] diverge en générant la suite (infinie) de clauses $\{f(x) \not\approx c, f(f(x)) \not\approx f(c), f(f(f(x))) \not\approx f(f(c)), \ldots\}$. L'analyse de cette sortie par notre approche permet de déduire la forme générale $f(\diamond)^{N+1}.x \not\approx f(\diamond)^N.c$. Cette dernière formule implique que pour tout $n, m \in \mathbb{N}$, $n \neq m \Rightarrow f(\diamond)^n.c \not\approx f(\diamond)^m.c$. La preuve de cette assertion est simple à établir par récurrence sur m : si $m = 0$ alors en instanciant N avec 0 (dans la forme générale trouvée) nous avons $f(x) \not\approx c$ pour tout x et donc en particulier, pour tout $k \in \mathbb{N}$ $f(\diamond)^{k+1}.c \not\approx c$ soit $f(\diamond)^n.c \not\approx c$ où $n \neq 0$. Supposons que l'assertion est valable jusqu'à m. Nous montrons qu'elle est aussi valable pour $m + 1$. Soit $n \in \mathbb{N}$. Si $n \leq m$ alors par hypothèse de récurrence (appliquée à n puisque $n \leq m$) nous avons $\forall k \in \mathbb{N}$, $k \neq n \Rightarrow f(\diamond)^k.c \not\approx f(\diamond)^n.c$ et donc en particulier pour $k = m + 1$ nous avons $f(\diamond)^n.c \not\approx f(\diamond)^{m+1}.c$ (la relation $\approx$ est commutative et sa négation l'est aussi). Si $n > m + 1$ alors soit $k = n - m - 2$ ($k \in \mathbb{N}$). En instanciant N avec $m + 1$ dans la forme générale nous avons $\forall x, f(\diamond)^{m+2}.x \not\approx f(\diamond)^{m+1}.c$ et donc en particulier pour $x = f(\diamond)^k.c$ nous avons $f(\diamond)^{m+2}.f(\diamond)^k.c \not\approx f(\diamond)^{m+1}.c$ soit $f(\diamond)^{m+2+k}.c \not\approx f(\diamond)^m.c$ et donc $f(\diamond)^n.c \not\approx f(\diamond)^m.c$. Ainsi, les puissances successives de la fonction f sur c ($\{f(\diamond)^k.c\}_{k\in\mathbb{N}}$) sont deux à deux distinctes et donc un modèle de ϕ ne peut pas être fini (par le principe des cages à pigeons). Le spectre de ϕ est donc égal à l'ensemble vide.

DS3 aide dans cet exemple à découvrir le spectre d'une formule, chose que l'on ne demande pas (encore) à un démonstrateur mais qui est très informative comme illustré ici.

On note également, qu'afin de générer des lemmes aussi informatifs que possible, il est préférable d'utiliser des stratégies de recherche de preuve aussi peu restrictives que possible (pour engendrer un ensemble de conséquences suffisamment riche).

Troisième partie

Calculs et I-termes

Introduction

Les techniques décrites dans la partie précédente permettent d'engendrer des I-termes à partir d'ensembles de clauses standards. Afin de tirer partie de l'information ainsi extraite, il faut ensuite disposer de procédures de preuve capables de prendre en charge les clauses avec I-termes, soit pour démontrer les lemmes que l'on vient d'engendrer, soit pour les utiliser afin de déduire de nouvelles formules. Cela pose un double problème : d'une part, il s'agit d'étendre les règles d'inférence aux clauses avec I-termes tout en préservant (si possible) leurs propriétés théoriques (correction et complétude pour la réfutation). D'autre part, il faut étendre aux I-termes les techniques couramment utilisées en démonstration automatique pour la mise en oeuvre *efficace* de ces calculs (car l'existence d'un calcul n'est pas suffisante en pratique pour définir une procédure de preuve efficace). Parmi ces techniques, les méthodes d'indexation des termes jouent un rôle fondamental. Elles sont essentielles pour implémenter les règles de simplification (telle que la règle de subsomption) qui permettent, en éliminant les clauses redondantes, de limiter la taille des ensembles de clauses engendrées. Notre objectif est d'étendre les techniques de démonstration par saturation (utilisant des termes standards) aux I-termes. Notre contribution se décline en deux axes : au premier chapitre, nous étendons les calculs de résolution et de superposition aux I-termes et nous établissons les propriétés théoriques des calculs ainsi étendus. Au second, nous étendons le formalisme des *arbres de discrimination parfaits* [McC92, RSV01, Gra96] qui sont l'une des techniques d'indexation de termes les plus utilisées. L'une des différences principales avec le cas standard est que les graphes que nous construisons peuvent contenir des cycles (codant les itérations de contexte dans un I-terme), ce qui complique notablement les algorithmes de recherche. Enfin, le chapitre 7 est consacré à une description du système DEI (sigle de **D**eduction with the **E**-prover and **I**-terms). DEI est une extension du démonstrateur « E » [Sch04] réalisé par S. Schultz. Il s'agit à notre connaissance du premier démonstrateur intégrant des schématisations de termes.

Chapitre 5

Règles de déduction

dans ce chapitre sur l'extension des procédures de preuve par résolution et superposition à des clauses contenant des I-termes. Ce chapitre est organisé comme suit : À la section 5.1, les notions de position et d'ordre sur les termes/clauses sont étendues aux I-termes. À la section 5.2, nous étudions le cas non équationnel où nous présentons une de nos contributions, à savoir une preuve que l'extension du calcul par résolution ordonnée conserve la propriété de complétude réfutationnelle. À la section 5.3, nous traitons le cas (purement) équationnel où nous présentons trois contributions : nous expliquons pourquoi le calcul de superposition n'a plus la propriété de complétude réfutationnelle en présence de I-termes, nous présentons une nouvelle règle d'inférence appelée *H-superposition* qui restaure la complétude réfutationnelle (en présence de I-termes) et finalement nous fournissons une démonstration de la complétude réfutationnelle du nouveau calcul (superposition + H-superposition).

5.1 Définitions et notations

5.1.1 Positions et sous-termes

Afin d'étendre les règles de remplacement aux clauses contenant des I-termes, il est nécessaire d'étendre la notion de position. À noter que cette extension peut-être réalisée de plusieurs façons possibles : soit en considérant toutes les positions dans un I-terme de manière standard, soit en considérant uniquement un sous-ensemble de ces positions (par exemple celles correspondant à un I-terme, ou à l'extérieur des N-termes) ou bien encore en considérant toutes les positions contenues dans les termes standards représentés par le I-terme (dans ce dernier cas, l'ensemble de positions est évidemment infini). Comme pour le cas standard, une position est une suite finie de nombres entiers. Λ dénote toujours la position vide et l'opérateur . dénote la concaténation de positions et nous écrivons parfois p^k au lieu de $p.p.\ldots.p$ pour désigner la concaténation k fois de la position p. La définition 51 formalise la fonction Pos qui donne l'ensemble des *positions* dans un I-terme. La définition est légèrement différente de celle fournie dans [Com95] mais utilise les mêmes notations. Ainsi, étant donné un I-terme t, $Pos(t)$ contient les positions correspondant à tous les I-termes survenant dans t (sous-termes) en considérant même ceux apparaissant dans un contexte inductif ou un terme de base d'un N-terme.

Exemple 23. Soit $t = f(a, g(\diamond))^N.g(b)$. Alors $Pos(t) = \{\Lambda, 1.1, 2\}$. La position Λ correspond au terme t lui même. Nous parlons dans ce cas de *position racine*. 1.1 correspond au sous-terme a, 2 correspond au terme de base $g(b)$ et 2.1 correspond au sous-terme (du terme de base) b. A

noter que 1, 1.2 et 1.2.1 n'appartiennent pas à $Pos(t)$ puisqu'ils correspondent respectivement aux termes $f(a, g(\diamond))$, $g(\diamond)$ et $\diamond$ qui *ne sont pas* des I-termes. ■

Définition 51 (Positions d'un I-terme)**.** La fonction Pos retournant l'ensemble des *positions d'un terme dans* $T_I \cup T_\diamond$ est définie par :
- $Pos(t) = \{\Lambda\}$ si $t \in V_X$.
- $Pos(\diamond) = \emptyset$.
- $Pos(f(t_1, \ldots, t_n)) = E \cup \{i.q | t_i \in T_I \cup T_\diamond, q \in Pos(t_i)\}$, où $E = \{\Lambda\}$ si $t_1, \ldots, t_n \in T_I$ et $E = \emptyset$ sinon.
- $Pos(t^n.s) = \{\Lambda\} \cup \{1.q | q \in Pos(t)\} \cup \{2.q | q \in Pos(s)\}$.

■

Soient $t \in T_I \cup T_\diamond$ et $p \in Pos(t)$. La notation $t|_p$ dénote comme dans le cas usuel, le sous-terme de t apparaissant à la position p. Plus formellement nous avons :
- $t|_p = t$ si $p = \Lambda$.
- $f(t_1, \ldots, t_n)|_{i.q} = t_i|_q$ si $i \in [1..n]$ et $q \in Pos(t_i)$.
- $(t^N.v)|_{1.q} = t|_q$ si $q \in Pos(t)$.
- $(t^N.v)|_{2.q} = v|_q$ si $q \in Pos(v)$.

La proposition suivante justifie la correction de la notation précédente.

Proposition 11. *Pour chaque terme $t \in T_I \cup T_\diamond$ et pour chaque position $p \in Pos(t)$, $t|_p$ est un I-terme.* ■

Preuve. La preuve est par induction sur la longueur de p. □

La notation $t[s]_p$ est utilisée comme pour le cas usuel, pour dénoter le terme obtenu à partir de t en remplaçant le sous-terme apparaissant à la position $p \in Pos(t)$ par s. Formellement, nous avons :
- $t[s]_p = s$ si $p = \Lambda$.
- $f(t_1, \ldots, t_n)[s]_{i.q} = f(t_1, \ldots, t_{i-1}, t_i[s]_q, t_{i+1}, \ldots, t_n)$ si $i \in [1..n]$ et $q \in Pos(t_i)$.
- $(t^N.v)[s]_{1.q} = (t[s]_q)^N.v$ si $q \in Pos(t)$.
- $(t^N.v)[s]_{2.q} = t^N.(v[s]_q)$ si $q \in Pos(v)$.

Proposition 12. *Pour tout $s \in T_I$, pour tout $t \in T_I$ (resp. $t \in T_\diamond$) et pour tout $p \in Pos(t)$, nous avons $t[s]_p \in T_I$ (resp. $t[s]_p \in T_\diamond$).* ■

Preuve. La preuve est par induction sur la longueur de p. □

Comme c'est le cas pour le cas standard, quand il n'y a pas de confusion possible, la même notation $t[s]_p$ peut être utilisée pour assurer que $t|_p$ et s sont syntaxiquement égaux, c'est à dire $t|_p = s$.

Notons que si t est un terme standard, les notations précédentes coïncident avec celles usuelles.

5.1.2 Relations d'ordre et I-termes

Nous utiliserons les stratégies de restriction usuelles des règles d'inférence, en particulier les stratégies d'ordonnancement avec fonction de sélection, décrites au chapitre 7. Pour cela nous supposerons donné un ordre de réduction sur les I-termes, avec les propriétés usuelles : ordre total sur les I-termes fermés, bien fondé et fermé par substitution. Nous supposerons en outre que cet ordre est compatible avec l'opération de dépliage des contextes inductifs (voir la définition 52 ci-dessous), par exemple $f(\diamond)^{n+1}.a$ et $f(f(\diamond)^n.a)$ seront ordonnés de la même manière. Ainsi un I-terme t sera identifié à sa forme normale par dépliage $t \downarrow_I$.

Définition 52. Une relation sur les I-termes est dite *compatible avec le dépliage* si et seulement si pour tous I-termes $t, s : (t =_I s) \Rightarrow (\forall u, t \bowtie u \Rightarrow s \bowtie u)$ où $\bowtie \in \{\succ, \prec\}$. ■

Définition 53 (Ordre de réduction sur les I-termes)**.** Une relation $\succ_I$ est appelée un *ordre de réduction sur les I-termes* (ou *I-ordre*) ssi :

1. $\succ_I$ est totale sur les I-termes fermés.
2. $\succ_I$ est une relation bien fondée.
3. $\succ_I$ est fermée par substitution.
4. $\succ_I$ est compatible avec le dépliage.

■

Les propriétés 1, 2 et 3 sont standards, la propriété 4 quant à elle est propre aux I-termes et permet d'assurer la compatibilité entre la relation d'ordre et la sémantique des I-termes. Il est facile de voir que les règles ci-dessus sont consistantes, en effet, il suffit de prendre une relation d'ordre vide sauf pour les termes fermés où elle est totale.

Cet ordre est évidemment trop trivial et trop peu restrictif pour être d'une quelconque utilité pratique. Nous pouvons cependant définir des ordres plus restrictifs par exemple en étendant un ordre total sur les termes fermés. La définition suivante définit une extension d'un ordre $\succ$ vérifiant les règles $1, 2, 3$ en un ordre $\succ_I$ sur les I-termes vérifiant la règle 4.

Définition 54. Soient $\succ$ un ordre vérifiant les règles $1, 2, 3$, t, s deux I-termes normalisés par rapport au système R_I définissant la sémantique des I-termes et x une variable n'apparaissant ni dans s ni dans t. $t \succ_I s$ si et seulement si une des conditions suivantes est vérifiée :

- t et s sont des termes standards et $t \succ s$.
- $t = f(t_1, \ldots, t_n)$, $s = f(s_1, \ldots, s_n)$ et $\forall i \in [1..n], t_i \succeq s_i$ et $\exists i \in [1..n], t_i \succ s_i$.
- $t = u^N.v$ et $v \succ_I s$.
- $t = u^N.v$, $u[x]_\diamond \succ_I s$ et $v \succ s\{N \to 0\}$.
- $t = u^N.v$, $s = w^N.r$, $u[x]_\diamond \succ_I w[x]_\diamond$ et $v \succ_I w$.

■

Par exemple, nous avons $f(f(\diamond))^N.g(g(a)) \succ_I f(\diamond)^N.g(a) \succ_I a$ et $g(f(\diamond)^N.a, \diamond)^N.a \succ_I f(\diamond)^N.a$, en revanche, $g(f(\diamond)^N.x, \diamond)^N.a \not\succ_I f(\diamond)^N.x$ (en effet, il suffit de prendre $N = 0$ et $x \succ a$).

Lemme 13. *L'ordre $\succ_I$ satisfait les conditions* 1-4. ■

Preuve. La preuve est par induction sur la structure des I-termes. □

D'autres relations d'ordre plus restrictives peuvent être définies. Dans nos travaux futurs, nous envisageons d'étudier la faisabilité d'étendre les ordres réductifs usuels aux I-termes et de définir des algorithmes efficaces pour calculer ces relations d'ordre.

Dans la suite $\succ_I$ dénote un ordre de réduction sur les I-termes, que nous supposons donné (le calcul est paramétré par la donnée de cet ordre). Quand il n'y a pas de risque d'ambiguïté, $\succ_I$ sera noté tout simplement $\succ$.

L'ordre $\succ$ est étendu comme pour le cas usuel aux I-atomes (en présence d'atomes équationnels, nous utilisons l'extension multi-ensembles de l'ordre $\succ$), I-littéraux (en ignorant le symbole de négation) et aux I-clauses (par extension multi-ensemble).

Les preuves sont construites en utilisant le *calcul de résolution ordonnée* introduit dans la définition 17 pour le cas non équationnel et en utilisant le *calcul de superposition* pour le cas équationnel (voir définition 23). En plus des règles d'inférence, ces calculs utilisent (comme pour le cas standard) des règles d'élimination de la redondance pour élaguer l'espace de recherche.

Les notions de redondance et de saturation telles que définies au chapitre 2 sont directement extensibles aux I-clauses.

Pour les fonctions de sélection également définies au chapitre 2, nous supposerons qu'elles sont compatibles avec l'égalité modulo dépliage $=_I$ i.e. si C, D sont des I-clauses, L un I-littéral et si sel est une fonction de sélection, alors $C =_I D \Rightarrow sel(C) =_I sel(D)$ et $L\sigma \in sel(C\sigma) \Rightarrow L \in C$ pour toute substitution σ.

5.2 Cas non équationnel

Nous commençons notre extension du calcul de résolution ordonnée aux I-termes en considérant le cas non équationnel, i.e. le cas où le prédicat $\approx$ n'apparaît pas dans l'ensemble de clauses. Le calcul dans ce cas est une extension directe du calcul introduit au chapitre 2.

Définition 55 ($\mathcal{IOR}$)**.** Le calcul de *résolution ordonnée étendue aux I-termes* ($\mathcal{IOR}$) est défini par les deux règles d'inférence suivantes :

Résolution ordonnée

$$\frac{C \vee A \qquad D \vee \neg B}{(C \vee D)\sigma}$$

où $\sigma \in mgu(A, B)$, $A\sigma$ et $\neg B\sigma$ sont sélectionnés dans $(C \vee A)\sigma$ et $(D \vee \neg B)\sigma$ respectivement.

Factorisation ordonnée

$$\frac{C \vee A \vee B}{(C \vee B)\sigma}$$

où $\sigma \in mgu(A, B)$ et $B\sigma$ est sélectionné dans $(C \vee A \vee B)\sigma$. ■

Cette définition coïncide avec celle introduite au chapitre 2 à l'exception des conditions $\sigma = mgu(A, B)$ du cas standard, remplacées par les conditions $\sigma \in mgu(A, B)$ dans le cas de I-termes. En effet, il est bien établi que l'unificateur le plus général est unique dans le cas standard à un renommage des variables près. En revanche et comme signalé plus haut, la propriété d'unicité n'est plus vérifiée dans le cas des I-termes où il existe un nombre arbitraire (mais fini) de mgu.

Notons que $\mathcal{IOR}$ coïncide avec $\mathcal{OR}$ si les clauses considérées sont standards. Il est également établi (comme présenté au chapitre 2) que le calcul de résolution ordonné a la propriété de complétude réfutationnelle dans le cas de termes standards. Nous utilisons ce résultat pour prouver que la complétude (réfutationnelle) est également assurée pour le calcul étendu aux I-termes.

Théorème 6 (Correction de $\mathcal{IOR}$)**.** *Le calcul $\mathcal{IOR}$ est correct.* ■

Preuve. La preuve est similaire à celle du cas standard. □

La proposition suivante permet de décomposer toute substitution fermée en une composition d'une substitution $\mathbb{N}$-clôturante et d'une substitution fermée opérant exclusivement sur les variables non arithmétiques.

Proposition 14. *Soit σ une substitution fermée. Il existe une substitution $\mathbb{N}$-clôturante θ et une substitution fermée τ telles que :*

- *$dom(\tau) \cap V_N = \emptyset$: τ opère exclusivement sur des variables non arithmétiques.*
- *$\sigma = \theta\tau$: σ peut être décomposée en une partie variables arithmétiques et une partie variables standards.*

■

Preuve. Par récurrence sur le nombre de variables arithmétiques dans $dom(\sigma)$. Soit n le nombre de variables arithmétiques dans $dom(\sigma)$.

- Si $n = 0$, dans ce cas, il suffit de prendre $\theta = \emptyset$ et $\tau = \sigma$.
- Supposons que le résultat est valable jusqu'à n et montrons qu'il est valable pour $n + 1$. Soit $M \in dom(\sigma) \cap V_N$. Soit σ' la restriction de σ sur $dom(\sigma)\backslash\{M\}$. Comme $M\sigma$ ne contient pas de variables arithmétiques (puisque σ est fermée et donc $M\sigma \in \mathbb{N}$) nous avons $\sigma = \{M \to M\sigma\}\sigma'$. Par hypothèse de récurrence, il existe θ' $\mathbb{N}$-clôturante et τ' telles que $\sigma' = \theta'\tau'$ et donc, $\sigma = \{M \to M\sigma\}\theta'\tau'$. Le résultat est établi en prenant $\theta = \{M \to M\sigma\}\theta'$ et $\tau = \tau'$.

□

Soit S un ensemble de (I-)clauses. $Inst(S)$ désigne l'ensemble des instances fermées des clauses de S et $NInst(S)$ l'ensemble de toutes les $\mathbb{N}$-instances des clauses de S.

Proposition 15. *Soient S et S' deux ensembles de (I-)clauses. Si $Inst(S) = Inst(S')$ alors toute clause redondante dans S est également redondante dans S'.* ■

Preuve. Soient $\prec$ un ordre sur les I-clauses et C une clause redondante par rapport à S. Par définition, il existe n clauses $D_1, \dots D_n \in S$ et n substitutions fermées $\theta_1, \dots, \theta_n$ telles que pour chaque instance fermée de C, $C\sigma$ nous avons $D_i\theta_i \prec C\sigma$ pour chaque i et $\{D_i\theta_i \mid i \in [1..n]\} \models C\sigma$. Par hypothèse $Inst(S) = Inst(S')$ et donc $\forall i, D_i\theta_i \in Inst(S')$. Ainsi, C est redondante par rapport à S'. □

Proposition 16. *Pour tout ensemble S de (I-)clauses, $Inst(S) = Inst(NInst(S))$.* ■

Preuve. $Inst(NInst(S)) \subseteq Inst(S)$. Soit $C \in Inst(NInst(S))$. Par définition, il existe une substitution σ fermée, une substitution $\mathbb{N}$-clôturante θ et une clause $D \in S$ telles que $C = D\theta\sigma$ et donc $C \in Inst(S)$.

$Inst(S) \subseteq Inst(NInst(S))$. Soit $C \in Inst(S)$. Il existe une substitution fermée σ et une clause $D \in S$ telles que $C = D\sigma$. D'après la proposition 14, il existe une substitution $\mathbb{N}$-clôturante θ et une substitution τ telles que $dom(\tau) \cap V_N = \emptyset$ et $\sigma = \theta\tau$. Nous avons donc $C = D\theta\tau$. Or comme θ est $\mathbb{N}$-clôturante, alors $D\theta \in NInst(S)$. Ainsi, $D\theta\sigma \in Inst(NInst(S))$. □

Le lemme suivant lie la saturation d'un ensemble de I-clauses à celle de l'ensemble de ses $\mathbb{N}$-instances (cet ensemble ne contient que des clauses standards).

Lemme 17. *Soit S_I un ensemble de I-clauses et soit S l'ensemble de* toutes *les $\mathbb{N}$-instances de S_I ($S = NInst(S_I)$). Si S_I est saturé sous $\mathcal{IOR}$ alors S est saturé sous $\mathcal{IOR}$.* ■

Preuve. Supposons que S_I est saturée sous $\mathcal{IOR}$. Soit C une clause dérivée de S (par $\mathcal{IOR}$). Nous montrons que C est redondante par rapport à S. Notons que par définition, la fonction de sélection *sel* est fermée par substitution et compatible avec l'égalité modulo dépliage $=_I$. Nous distinguons deux cas :

- Si C est dérivée à partir de deux clauses $D = D' \vee A$ et $E = E' \vee \neg B$ par résolution ordonnée où $D'\sigma$ et $E'\sigma$ sont sélectionnées dans $D\sigma$ et $E\sigma$ respectivement, alors $C = (D' \vee E')\sigma$ où $\sigma \in mgu(A, B)$. Par définition, il existe deux I-clauses D_I et E_I dans S_I telles que D et E sont des instances de D_I et E_I respectivement. D_I et E_I sont de la forme $D'_I \vee A_I$ et $E'_I \vee \neg B_I$ et deux substitutions $\mathbb{N}$-clôturantes θ_D et θ_E telles que $D'_I\theta_D =_I D'$, $E'_I\theta_E =_I E'$, $A_I\theta_D = A$ et $B_I\theta_E = B$.
  Nous montrons que la résolution ordonnée est possible entre D_I et E_I en A_I et $\neg B_I$.
  - A_I et B_I sont unifiables : Soit la substitution $\theta = \theta_D\theta_E$. Comme par définition D_I et E_I n'ont pas de variables communes, nous avons $A_I\theta =_I A$ et $B_I\theta =_I B$. Ceci implique que A_I et B_I sont unifiables puisque $A_I\theta\sigma = A\sigma = B\sigma = B_I\theta\sigma$ et il existe donc $\sigma_I \in mgu(A_I, B_I)$ et une substitution τ tels que $\theta\sigma = \sigma_I\tau$.

– $A_I\sigma_I$ et $\neg B_I\sigma_I$ sont sélectionnés dans $D_I\sigma_I$ et $E_I\sigma_I$ respectivement : Comme la fonction de sélection *sel* est fermée par substitution et compatible avec l'égalité modulo dépliage $=_I$ et comme $A\sigma \in sel(D\sigma)$ et $\neg B\sigma \in sel(E\sigma)$, nous avons $A_I\sigma_I \in sel(D_I\sigma_I)$ et $B_I\sigma_I \in sel(D_I\sigma_I)$.

La résolution ordonnée peut donc s'appliquer entre D_I et E_I en A_I et $\neg B_I$ générant comme résultat une clause de la forme $C_I = (D'_I \vee E'_I)\sigma_I$. De plus, nous avons $C_I\tau = C$. Comme S_I est supposé saturé par hypothèse, C_I est redondante dans S_I (par rapport à $\mathcal{IOR}$). Donc C est également redondante puisque toute instance fermée de C est aussi instance fermée de C_I. Or comme $S = NInst(S_I)$ nous avons donc $Inst(S_I) = Inst(S)$ par la proposition 16 et donc C est redondante dans S par la proposition 15.

– Si C est dérivée par factorisation à partir d'une clause $D = Q \vee A \vee B$ où $B\sigma$ est sélectionné dans $(Q \vee A \vee B)\sigma$, alors $C = (Q \vee B)\sigma$ où $\sigma \in mgu(A, B)$. Par définition, il existe une clause $D_I \in S_I$ telle que D est une ($\mathbb{N}$)-instance de D_I. D_I est de la forme $D_I = Q_I \vee A_I \vee B_I$ et il existe une substitution $\mathbb{N}$-clôturante θ telle que $Q_I\theta = Q$, $A_I\theta = A$ et $B_I\theta = B$.

Nous montrons que la factorisation est possible sur D_I.

– A_I et B_I sont unifiables : Nous avons $A_I\theta =_I A$ et $B_I\theta =_I B$. Ceci implique que A_I et B_I sont unifiables puisque $A_I\theta\sigma = A\sigma = B\sigma = B_I\theta\sigma$ et il existe donc $\sigma_I \in mgu(A_I, B_I)$ et une substitution τ tels que $\theta\sigma = \sigma_I\tau$.

– $B_I\sigma_I$ est sélectionné dans $D_I\sigma_I$: Comme la fonction de sélection *sel* est fermée par substitution et compatible avec l'égalité modulo dépliage $=_I$ et comme $B\sigma \in sel(D\sigma)$, nous avons $B_I\sigma_I \in sel(D_I\sigma_I)$.

La factorisation est donc applicable sur D_I et génère une clause de la forme $C_I = (Q_I \vee B_I)\sigma_I$ avec $C_I\tau = C$. Or comme S_I est supposé saturé par hypothèse, C_I est redondante dans S_I (par rapport à $\mathcal{IOR}$). Donc C est également redondante puisque toute instance fermée de C est aussi instance fermée de C_I. Et comme $S = NInst(S_I)$ nous avons donc $Inst(S_I) = Inst(S)$ par la proposition 16 et finalement C est redondante dans S par la proposition 15.

□

Le théorème 7 établit la complétude réfutationnelle du calcul $\mathcal{IOR}$.

Théorème 7 (Complétude réfutationnelle de $\mathcal{IOR}$)**.** *Tout ensemble insatisfaisable de I-clauses qui est saturé par rapport à $\mathcal{IOR}$ contient la clause vide.* ■

Preuve. Soit S_I un ensemble insatisfaisable et saturé par rapport au calcul $\mathcal{IOR}$. L'ensemble S de toutes les $\mathbb{N}$-instances de S_I est également insatisfaisable. Par le lemme 17, S (contenant uniquement des clauses standards) est saturé. Or comme la restriction du calcul $\mathcal{IOR}$ coïncide avec $\mathcal{OR}$ en cas de clauses standards et comme le calcul $\mathcal{OR}$ a la propriété de complétude réfutationnelle, S contient la clause vide $\square$. Par conséquent, S_I contient également la clause vide puisqu'aucune autre I-clause (à part $\square$) n'a pour instance $\square$. □

5.3 Cas équationnel

5.3.1 Superposition et I-termes

L'objectif de cette section est d'étendre le calcul de superposition introduit dans la définition 23 aux I-clauses. Comme pour le cas usuel, nous pouvons supposer que l'égalité est l'unique prédicat considéré (les atomes de la forme $P(t_1, \ldots, t_n)$ peuvent être codés par $P(t_1, \ldots, t_n) \approx true$).

Définition 56 (Le calcul de superposition $\mathcal{ISC}$)**.** Le calcul de *superposition étendue aux I-termes* ($\mathcal{ISC}$) est défini par les règles d'inférence suivantes :

Superposition (S)

$$\frac{Q \vee (l \approx r) \qquad D \vee (w[t]_p \bowtie u)}{(Q \vee D \vee (w[r]_p \bowtie u))\sigma}$$

où $\bowtie \in \{\approx, \not\approx\}$, $\sigma \in mgu(l,t)$, $l\sigma \not\preceq r\sigma$, $w\sigma \not\preceq u\sigma$, $(l \approx r)\sigma$, $(w \bowtie u)\sigma$ sont sélectionnés dans $(Q \vee l \approx r)\sigma$ et $(D \vee w \bowtie u)\sigma$ respectivement, et t n'est pas une variable.

Résolution équationnelle (ER)

$$\frac{Q \vee (l \not\approx r)}{Q\sigma}$$

où $\sigma \in mgu(l,r)$ et $(l \not\approx r)\sigma$ est sélectionné dans $(Q \vee l \not\approx r)\sigma$.

Factorisation équationnelle (EF)

$$\frac{Q \vee (t \approx s) \vee (l \approx r)}{(Q \vee (s \not\approx r) \vee (l \approx r))\sigma}$$

où $\sigma \in mgu(l,t)$, $l\sigma \not\preceq r\sigma$, $t\sigma \not\preceq s\sigma$ et $(t \approx s)\sigma$ est sélectionné dans $(Q \vee l \approx r \vee t \approx s)\sigma$. ■

Comme pour le cas de la résolution ordonnée étendue aux I-termes, la seule différence avec le calcul de superposition appliqué aux clauses standards réside dans le fait que l'unificateur le plus général n'est plus forcément unique. Ce calcul coïncide bien entendu avec le calcul standard en l'absence de N-clauses.

5.3.2 Incomplétude réfutationnelle de $\mathcal{ISC}$

Le calcul de superposition $\mathcal{ISC}$ n'est pas complet (au sens de la complétude réfutationnelle). L'exemple suivant en explique les raisons :

Exemple 24.

$$\begin{cases} (1) & P(f(a,\diamond)^N.b) \\ (2) & \neg P(c) \\ (3) & f(a,b) \approx d \\ (4) & f(a,d) \approx c \end{cases}$$

■

Cet ensemble d'I-clauses est clairement insatisfaisable. En effet, en instanciant N par 2 nous obtenons $f(a, f(a,b))$ puis en remplaçant successivement $f(a,b)$ par d et $f(a,d)$ par c nous arrivons à la contradiction $P(c)$ et $\neg P(c)$. En revanche, la clause vide ne peut pas être dérivée en utilisant les règles de $\mathcal{SC}$. Une seule unification est possible : $f(a,\diamond)^N.b$ et $f(a,b)$ avec la substitution $\sigma = \{N \to 1\}$. La règle de superposition génère la clause $P(d)$ à partir de (1) et (3). d n'est unifiable ni avec $f(a,b)$ ni avec $f(a,d)$. Aucune superposition n'est alors possible et le processus de recherche de preuve termine sans dériver la clause vide. L'ensemble saturé S est obtenu :

$$S = \begin{cases} (1) & P(f(a,\diamond)^N.b) & \\ (2) & \neg P(c) & \\ (3) & f(a,b) \approx d & \\ (4) & f(a,d) \approx c & \\ (5) & P(d) & (S(1,3), \quad \sigma = \{N \to 1\}) \end{cases}$$

5.3.3 H-superposition

Le problème mis en évidence dans l'exemple 24 est qu'en unifiant $f(a,\diamond)^M.b$ avec son instance $f(a,b)$, la structure en N-terme est « perdue ». En effet, $f(a,\diamond)^M.b$ représente l'ensemble infini $\{b, f(a,b), f(a,f(a,b)), f(a,f(a,f(a,b))), \ldots\}$. Or quand nous appliquons la superposition entre (1) et (3) après avoir unifié $f(a,\diamond)^M.b$ avec $f(a,b)$ qui en est une instance, nous récupérons une clause correspondant à cette seule instance ($P(c)$) et nous ignorons toutes les autres applications possibles de la superposition dans le cas des autres instances. En particulier, dans cette exemple, l'unification empêche de faire la superposition entre par exemple $P(f(a,f(a,b)))$ et $P(f(a,b))$ qui engendrerait la clause $P(f(a,d))$ nécessaire pour dériver la clause vide. Cette remarque est également vraie pour les autres instances correspondant à $M \geq 2$. Pour tenir compte de toutes ces autres instances, il est nécessaire de garder la structure en N-terme et l'idée donc est de changer la règle de superposition pour préserver cette structure. Une analyse approfondie révèle que le problème survient uniquement quand la superposition est appliquée (comme pour l'exemple précédent) à l'intérieur d'un sous-terme apparaissant le long du chemin menant au trou d'un N-terme (par exemple dans le terme de base).

Dans le nouveau calcul que nous proposons, les règles de résolution et de factorisation équationnelles sont conservés sans changement ainsi que la superposition impliquant des termes n'apparaissant pas le long d'un chemin menant à un trou. Nous introduisons une règle spécifique pour traiter ce dernier cas.

Définir un calcul de superposition complet pour des clauses contenant des I-termes n'est pas une tâche simple. La raison est qu'une I-clause dénote en réalité une suite infinie de clauses standards, avec un nombre non borné de positions distinctes. Par exemple, les $\mathbb{N}$-instances de $P(f(\diamond)^N.a)$ contiennent les positions $\Lambda, 1, 1.1, 1.1.1, \ldots, 1.\underbrace{1.\ldots.1}_{N \text{ fois}}\ldots$ alors que $Pos(P(f(\diamond)^N.a))$ contient uniquement $\{\Lambda, 1, 1.2\}$. En principe, la règle de superposition devrait être appliquée à chacune de ces positions ce qui risque de rendre le calcul non finitaire dans le sens où il est possible de déduire par une seule application des règles d'inférence une infinité de I-clauses. Un calcul de ce type est très difficile à utiliser en pratique. Il y a une différence majeure avec le cas usuel (clauses standards) : même si une clause du premier ordre peut également représenter une infinité de clauses fermées avec un nombre non borné de positions, il est bien connu que l'application de la superposition « sous » les variables est inutile et donc il est possible de se restreindre aux positions apparaissant dans les termes du premier ordre (ne correspondant pas bien entendu à une variable). Ce n'est pas le cas ici et l'exemple 24 illustre que la restriction à des positions apparaissant dans les I-termes n'est pas suffisant.

Nous présentons dans ce qui suit comment surmonter ce problème. L'idée est que grâce au pouvoir d'expression des I-termes, toutes les conséquences (de l'application de la règle de superposition) peuvent en fait être codées par une seule I-clause. Pour cela nous avons besoin d'une nouvelle règle d'inférence appelée H-superposition (H comme « hole »). La règle de H-superposition est définie comme suit :

$$\frac{Q \vee (l \approx r) \qquad D \vee (w[t[\diamond]^M_p.s]_o \bowtie u)}{(Q \vee D \vee (w[t^N.t[r]_q]_o \bowtie u))\sigma_M\sigma}$$

où $\bowtie \in \{\approx, \not\approx\}$, q est un préfixe non vide de p (avec potentiellement $p = q$), $o \in Pos(w)$ et :

- N, N' sont deux variables arithmétiques distinctes n'apparaissant pas dans les prémisses.
- $\sigma_M = \{M \to N + N' + 1\}$.
- $\sigma \in mgu(t[t^{N'}.s]_\diamond|_q\sigma_M, l)$.

Puisque nous ne considérons que des exposants variables (et non pas des formules arithmétiques quelconques), les termes de la forme $t^{N+N'+1}.s$ sont systématiquement transformés en utilisant les règles définissant la sémantique des I-termes.

Dans la suite, nous dénotons par $\mathcal{IS}_H$ le calcul défini par les règles d'inférences ci-dessus : H-superposition, superposition et résolution et factorisation équationnelles.

5.3.4 Exemples

Avant de donner une preuve de la complétude réfutationnelle du calcul $\mathcal{IS}_H$, nous donnons quelques exemples. Pour plus de clarté, nous distinguons deux cas :

– Supposons en premier lieu que $q = p$. Dans ce cas, le N-terme $t^M.s$ est unifié avec un I-terme l et la substitution solution instancie M par un nombre entier $m \in \mathbb{N}$. $t^m.s$ est unifié avec l et peut donc être remplacé par un terme r dans la partie droite de l'équation. D'un autre côté, si nous considérons l'ensemble des $\mathbb{N}$-instances de $t^M.s$, le remplacement de $t^m.s$ ne doit pas se faire uniquement au niveau de la racine de $t^M.s$. Pour assurer la complétude (réfutationnelle), le remplacement doit aussi se faire dans les termes $t[t^m[s]_p]_p$, $t[t[t^m[s]_p]_p]_p$ et ainsi de suite *pour toutes les valeurs possibles* de M inférieures ou égales à m. La superposition standard ignore toutes les unifications possibles à l'exception du cas où M est instanciée par m unifiant $t^m.s$ et l. La règle de H-superposition permet justement de contourner ce problème en considérant toutes ces possibilités : $t^m.s$ est unifiée avec l et la structure en N-terme est préservée en introduisant une nouvelle variable N qui encode le nombre de dépliages du terme $t^M.s$ avant d'atteindre la position à laquelle le remplacement est réalisé. De manière intuitive, M est décomposée en deux variables entières : $M = N + 1 + N'$. Le terme $t[\diamond]_p^M.s$ peut alors être représenté comme $t[\diamond]_p^N.t^1.t[\diamond]_p^{N'}.s$. Le nouveau terme de base $t[\diamond]_p^{N'}.s$ est unifié avec l et le nouveau contexte itératif ayant N comme exposant permet de garder la structure en N-terme. En revenant à l'exemple insatisfaisable précédent, nous montrons comment dériver la clause vide en utilisant la règle de H-superposition :

(1)	$P(f(a,\diamond)^M.b) \approx true$		
(2)	$P(c) \not\approx true$		
(3)	$f(a,b) \approx d$		
(4)	$f(a,d) \approx c$		
(5)	$P(f(a,\diamond)^N.f(a,d)) \approx true$	$HS\,(3),(1)$	$\sigma=\{M\rightarrow N+N'+1\}\{N'\rightarrow 0\}$
(6)	$P(c) \approx true$	$S\,(4),(5)$	$\sigma=\{N\rightarrow 0\}$
(7)	$true \not\approx true$	$S\,(6),(2)$	
(8)	$\square$	$EIR\,(7)$	

– si $p \neq q$ alors l est unifié avec un sous-terme de $t[\diamond]_p^M.s$ apparaissant le long du chemin inductif. Dans ce cas, aucune unification n'est possible sans dépliage. Comme pour le cas précédent, considérer l'unification directement après dépliage sans décomposer la variable M entraîne la perte de la structure en N-terme. Il y a cependant une différence, à savoir que le sous-terme concerné par le remplacement peut apparaître à n'importe quelle position le long du chemin menant au trou. C'est pourquoi nous considérons dans notre règle tous les préfixes non vides de la position du trou. Notons qu'il est inutile de traiter les cas où le préfixe q est vide puisque ce cas coïncide avec le résultat de l'application de la superposition en s (si s et l sont unifiables) tout en conservant le contexte inductif. La structure en N-terme est alors préservée. L'exemple suivant illustre bien ces propos :

$$\begin{cases} (1) & P(f(a,g(\diamond))^M.b) \approx true \\ (2) & P(c) \not\approx true \\ (3) & f(a,e) \approx d \\ (4) & f(a,g(d)) \approx c \\ (5) & g(b) = e \end{cases}$$

Cet exemple est insatisfaisable et comme pour le cas précédent, aucune règle de superposition (standard) ne peut être appliquée puisqu'il n'y a pas d'unification possible entre deux termes apparaissant en partie gauche des (in)équations. L'usage de la règle de H-superposition règle le problème comme l'illustre la dérivation suivante :

(6)	$P(f(a,g(\diamond))^N.f(a,e)) = true$	$HS\,(5),(1)$	$\sigma=\{M\to N+N'+1\}\{N'\to 0\}$
(7)	$P(f(a,g(\diamond))^N.d) = true$	$S\,(3),(6)$	
(8)	$P(c) = true$	$S\,(4),(7)$	$\sigma=\{N\to 1\}$
(9)	$true \not\approx true$	$S\,(8),(2)$	
(10)	$\square$	$ER\,(9)$	

5.3.5 Correction et complétude réfutationnelle

Nous commençons par établir la correction du calcul $\mathcal{IS}_H$. La preuve est triviale et similaire au cas standard et la seule différence réside dans la présence de la nouvelle règle de H-superposition pour laquelle il faut établir la correction.

Théorème 8 (Correction de $\mathcal{IS}_H$)**.** *Le calcul $\mathcal{IS}_H$ est correct.* ■

Preuve. Il s'agit de montrer que toutes les règles sons correctes.

- Les règles de superposition, de résolution équationnelle et de factorisation équationnelle sont correctes. La preuve est identique au cas standard.
- La règle de H-superposition est correcte. Soient $Q' = Q \vee (l \approx r)$ et $D' = D \vee (w[t[\diamond]_p^M.s]_o \bowtie u)$ deux (I)-clauses et soit $C = (Q \vee D \vee (w[t^N.t[r]_q]_o \bowtie u))\sigma_M\sigma$ la clause engendrée par H-superposition entre Q' et D' où $\bowtie \in \{\approx, \not\approx\}$, q est un préfixe non vide de p (avec potentiellement $p = q$), $o \in Pos(w)$, N, N' sont deux variables arithmétiques distinctes n'apparaissant pas dans les prémisses, $\sigma_M = \{M \to N + N' + 1\}$ et $\sigma \in mgu(t[t^{N'}.s]_\diamond|_q\sigma_M, l)$. Soit $\mathcal{I}$ un modèle de Q' et de D' (i.e. $\mathcal{I} \models Q', D'$).
  Soient $Vars(C) = \{x_1, \ldots, x_n\}$ les variables libres de C et soient $v_1, \ldots, v_n$ des éléments (valeurs) du domaine de $\mathcal{I}$. Nous montrons que $\mathcal{J}(C) = vrai$ où $\mathcal{J} = \mathcal{I}[x_1 \to v_1, \ldots, x_n \to v_n]$ (les clauses sont quantifiées universellement par définition).
  - Si $\mathcal{J}(Q) = vrai$ ou $\mathcal{J}(D) = vrai$ alors $\mathcal{J}(C) = vrai$.
  - Sinon $\mathcal{J}(l \approx r) = vrai$ et $\mathcal{J}\left(w[t[\diamond]_p^M.s]_o \bowtie u\right) = vrai$ et donc $\mathcal{J}\left(\left(w[t[\diamond]_p^M.s]_o \bowtie u\right)\sigma_M\right) = vrai$. Par décomposition nous avons donc $\mathcal{J}\left(\left(w[t^N.t[v]_q]_o \bowtie u\right)\sigma_M\right) = vrai$ où $v = t[t^{N'}.s]_\diamond|_q\sigma_M$ or $\mathcal{J}(l \approx r) = vrai$ et comme $\sigma \in mgu(t[t^{N'}.s]_\diamond|_q\sigma_M, l)$ alors $\mathcal{J}(l\sigma \approx r\sigma) = vrai$ et $\mathcal{J}(v\sigma \approx r\sigma) = vrai$ (par définition). D'autre part comme $\mathcal{J}\left(\left(w[t^N.t[v]_q]_o \bowtie u\right)\sigma_M\right) = vrai$ alors $\mathcal{J}\left(\left(w[t^N.t[v]_q]_o \bowtie u\right)\sigma_M\sigma\right) = vrai$ et donc $\mathcal{J}\left(\left(w[t^N.t[r]_q]_o \bowtie u\right)\sigma_M\sigma\right) = vrai$.

□

Nous présentons maintenant une preuve de la complétude réfutationnelle du calcul $\mathcal{IS}_H$. La démonstration utilise la notion de redondance et il est très difficile de pouvoir s'en passer pour établir la complétude. En effet, si nous considérons l'ensemble $S_I = \{p(f(a,\diamond)^n.c), a \approx b\}$ alors l'utilisation de $\mathcal{SC}$ sur S_I permet de générer la clause $C = p(f(b, f(a, c)))$ par exemple alors que le calcul $\mathcal{IS}$ ne permet de générer que la I-clause $p(f(b,\diamond)^n.c)$ dont C n'est pas une instance. Ceci implique que les arguments invoqués dans le lemme 17 ne peuvent plus être utilisés directement. C est cependant redondante par rapport à l'ensemble $\{p(f(b,\diamond)^n.c), a \approx b\}$ et peut donc être ignorée en appliquant l'élimination de la redondance.

Tout d'abord, nous commençons par prouver un lemme essentiel qui relie les sous-termes apparaissant dans un I-terme et ceux apparaissant dans une de ses instances fermées.

Lemme 18. *Soit t_I un I-terme et soit $t = t_I\sigma\downarrow$ une instance fermée de t. Si s est un sous-terme de t alors une des conditions suivantes s'applique :*

1. *t_I contient une variable x comme sous-terme et s est un sous-terme de $x\sigma\downarrow$.*
2. *t_I contient un sous-terme s_I tel que s_I n'est pas une variable et $s_I\sigma\downarrow = s$.*
3. *t_I contient un N-terme $u^N.v$ et il existe $m < N\sigma$ tel que s apparaît dans $u[u^m.v]_\diamond\sigma\downarrow$ à une position ne correspondant pas à un préfixe vide de la position du trou dans u.*

■

Preuve. La preuve est par induction sur le terme t_I. Par définition, si $s = t$ alors si t_I est une variable, la condition 1 est vérifiée. Sinon c'est la condition 2 qui est vérifiée.

Si t_I est une variable x alors la condition 1 est vérifiée.

Si t_I est de la forme $f(t_I^1, \ldots, t_I^n)$ alors s doit apparaître dans un terme $t_I^i\sigma\downarrow$ pour $i \in [1..n]$. Par hypothèse d'induction, une des conditions 1, 2 ou 3 doit se vérifier pour t_I^i, et donc également pour t_I puisque t_I contient t_I^i.

Si t_I est de la forme $u^N.v$ et s apparaît dans $v\sigma\downarrow$ alors en appliquant l'hypothèse d'induction sur v une des conditions 1, 2 ou 3 s'applique pour v et donc pour t_I puisque t_I contient v.

Si t_I est de la forme $u^N.v$ et s apparaît dans $u\sigma\downarrow$ alors s apparaît forcément dans un sous-terme stricte r de $u\sigma\downarrow$(puisque u est un terme avec un trou de la forme $f(t_I^1, \ldots, t_I^n)$). En appliquant alors l'hypothèse d'induction sur r une des conditions 1, 2 ou 3 s'applique pour r et donc pour t_I puisque t_I contient r.

Si $t_I = u^N.v$ et s n'apparaît ni dans $u\sigma\downarrow$ ni dans $v\sigma\downarrow$, alors s doit apparaître dans $t_I\sigma\downarrow$ le long de la position $p^{N\sigma}$, où p est la position du trou $\diamond$ dans u. Remarquons que le long de la position du trou il ne peut exister aucune variable. Si la position de s dans $t_I\sigma\downarrow$ est vide, alors la condition 2 est vérifiée par définition puisque $s = t_I\sigma\downarrow$. Sinon, s apparaît à une position de la forme $p^k.q$ où q est un préfixe non vide de p et $k < N\sigma$. Dans ce cas, s apparaît dans $u[u^{N\sigma-1-k}.v]_\diamond\sigma\downarrow$ et donc la condition 3 est vérifiée. □

La définition suivante formalise une relation de remplacement entre les termes. Étant donnés deux termes s et u, cette relation définit le remplacement de certaines occurrences de s par u dans n'importe quel terme t contenant s.

Définition 57. Soient deux termes s, u. La relation de *remplacement de plusieurs occurrences de s par u* notée$\rightarrow_{u/s}$ est la plus petite relation réflexive et transitive telle que $t[s]_p \rightarrow_{u/s} t[u]_p$, pour toute position p et pour tout terme t. ■

Pour tous termes t, t', $t' \rightarrow_{u/s} t$ signifie que t peut être obtenu à partir de t' en remplaçant quelques occurrences de s dans t' par u. La réflexivité est utilisée pour gérer le cas où les deux termes t et t' sont identiques : dans ce cas aucun remplacement n'est nécessaire. Cette notation peut bien évidemment être étendue aux clauses de manière directe. Pour deux clauses C, C', $C' \rightarrow_{u/s} C$ signifie que pour tous termes t, t' de C, C' respectivement, $t' \rightarrow_{u/s} t$.

Lemme 19. *Soient t_I, s_I deux I-termes et soit $p \in Pos(t_I)$. Soit σ une substitution fermée. Soient $t'_I = t_I[s_I]_p$, $t = t_I\sigma\downarrow$, $s = s_I\sigma\downarrow$ et $t' = t'_I\sigma\downarrow$. Soit u_I le sous-terme apparaissant à la position p dans t_I et soit $u = u_I\sigma\downarrow$. Nous avons alors $t' \rightarrow_{u/s} t$.* ■

Preuve. La preuve est par induction sur t_I. Il faut distinguer plusieurs cas selon la position p et la forme du terme t_I :

- Si $p = \Lambda$, nous avons par définition $t_I = u_I$, $t = u$ et $t' = s$. La preuve est donc immédiate.
- Si $p = i.q$ et t_I est de la forme $f(t_I^1, \ldots, t_I^n)$, alors $t'_I = f(t_I^1, \ldots, t_I^i[s_I]_q, \ldots, t_I^n)$. Par hypothèse d'induction, $t_I^i[s_I]_q\sigma\downarrow \rightarrow_{u/s} t_I^i\sigma\downarrow$, par conséquent, la même chose est valable pour $t = f(t_I^1\sigma\downarrow, \ldots, t_I^n\sigma\downarrow)$ et $t' = t'_I = f(t_I^1\sigma\downarrow, \ldots, t_I^i[s_I]_q\sigma\downarrow, \ldots, t_I^n\sigma\downarrow)$.

– Si t_I est de la forme $v^N.w$ (naturellement nous avons $p = i.q$). Si $p = 2.q$ où q est une position dans w alors la preuve est similaire au cas précédent. Sinon, $p = 1.q$ où q est une position dans v (à noter que par définition de $Pos(t_I)$, $v|_q$ ne peut pas contenir le trou). Soit $r = v[s_I]_q$. Nous avons $t'_I = r^N.w$, $t_I\sigma \downarrow = v[\ldots[v[w\sigma\downarrow]_\diamond]\ldots]_\diamond$ et $t'_I\sigma\downarrow = r[\ldots[r[w\sigma\downarrow]_\diamond]\ldots]_\diamond$. Or par hypothèse d'induction (en l'appliquant à un sous-terme de v contenant le terme à la position q mais pas le trou), $v \rightarrow_{u/s} r$. En répétant ces remplacements pour chaque occurrence de v et de r, nous avons le même résultat pour t et t'.

□

Lemme 20. *Soient deux clauses fermées C, C' et deux termes fermés l, r tels que $l \succ r$. Si $C \rightarrow_{l/r} C'$ alors $C \preceq C'$.* ■

Preuve. La preuve est par induction sur le nombre de remplacements en plus du fait que la relation d'ordre $\succ$ est un ordre de réduction stable avec la mise en contexte, i.e ; si $l \succ r$ alors pour tout terme w et toute position $p \in Pos(w)$ nous avons $w[l]_p \succ w[r]_p$. □

Comme pour le cas non équationnel, nous avons besoin d'établir un lien entre la saturation d'un ensemble de I-clauses et celle de l'ensemble de ses instances fermées (qui ne contient que des clauses standards fermées). Ce lien est établi par le lemme 22. La preuve de ce lemme utilise les résultats suivants :

Lemme 21. *Soient Q une disjonction de littéraux (fermés), C, C' deux clauses fermées et s, u deux termes fermés tels que $u \succ s$, si $C \rightarrow_{u/s} C'$ alors $Q \vee C'$ est redondante par rapport à $\{Q \vee C, Q \vee u \approx s\}$.* ■

Preuve. Le cas où $C = C'$ est trivial (aucun remplacement n'est fait). Nous considérons donc uniquement le cas $C' \neq C$ (i.e. au moins un remplacement a été effectué). Par conséquent, s est un sous-terme de C et u est un sous-terme de C'.

– Nous montrons d'abord que $(Q \vee C), (Q \vee u \approx s) \models (Q \vee C')$. Soit $\mathcal{I}$ un modèle de $(Q \vee C), (Q \vee u \approx s)$.
  – Si $\mathcal{I} \models Q$ alors $\mathcal{I} \models (Q \vee C')$.
  – Si $\mathcal{I} \not\models Q$ alors $\mathcal{I} \models C$ et $\mathcal{I} \models (u \approx s)$ Ainsi $\mathcal{I} \models C'$ (puisque le prédicat de l'égalité définit une congruence) et donc $\mathcal{I} \models C'$ (puisque C' est fermé).

Donc $(Q \vee C), (Q \vee u \approx s) \models (Q \vee C')$.

– Nous montrons ensuite que $(Q \vee C) \prec (Q \vee C')$. Il suffit de montrer que $C \prec C'$. C'est une application du lemme 20.

□

Lemme 22. *Soit S_I un ensemble d'I-clauses et soit S l'ensemble de toutes les instances fermées de S_I. Si S_I est saturé sous $\mathcal{IS}_H$ alors S est saturé sous $\mathcal{SC}$.* ■

Preuve. Supposons que S_I est saturé sous $\mathcal{IS}_H$. Soit C une clause dérivée de S par $\mathcal{SC}$. Nous montrons que C est redondante par rapport à S. Il faut distinguer tous les cas possibles :

– C est dérivée par superposition. Il existe alors deux clauses (fermées) $Q = Q' \vee (l \approx r)$, $D = D' \vee (w[t]_q \bowtie u)$ telles que $C = Q' \vee D' \vee (w[r]_q \bowtie u)$ avec $l \succ r$ et $l = t$. Par définition, il existe deux I-clauses Q_I et D_I dans S_I telles que Q et D sont des instances fermées de Q_I et D_I respectivement. Il existe donc quatre I-termes l_I, r_I, w_I, u_I, deux I-clauses Q'_I et D'_I et deux substitutions fermées θ_Q, θ_D tels que $l_I\theta_Q =_I l$, $r_I\theta_Q =_I r$, $Q'_I\theta_Q = Q'$, $w_I\theta_D =_I w[t]_q$, $u_I\theta_D =_I u$.
  Nous dénotons par θ la substitution $\theta_D\theta_Q$. Nous distinguons trois cas :

  1. w_I contient une variable x comme sous-terme à une position $p \in Pos(w_I)$ et t est un sous-terme de $x\theta\downarrow$, i.e. $(x\theta\downarrow)|_{p'} = t$. Nous montrons que C est redondante par rapport

à S. Nous avons $p.p' = q$. Soit σ une substitution fermée telle que $x\sigma = (x\theta\downarrow)[r]_{p'}$ et coïncidant avec θ sur $V_X \backslash \{x\}$, i.e. $\forall y \in V_X, (y \neq x) \Rightarrow y\sigma = y\theta$.

- Si D_I contient une seule occurrence de la variable x alors $D_I\sigma$ subsume C ($D_I\sigma$ est une sous-clause de C) et donc C est redondante par rapport à S.
- Si D_I contient plusieurs occurrences de x nous montrons alors que C est redondante par rapport à $\{D_I\sigma, Q'_I\theta \vee l \approx r\}$. Soit $\mathcal{I}$ un modèle de $\{D_I\sigma, Q'_I\theta \vee l \approx r\}$. Si $\mathcal{I} \models Q'_I\theta$ alors $\mathcal{I} \models C$ ($Q'_I\theta$ est une sous-clause de C). Sinon $\mathcal{I} \models l \approx r$ et $\mathcal{I} \models D_I\sigma$. Par définition de σ la sous-clause $D' \vee (w[r]_q \bowtie u)$ de C peut être obtenue en remplaçant des occurrences de r par l dans $D_I\sigma$ (i.e. $D_I\sigma \rightarrow_{l/r} D' \vee (w[r]_q \bowtie u)$). Or comme $l \succ r$ alors d'après le lemme 21 nous avons $D' \vee (w[r]_q \bowtie u)$ est redondante par rapport à $\{D_I\sigma, l \approx r\}$ (en choisissant Q suivant les notations du lemme égale à la clause vide) et donc $\mathcal{I} \models D' \vee (w[r]_q \bowtie u) \models C$. Donc pour tout modèle $\mathcal{I} \models \{D_I\sigma, Q'_I\theta \vee l \approx r\}$, $\mathcal{I} \models C$. D'autre part comme C est le résultat de la superposition entre D et Q nous avons alors $l \succ r$, l est maximal dans Q et w maximal dans D. Puisque D_I contient plusieurs occurrences de x alors C contient plusieurs occurrences de l (au moins deux puisque $(x\theta\downarrow)|_{p'} = t = l$) et donc l est un sous-terme de C et par conséquent, $l \approx r \prec C$ et donc $Q'_I\theta \vee l \approx r \prec C$ (puisque l est maximal dans $Q'_I\theta \vee l \approx r$ i.e. $Q'_I\theta \prec l$). D'autre part par le lemme 20 nous avons $D'_I\sigma \preceq D'$ et $(w_I \bowtie u_I)\sigma \prec w[r]_q \bowtie u)$ (puisque $(w_I \bowtie u_I)\sigma \rightarrow_{l/r} w[r]_q \bowtie u)$ et $w_I\sigma$ et w sont tous les deux maximaux). Ainsi, $D_I\sigma \prec C$ et donc C est redondante dans $\{D_I\sigma, Q'_I\theta \vee l \approx r\}$ et donc dans S.

2. w_I contient un sous-terme t_I, à une position $p \in Pos(w_I)$, tel que $t_I\theta =_I t$. Nous montrons d'abord que la superposition peut s'appliquer entre Q_I et D_I. Par hypothèse t_I n'est pas une variable. De plus, $p \in Pos(w_I)$ et t_I est unifiable avec l_I, avec l'unificateur θ (puisque $t_I\theta = t = l = l_I\theta$). Par définition, il existe donc un unificateur le plus général $\sigma_I \in mgu(t_I, l_I)$ et une substitution τ tels que $\theta = \sigma_I\tau$. Par ailleurs, comme $r \prec l$ et comme $\prec$ est fermé par substitution, nous devons avoir $l_I\sigma_I \not\preceq r_I\sigma_I$. Le même raisonnement s'applique pour les autres restrictions d'ordre et de sélection dans les conditions d'application de la règle de superposition. Donc la superposition est possible entre Q_I et D_I et le résultat de son application est une I-clause $C_I = (Q'_I \vee D'_I \vee w_I[r_I]_p \bowtie u_I)\sigma_I$.
   Nous montrons maintenant que C_I est redondante par rapport à S_I. Considérons la clause $C_I\tau$. Comme $\sigma_I\tau = \theta$, nous avons $C_I\tau = (Q'_I \vee D'_I \vee w_I[r_I]_p \bowtie u_I)\theta =_I (Q' \vee D' \vee w_I[r_I]_p\theta \bowtie u)$. Cette clause *n'est pas* I-équivalente à C en général. En effet, le remplacement de t_I par r_I dans w_I peut créer plusieurs copies de ce terme après avoir déplié les différents contextes inductifs auxquels il appartient alors que dans w une seule instance de ce terme est remplacée. D'après le lemme 19, le terme (standard) $w_I\sigma_I\tau\downarrow$ peut être obtenu à partir de $w_I[r_I]_p\sigma_I\tau\downarrow$ en remplaçant quelques occurrences de r par l. De même pour $w[r]_q = (w_I\sigma_I\tau)[r]_q$ et donc $D' \vee (w[r]_q \bowtie u)$ peut être obtenue à partir de $(D'_I \vee w_I[r_I]_p \bowtie u_I)\theta\downarrow$ en remplaçant quelques occurrences de r par l, i.e. $(D'_I \vee w_I[r_I]_p \bowtie u_I)\theta\downarrow \rightarrow_{l/r} D' \vee (w[r]_q \bowtie u)$, soit d'après le lemme 21, $Q' \vee D' \vee (w[r]_q \bowtie u)$ est redondante par rapport à $\{C_I\tau = Q' \vee (D'_I \vee w_I[r_I]_p \bowtie u_I)\theta\downarrow, Q = Q' \vee l \approx r\}$ puisque $l \succ r$. Donc C est redondante par rapport à $\{C_I\tau, Q\}$. Comme S_I est saturé, C_I est redondante par rapport à S_I et par conséquent C est redondante par rapport à S_I. Or $Inst(S_I) = Inst(S) = S$ puisque S est un ensemble de clauses fermées. Donc par la proposition 15, C est redondante par rapport à S.

3. Sinon, par le lemme 18 w_I contient (à une position o) un N-terme $s_I^M.v$ tel que $t =_I \{s_I[s_I^m.v]_\diamond\}|_p\theta$, où p est un préfixe non vide de la position du trou dans s_I et $m < M\sigma$. Soient N, N' deux nouvelles variables entières et soit $\sigma_M = \{M \rightarrow N+N'+1\}$ (en utili-

sant les notations de la règle de H-superposition). La substitution θ est étendue aux variables N, N' comme suit : $N'\theta = m$ et $N\theta = M\theta - 1 - N'\theta$ (ainsi $M\theta = N\theta + N'\theta + 1$). D'après les relations précédentes nous avons $t =_I \{s_I[s_I^{N'}.v]_\diamond\}|_p\theta$. Donc $\{s_I[s_I^{N'}.v]_\diamond\}|_p$ et l_I ont un unificateur le plus général σ_I qui est plus général que θ, c'est à dire qu'il existe une substitution τ telle que $\theta = \sigma_I\tau$. Le même raisonnement que dans le cas précédent permet d'établir que les restrictions de sélection et d'ordre sur les conditions d'application de la règle de superposition sont vérifiés. Par conséquent, la règle de H-superposition peut s'appliquer sur Q_I et D_I engendrant une I-clause $C_I = (Q'_I \vee D'_I \vee (w_I[s_I^N.s_I[r_I]_p]_o \bowtie u_I))\sigma_M\sigma_I$. Comme $l_I\sigma_I = s_I[s_I^{N'}.v]_\diamond|_p\sigma_I$, nous avons $(s_I^{N+N'+1}.v)\sigma_I =_I (s_I^N.s_I^1.s_I^{N'}.v)\sigma_I$ et donc $(s_I^{N+N'+1}.v)\sigma_I =_I (s_I^N.s_I)\sigma_I[l_I\sigma_I]_p$. D'après le lemme 19, $w_I[s_I^N.s_I[r_I]_p]_o\sigma_I\tau\downarrow$ peut être obtenu depuis $w_I\sigma_I\tau\downarrow$ par plusieurs remplacements de $(s_I^{N+N'+1}.v)\sigma_I\tau\downarrow$ par $(s_I^N.s_I[r_I]_p)\sigma_I\tau\downarrow$. Ainsi, C est redondante par rapport à $C_I\tau \cup S_I$, et puisque S_I est saturé, C_I est alors redondante par rapport à S_I. Ceci implique que C est redondante par rapport à S_I (et donc à S).

- C est dérivée par résolution équationnelle. La preuve est similaire à celle utilisée pour la règle de résolution dans le lemme 17. Soit $C \in S$ telle qu'il existe une clause $D = D' \vee (l \not\approx l)$ dans S et telle que C soit dérivée de D par résolution équationnelle (appliquée dans le cas fermé avec dans ce cas $D' = C$). Par définition, il existe une clause $D_I \in S_I$ et une substitution fermée θ telles que $D_I = D'_I \vee (l_I \not\approx r_I)$, $D_I\theta = D$, $D'_I\theta = D'$ et $l_I\theta = r_I\theta = l$ (i.e. D est une instance fermée de D_I par θ). l_I et r_I sont unifiables ($l_I\theta = r_I\theta = l$) et donc il existe un unificateur le plus général $\sigma_I \in mgu(l_I, r_I)$ et une substitution τ tels que $\theta = \sigma_I\tau$. Un raisonnement similaire à celui du cas précédent permet d'établir que les restrictions de sélection et d'ordre sur les conditions d'application de la règle de résolution équationnelle sont vérifiées. Par conséquent, cette règle peut s'appliquer sur D_I inférant une clause $C_I = D'_I\sigma_I$. Par ailleurs, $C_I\tau = D'_I\sigma_I\tau = D'_I\theta = D' = C$ et donc C est une instance (fermée) de C_I. Or comme C_I est redondante dans S_I, C l'est également puisque c'est une instance fermée de C_I. Finalement comme $Inst(S_I) = Inst(S) = S$ et C est redondante par rapport à S_I, par la proposition 15 C est donc redondante par rapport à S.
- C est dérivée par factorisation équationnelle. Soit $C \in S$ telle qu'il existe une clause $D = D' \vee (l \approx s) \vee (l \approx r)$ dans S et telle que C soit dérivée de D par factorisation équationnelle. $C = D' \vee (s \not\approx r) \vee (l \approx r)$ avec $l \succ r$, $l \succ s$ et $l \approx s \in sel(D' \vee (l \approx s) \vee (l \approx r))$. Par définition, il existe une clause $D_I \in S_I$ et une substitution fermée θ telles que $D_I = D'_I \vee (l_I \approx s_I) \vee (t_I \approx r_I)$, $D_I\theta = D$, $D'_I\theta = D'$, $s_I\theta = s$, $r_I\theta = r$ et $l_I\theta = t_I\theta = l$ (i.e. D est une instance fermée de D_I par θ). l_I et t_I sont unifiables ($l_I\theta = t_I\theta = l$) et donc il existe un unificateur le plus général $\sigma_I \in mgu(l_I, t_I)$ et une substitution τ tels que $\theta = \sigma_I\tau$. Un raisonnement similaire à celui du cas précédent permet d'établir que les restrictions de sélection et d'ordre sur les conditions d'application de la règle de résolution équationnelle sont vérifiées. Par conséquent, cette règle peut s'appliquer sur D_I inférant une clause $C_I = D'_I\sigma_I \vee (s_I \not\approx r_I)\sigma_I \vee (l_I \approx r_I)\sigma$. Par ailleurs, $C_I\tau = D'_I\sigma_I\tau \vee (s_I \not\approx r_I)\sigma_I\tau \vee (l_I \approx r_I)\sigma_I\tau = D'_I\theta \vee (s_I \not\approx r_I)\theta \vee (l_I \approx r_I)\theta = D' \vee (s \not\approx r) \vee (l \approx r) = C$ et donc C est une instance (fermée) de C_I. Or comme C_I est redondante dans S_I, C l'est également puisque c'est une instance fermée de C_I. Finalement comme $Inst(S_I) = Inst(S) = S$ et C est redondante par rapport à S_I, par la proposition 15 C est donc redondante par rapport à S.

□

Finalement le théorème suivant établit la complétude réfutationnelle du calcul $\mathcal{IS}_H$.

Théorème 9 (Complétude réfutationnelle de $\mathcal{IS}_H$)**.** *Tout ensemble insatisfaisable et saturé par*

rapport au calcul $\mathcal{IS}_H$ contient la clause vide. ■

Preuve. La preuve est similaire à celle du théorème 7 :

Soit S_I un ensemble insatisfaisable et saturé par rapport au calcul $\mathcal{IS}_H$. L'ensemble S de toutes les instances fermées de S_I est également insatisfaisable. Par le lemme 22, S (contenant uniquement des clauses standards) est saturé. Or comme la restriction du calcul $\mathcal{IS}_H$ coïncide avec $\mathcal{SC}$ en cas de clauses standards et comme le calcul $\mathcal{SC}$ a la propriété de complétude réfutationnelle, S contient la clause vide $\square$. Par conséquent, S_I contient également la clause vide puisqu'aucune autre I-clause (à part $\square$) n'a pour instance $\square$. $\square$

Chapitre 6

Règles de simplification

Le Scythe l'y trouva, qui la serpe à la main,
De ses arbres à fruits retranchait l'inutile,
Ebranchait, émondait, ôtait ceci, cela.

Jean de La Fontaine

Une des caractéristiques du processus de recherche de preuves (réfutations) dans le contexte de la démonstration par saturation - surtout pour les problèmes non triviaux - est la génération d'un très grand nombre de clauses. La taille de l'espace de recherche augmente alors considérablement et même l'utilisation des stratégies de restriction introduites initialement pour diminuer le nombre d'inférences n'est pas suffisante pour contrôler cette croissance.

L'approche la plus utilisée pour diminuer la taille de l'espace de recherche consiste à utiliser des *règles de simplification*, qui permettent de remplacer certaines clauses par des clauses « plus simples » (et donc susceptibles de générer moins de clauses par la suite) ou de supprimer certaines clauses redondantes (selon le critère de redondance défini au chapitre 2) de l'espace de recherche. La règle de démodulation [WRCS67] (introduite au chapitre 2) est un exemple de règle de simplification. Dans cette règle, nous avons besoin de pouvoir trouver pour un sous-terme w d'un terme u, une équation $s = t$ de l'espace de recherche telle que w est une instance de s, c'est à dire chercher un terme général (s) dont le terme requête (w) est une instance ($s\sigma = w$). Un autre exemple de règle de simplification est la règle de subsomption (avant et arrière). La subsomption avant (forward subsumption) consiste à tester si la clause générée est subsumée par une clause existante (si c'est le cas, la clause est redondante et n'a pas besoin d'être ajoutée à l'espace de recherche). Dans ce cas, nous avons besoin comme pour la démodulation, de pouvoir trouver des termes généralisants. La subsomption arrière (backward subsumption) consiste à éliminer de l'espace de recherche toutes les clauses subsumées par la clause engendrée. Dans ce cas, nous avons besoin de trouver pour un terme donné t, un ensemble de termes qui sont des instances de t.

Afin de réaliser cette recherche de manière efficace, il est nécessaire d'utiliser des techniques d'indexation de termes, qui permettent de rechercher rapidement dans un ensemble de taille importante, un terme qui est une instance d'un terme donné. Des techniques d'indexation ont également été développées pour trouver les bons candidats sur lesquels appliquer les règles d'inférence. En particulier, étant donné une clause C et un terme t apparaissant dans C, nous souhaitons *trouver d'une manière efficace*, parmi un ensemble S de très grande taille (l'espace de recherche), le sous-ensemble des termes apparaissant dans S qui sont unifiables avec t. Les clauses contenant ces termes seront utilisées dans les règles d'inférence avec C pour générer de

nouvelles clauses. Ces techniques n'étant pas utilisées ni par le démonstrateur E ni par DEI nous ne les avons pas considérées dans cette thèse.

Comme décrit dans [RSV01] et en reprenant les mêmes notations, le problème de l'indexation peut être formalisé de la manière abstraite suivante : étant donné un ensemble $\mathcal{L}$ (représentant un *ensemble de termes indexés*), une relation binaire R sur les termes et un terme t, l'objectif est de trouver un sous-ensemble $\mathcal{M}$ de $\mathcal{L}$ tel que tous ses éléments soient en relation avec t, c'est à dire, $\mathcal{M} = \{l \in \mathcal{L} | R(l,t)\}$. Il est parfois utile (à cause principalement de problèmes d'efficacité) de se contenter d'algorithmes calculant en fait un sur-ensemble $\mathcal{N}$ de $\mathcal{M} \subset \mathcal{N}$ au lieu de $\mathcal{M}$ (c'est à dire que $\mathcal{N}$ peut contenir des termes qui ne sont pas en relation avec t). Nous parlons alors dans ce cas de *filtrage imparfait*. Dans le premier cas ($\mathcal{M} = \{l \in \mathcal{L} | R(l,t)\}$) nous parlons de *filtrage parfait*.

Il existe dans la littérature plusieurs techniques d'indexation des termes standards. Pour un traitement détaillé de ces techniques, le lecteur pourra consulter [RSV01, Gra96]. Cependant, à notre connaissance, aucune technique n'existe pour indexer des schémas de termes. Dans la suite de ce chapitre, nous présentons notre contribution, à savoir une technique d'indexation pour une sous-classe des I-termes.

6.1 Définitions et notations

Nous commençons par définir la sous-classe de I-termes que nous considérons dans ce chapitre. Celle-ci est définie en comparant les symboles de tête du terme de base à ceux du contexte. Il apparaît clairement à partir de la définition de la sémantique des I-termes introduite à la section 3.3.1 que le symbole de tête de la forme normale d'un N-terme $t^n.s$ dépend de la valeur de n. La fonction ph que nous introduisons juste après retourne l'ensemble des symboles de tête potentiels d'un terme i.e. l'ensemble des symboles de têtes des termes de la forme $t\sigma \downarrow_I$ où σ est une substitution fermée. Cette fonction est utilisée pour restreindre la classe des I-termes pour laquelle nous proposons un algorithme d'indexation.

Définition 58. La fonction $ph : (T_I \cup T_\diamond) \to \mathcal{P}(\mathcal{F})$ qui retourne les *symboles de tête potentiels d'un I-terme* est définie par : ■

- $ph(x) = \mathcal{F}$ si $x \in V_X$.
- $ph(\diamond) = \emptyset$.
- $ph(f(t_1, \ldots, t_k)) = \{f\}$.
- Si $t \in T_\diamond \backslash \{\diamond\}$, $s \in T_I$, $N \in V_N$ alors $ph(t^N.s) = ph(t) \cup ph(s)$.

Exemple 25 (La fonction ph)**.** $ph(a) = \{a\}$ où a est une constante.
$ph(f(a,b)) = \{f\}$.
$ph(f(a,\diamond,b)) = \{f\}$.
$ph(f(a,\diamond,b)^N.c) = ph(f(a,\diamond,b)) \cup ph(c) = \{f\} \cup \{c\} = \{f,c\}$.
$ph(f(a,\diamond,b)^N.g(\diamond)^M.c) = ph(f(a,\diamond,b)) \cup ph(g(\diamond)^M.c) = \{f\} \cup \{g,c\} = \{f,g,c\}$. ■

Nous définissons maintenant formellement la classe des *I-termes éligibles*.

Définition 59. Un terme $t \in T_I \cup T_\diamond$ est dit *éligible* si et seulement si pour tout N-terme $u^N.v$ apparaissant dans t nous avons $ph(u) \cap ph(v) = \emptyset$. L'*ensemble des termes éligibles* dans T_I (respectivement $T_\diamond$) est noté Ξ (respectivement $\Xi_\diamond$).

Une clause est *éligible* si et seulement si elle ne contient que des termes éligibles.

Un ensemble de clauses est *éligible* si toutes ses clauses le sont. ■

Les N-termes de la forme $u^N.x$, avec $x \in V_X$ sont par définition non éligibles.

Exemple 26 (I-termes éligibles et non éligibles)**.** $f(a,\diamond,b)^N.g(c)$ et $f(\diamond)^N.g(\diamond)^M.b$ sont des I-termes éligibles, $f(\diamond)^N.g(\diamond)^M.f(a)$ ne l'est pas car $ph(g(\diamond)^M.f(a)) = \{g\} \cup ph\{f(a)\} = \{g\} \cup \{f\}$ et donc $f \in ph(g(\diamond)^M.f(a))$. ■

Proposition 23. *L'ensemble des terme éligibles Ξ est fermé par substitution (si les variables sont remplacées par des termes éligibles).* ■

Preuve. Par induction sur la structure des termes éligibles. □

Cette proposition implique qu'il est simple de restreindre un démonstrateur par résolution aux termes éligibles comme le montre la proposition suivante puisqu'il suffit de s'assurer que tous les termes apparaissant dans l'ensemble *initial* sont éligibles.

Proposition 24. *Si un ensemble de clauses S est éligible alors toutes les clauses C produites par résolution ou factorisation à partir de S sont éligibles.* ■

Preuve. Tous les termes produits par la méthode de résolution étant, par définition du calcul, des instances des termes initiaux, ils sont donc également éligibles □

Ξ n'est pas fermé par substitutivité. En effet si nous considérons par exemple l'ensemble (ne contenant initialement que des termes éligibles) $\{x = f(a), f(\diamond)^N.x = b\}$ alors le remplacement de x par $f(a)$ dans $f(\diamond)^N.x$ engendre le terme *non* éligible $f(\diamond)^N.f(a)$. Pour pouvoir appliquer notre technique d'indexation dans le contexte de la preuve par superposition, il faut donc des restrictions supplémentaires pour être sûr qu'aucun terme non éligible n'est généré. Une méthode très simple pour assurer cette propriété est de se restreindre à des N-termes de la forme $t^N.a$, où a est une constante et où l'ordre utilisé est tel qu'une constante ne puisse pas être remplacée par un terme plus *complexe*.

Comme nous le verrons pas la suite, cette restriction définissant les termes éligibles rend le graphe d'index déterministe dans le sens où la lecture du prochain symbole est suffisante pour décider si nous sommes dans le contexte inductif ou dans le terme de base.

Dans la suite de ce chapitre et sauf mention contraire, nous considérons *uniquement* des termes éligibles.

6.2 Graphes d'index

Un *graphe d'index* est un automate (voir par exemple [HMU00]) contrôlé par les symboles apparaissant de gauche à droite dans le terme indexé. Nous distinguons trois types de transitions :

- Des transitions étiquetées avec un symbole du terme indexé ou avec le symbole #. Ces transitions correspondent à la structure du terme indexé.
- Des transitions étiquetées par des symboles de variables arithmétiques (c'est à dire des symboles de V_N). Ces transitions permettent de spécifier que le filtrage d'un N-terme est terminé.
- Des transitions étiquetées avec des équations du type $N = 0$ où $N \in V_N$. Ce type de transition permet de modéliser la fin de traitement du terme de base dans le cas où l'exposant est nul.

Définition 60. Soit le *vocabulaire* $\mathcal{V} = V_X \cup \mathcal{F} \cup \{\diamond, \#\}$ et $\mathcal{L} = \mathcal{V} \cup V_N \cup V_N^0$ l'ensemble des *étiquettes de transition* où V_N^0 est l'ensemble des équations de la forme $N = 0$ avec $N \in V_N$. Un *graphe d'index* est un tuple (S, Σ, δ, F), où $S \in \Sigma$ correspond à l'*état initial*, Σ est un ensemble d'états, $F \subseteq \Sigma$ est l'ensemble des états *finaux (ou acceptants)* et $\delta : \Sigma \times \mathcal{L} \to \Sigma$ est une *fonction de transition. Dans le cas où* $\delta(p, \mu)$ n'est pas défini avec $p \in \Sigma$ est $\mu \in \mathcal{L}$, $\delta(p, \mu)$ est noté $\delta(p, \mu) = \bot$. ■

Si $\delta(p,\mu) = q \neq \perp$ alors nous disons que le graphe contient une *transition* de p vers q étiquetée par μ. q est un *successeur* de p. Une transition étiquetée par un élément d'un ensemble $E \subseteq \mathcal{L}$ (resp. par une étiquette $\mu \in \mathcal{L}$) est appelée une *E-transition* (resp. μ-transition). Une *dérivation* est une suite $(p_i, a_i)_{i\in[1..n]}$ telle que pour tout $i \in [1..n]$ $(p_i, a_i) \in \Sigma \times \mathcal{L}$ et $\delta(p_i, a_i) = p_{i+1}$. Dans ce cas, n est appelée *longueur* de la dérivation. Le *langage représenté par un graphe d'index* est défini comme pour le cas usuel avec les automates [HMU00], i.e. si $IG = (S, \Sigma, \delta, F)$ est un graphe d'index alors le langage représenté par IG et noté $L(IG)$ est l'ensemble de mots $w \in \mathcal{L}^*$ tels que $p_1 = S$, $p_n \in F$ et $w = a_1.a_2.\ldots.a_n$ (w est la concaténations de symboles de transitions menant de l'état initial à l'état final).

Notons que les symboles du vocabulaire terminal permettent de coder non seulement la suite des symboles de fonctions et de constantes du terme considéré, mais aussi les boucles correspondant aux itérations (en donnant pour chacune d'entre-elle la variable entière correspondante et le terme de base).

Nous introduisons quelques opérations basiques sur les graphes d'index. Ces opérations seront utiles dans la suite pour définir les algorithmes de construction des graphes d'index.

Définition 61. Un graphe IG_1 est une *copie* d'une graphe IG_2 si et seulement si IG_1 et IG_2 sont identiques à un renommage des états près et leurs ensembles d'états sont disjoints. ■

Si f est un symbole de $\mathcal{V}$ alors nous dénotons par f (quand il n'y a pas de confusion possible) un graphe arbitraire de la forme $(p, \{p, q\}, \{(p, f) \mapsto q\}, \{q\})$ où $p \neq q$ (c'est à dire un graphe acceptant $\{f\}$).

Nous introduisons la notion de concaténation de deux graphes index. Si nous considérons deux graphes d'index IG_1 et IG_2, l'opérateur de concaténation construit un graphe d'index qui correspond au parcours séquentiel de IG_1 puis IG_2 dans cet ordre.

Définition 62. Soient $IG_1 = (S_1, \Sigma_1, \delta_1, F_1)$ et $IG_2 = (S_2, \Sigma_2, \delta_2, F_2)$. Soit $\{p_1, ...p_q\}$ l'ensemble des états de F_1 (i.e. les états finaux de IG_1) et soit $(IG_2^i = (S_2^i, \Sigma_2^i, \delta_2^i, F_2^i))_{i\in[1..q]}$, q copies de IG_2 (deux à deux disjointes et disjointes également de IG_1). Nous définissons la *concaténation* de IG_1 et IG_2, $IG_1 \bullet IG_2 = (S_1, \Sigma, \delta, F)$ où :

- $\Sigma = \Sigma_1 \cup (\bigcup_{1\leq i\leq q}(\Sigma_2^i \backslash S_2^i))$.
- δ coïncide avec δ_1 sur $(\Sigma_1 \times \mathcal{L})$.
- Pour chaque $i \in [1..q]$, δ coïncide avec δ_2^i sur $(\Sigma_2^i \backslash \{S_2^i\} \times \mathcal{L})$.
- $\forall \mu \in \mathcal{L}, i \in [1..q]$, $\delta(p_i, \mu) = \delta_2^i(S_2^i, \mu)$.
- $F = \bigcup_{i\in[1..q]} F_2^i$.

■

Notons que cette définition suppose qu'il n'y a pas de transition menant vers S_2 dans IG_2. Comme nous allons le voir par la suite, c'est toujours le cas par construction du graphe d'index. Cette opération peut bien évidemment être itérée n fois pour chaque entier n non nul. Nous notons $IG_1 \bullet \ldots \bullet IG_n$ la concaténation (itérée) de $IG_1, \ldots, IG_n$.

Lemme 25. *Soient IG_1, IG_2 deux graphes d'index. Soit $L(IG_1 \bullet IG_2)$ le langage représenté par le graphe index $IG_1 \bullet IG_2$. Si IG_2 ne contient pas de transition vers son état initial alors $L(IG_1 \bullet IG_2) = L(IG_1).L(IG_2)$.* ■

Preuve. La preuve est par induction sur la longueur de la dérivation. □

Nous définissons ensuite une fonction de *fusion* sur les graphes d'index. Cette fonction fusionne les états initiaux de deux graphes d'index IG_1 et IG_2 à condition que les ensembles d'états de IG_1 et IG_2 soient disjoints et qu'il n'y ait pas de symbole autorisant en même temps une transition depuis les états initiaux de IG_1 et de IG_2. Le résultat de la fusion est bien défini et construit une « union » des deux graphes en fusionnant les deux états initiaux.

Définition 63. Soit $IG_1 = (S_1, \Sigma_1, \delta_1, F_1)$ et $IG_2 = (S_2, \Sigma_2, \delta_2, F_2)$ deux graphes d'index tels que $\Sigma_1 \cap \Sigma_2 = \emptyset$ et $\forall f \in \mathcal{V}$ ou bien $\delta(S_1, f) = \perp$ ou bien $\delta(S_2, f) = \perp$.

$IG_1|IG_2$ dénote le graphe d'index $IG = (S_1, \Sigma, \delta, F_1 \cup F_2)$ défini par :

- $\Sigma = \Sigma_1 \cup \Sigma_2 \backslash \{S_2\}$.
- δ coïncide avec δ_1 sur $(\Sigma_1 \times \mathcal{L})$.
- δ coïncide avec δ_2 sur $(\Sigma_2 \backslash \{S_2\} \times \mathcal{L})$.
- $\forall \mu \in \mathcal{L}$, $\delta(S_1, \mu) = \delta_2(S_2, \mu)$ si $\delta_2(S_2, \mu) \neq \perp$.
- $\forall \mu \in \mathcal{L}$, $\delta(S_1, \mu) = \delta_1(S_1, \mu)$ si $\delta_2(S_2, \mu) = \perp$.

■

Nous supposons dans cette définition qu'il n'y a ni N-transition ni 0-transition depuis S_1 et S_2 et qu'il n'y a pas non plus de transition menant vers S_1 ou S_2. Comme pour le cas précédent, ceci est toujours le cas par construction des graphes d'index.

Lemme 26. *Soient IG_1, IG_2 deux graphes d'index. Soit $L(IG_1|IG_2)$ le langage représenté par le graphe index $IG_1|IG_2$. Si IG_1, IG_2 ne contiennent pas de transition vers leurs états initiaux ni de N- ou 0-transition à partir des états initiaux alors $L(IG_1|IG_2) = L(IG_1) \cup L(IG_2)$.* ■

Preuve. La preuve est par induction sur la longueur de la dérivation. □

Nous avons besoin également d'une fonction qui construit des *extensions conservatrices* d'un graphe d'index donné. Cette fonction prend comme arguments un graphe d'index, un tuple composé d'un état source, un symbole de $\mathcal{V}$ pour étiqueter la transition et un état de destination. Le résultat est un graphe d'index qui contient le graphe initial plus une nouvelle transition depuis l'état source vers l'état final étiqueté par le symbole donné. L'état de destination est supposé final si l'état source l'est.

Définition 64. Soit $IG = (S, \Sigma, \delta, F)$ un graphe d'index et soient $\mu \in \mathcal{L}$ et p, q deux états tels que $p \in \Sigma$ ou $q \in \Sigma$ et $\delta(p, \mu) = \perp$[1]. $IG \cup \{p \rightarrow_\mu q\}$ dénote le graphe d'index $IG' = (S, \Sigma \cup \{p, q\}, \delta', F')$ défini par :

- $F' = F$ si $p \notin F$ et $F' = F \setminus \{p\} \cup \{q\}$ sinon.
- Si $(r, \mu) \neq (p, \mu)$ alors δ' coïncide avec δ pour (r, μ).
- $\delta'(p, \mu) = q$.

■

Nous définissons également une fonction qui remplace un état p d'un graphe index par un autre état q.

Définition 65. Soit $IG = (S, \Sigma, \delta, F)$ un graphe index et soient p, q deux états tels que $p \in \Sigma$. $IG[p/q]$ désigne le graphe d'index $IG' = (S', \Sigma \backslash \{p\} \cup q, \delta', F')$ défini par :

- $F' = F$ si $p \notin F$ et $F' = F \setminus \{p\} \cup \{q\}$ sinon.
- Si $p = S$ alors $S' = q$, sinon $S' = S$.
- δ' coïncide avec δ sur toutes les transitions à l'exception de celles impliquant l'état p : q remplacera p dans IG'. Soit $(\mu, r) \in \mathcal{L} \times \Sigma$, si $r \neq p$ alors $\delta'(r, \mu) = \delta(r, \mu)$, sinon $\delta'(q, \mu) = \delta(p, \mu)$ et $\forall r' \in \Sigma, \eta \in \mathcal{L}, \delta(r', \eta) = p \Rightarrow \delta'(r', \eta) = q$.

■

L'opération de remplacement peut naturellement être étendue à des remplacement multiples d'états p_i par des états q_i $(1 \leq i \leq n)$.

[1] Cette dernière condition assure que nous ne sommes pas en train de tenter d'ajouter une transition déjà existante dans IG.

Définition 66 (isomorphisme)**.** Soient $IG_1 = (S_1, \Sigma_1, \delta_1, F_1)$ et $IG_2 = (S_2, \Sigma_2, \delta_2, F_2)$ deux graphes d'index. IG_1 et IG_2 sont *isomorphes* si et seulement s'il existe une bijection $\varphi : \Sigma_1 \to \Sigma_2$ telle que $\forall p, q \in \Sigma_1, \delta_1(p) = q \Leftrightarrow \delta_2(\varphi(p)) = \varphi(q)$. ■

Toutes les opérations définies dans cette section sont uniques à un renommage des états près.

6.3 Indexation d'un I-terme

Dans cette section nous montrons comment construire le graphe d'index correspondant à un I-terme donné. Nous montrerons ensuite comment fusionner le graphe d'index obtenu avec un graphe représentant un ensemble de termes. Nous créons pour chaque terme ou contexte t un graphe d'index noté $IG(t)$ qui représente t. $IG(t)$ est défini inductivement par :

1. Si $t = f(t_1, t_2, \ldots, t_n)$ (avec éventuellement $n = 0$ et/ou $f \in V_X \cup \{\diamond\}$) alors $IG(t)$ est obtenu en concaténant $IG(t_1), \ldots, IG(t_n)$ et en ajoutant un état initial S n'apparaissant pas dans $IG(t_1), \ldots, IG(t_n)$ et une $\mathcal{V}$-transition de S vers l'état initial de $IG(t_1)$ étiquetée par le symbole f. Formellement : $IG(t) = f \bullet IG(t_1) \bullet \ldots \bullet IG(t_n)$.
2. Si $t = f(t_1, \ldots, t_n)^N.s$. Soit $IG_i = IG(t_i) = (S_i, \Sigma_i, \delta_i, F_i)$ pour $1 \leq i \leq n$. $IG_s = IG(s) = (S_s, \Sigma_s, \delta_s, F_s)$ et soit $IG^0_s = (S^0_s, \Sigma^0_s, \delta^0_s, F^0_s)$ une copie de IG_s. Nous supposons que les graphes IG_s, IG^0_s, IG_i ($i \in [1..n]$) ne partagent aucun état. c_s désigne le nombre d'états terminaux dans IG_s (ou IG^0_s).
   Le graphe d'index de t est composé de deux branches IG^0_b et $IG_{N\neq 0}$. IG^0_b est basé sur IG^0_s et correspond au cas $N = 0$. $IG_{N\neq 0}$ correspond au cas $N \neq 0$. Il est obtenu en concaténant deux graphes IG_c et IG_b correspondant respectivement au contexte itéré et au terme de base. Plus précisément, $IG^0_b, IG_c, IG_b, IG_{N\neq 0}$ et IG sont définis par :
   (a) IG_b et IG^0_b sont obtenus à partir de IG_s et de IG^0_s respectivement en ajoutant pour chaque état $p_s \in F_s$ (resp. $p^0_s \in F^0_s$) un nouvel état q_s (resp. q^0_s) et une transition de p_s vers q_s (resp. de p^0_s vers q^0_s) étiquetée par N (resp. par $N = 0$). Les états terminaux de IG_b and IG^0_b sont les états q_s, q^0_s. Formellement, nous considérons les séquences de graphes d'index $(IG^i_b)_{i\in[1..c_s]}$ et $(IG^{0,i}_b)_{i\in[1..c_s]}$ avec :
   – $IG^0_b = IG_s$ et $IG^{0,0}_b = IG^0_s$.
   – $IG^{i+1}_b = IG^i_b \cup \{q^{i+1}_s \to_N n^{i+1}_s\}$.
   – $IG^{0,i+1}_b = IG^{0,i}_b \cup \{q^{0,i+1}_s \to_{(N=0)} n^{0,i+1}_s\}$.
   Nous avons alors $IG_b = IG^{c_s}_b$ et $IG^0_b = IG^{0,c_s}_b$.
   (b) IG_c est obtenu à partir de $IG_1 \bullet \ldots \bullet IG_n$ en ajoutant un nouvel état r_q et une #-transition de q vers r_q pour chaque état terminal q dans IG_n. Chaque #-transition est utilisée pour créer une boucle dans le graphe (pour coder le dépliage itéré). Formellement, en notant les états finaux de IG_n $(q^i_n)_{i\in[1..c_n]}$ où c_n est le nombre d'états terminaux dans IG_n, nous considérons la séquence de graphe d'index $(IG^i_c)_{i\in[1..c_n]}$ avec :
   – $IG^0_c = IG_n$.
   – $IG^{i+1}_n = IG^i_n \cup \{q^{i+1}_n \to_\# r^{i+1}_q\}$.
   IG_c est défini par : $IG_c = IG^{c_n}_n$.
   (c) $IG_{N\neq 0}$ est obtenu à partir de $IG_c \bullet IG_b$ en ajoutant une f-transition depuis chaque état r_q vers l'état initial de IG_c (les états r_q sont ceux introduits dans le point précédent pour la construction de I_c). Formellement, en notant $\{r^i_q\}_{i\in[1..m]}$ les états r_q (m est le nombre de ces états) et S_c l'état initial de IG_c, nous considérons la séquence de graphe d'index $(IG^i_{N\neq 0})_{i\in[1..c_n]}$ avec :

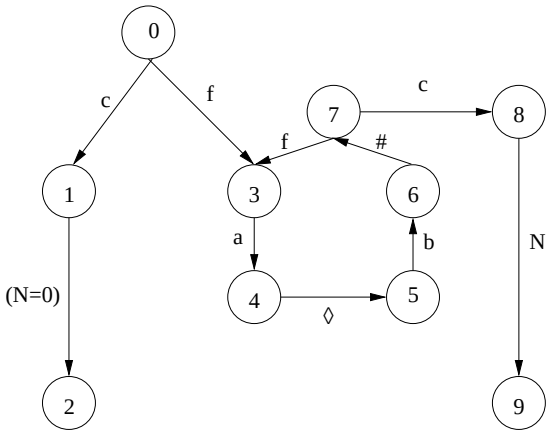

FIG. 6.1 – Le graphe d'index du I-terme $f(a, \diamond, b)^N.c$

- $IG^0_{N\neq 0} = IG_c \bullet IG_b$.
- $IG^{i+1}_{N\neq 0} = IG^i_{N\neq 0} \cup \{r_q^{i+1} \rightarrow_f S_c\}$.
  $IG_{N\neq 0}$ est défini par : $IG_{N\neq 0} = IG^m_{N\neq 0}$.

(d) Finalement, IG est construit en commençant par étendre $IG_{N\neq 0}$ avec une f-transition depuis un nouvel état (qui sera l'état initial de IG) vers l'état initial de $IG_{N\neq 0}$ étiquetée avec le symbole f. Ensuite, le résultat est fusionné avec le graphe IG^0_b. Formellement, IG est défini par : $IG = IG^0_b|(f \bullet IG_{N\neq 0})$. L'opération de fusion utilisée ici est bien définie. En effet, comme nous n'utilisons que des termes éligibles et comme f est le symbole de tête du contexte inductif, par définition, f ne peut pas être le symbole de tête du terme de base et donc il n'y a pas de transition depuis l'état initial de IG^0_b étiquetée avec f.

Le fait de considérer les deux copies du terme de base IG_s et IG^0_s peut sembler surprenant à première vue (car cela augmente la taille du graphe obtenu), mais cela est nécessaire pour éviter les ambiguïtés quand des N-termes distincts partagent le même terme de base (voir la remarque 1 de la page 91).

6.4 Indexation d'un ensemble de I-termes

À la section précédente, nous avons vu comment construire le graphe d'index associé à un terme ou un contexte donné. Cela n'est pas suffisant puisque en pratique nous avons besoin d'indexer simultanément des ensembles de termes, tout en factorisant les préfixes communs entre les termes. Notre objectif est de définir un graphe pour l'indexation de plusieurs I-termes éligibles. Ce graphe est construit à partir d'un index initialement vide en insérant successivement le graphe d'index de chaque terme. Cette opération d'insertion doit garantir le partage des préfixes communs dans le graphe obtenu.

Soit $IG = (S, \Sigma, \delta, F)$ un graphe d'index associé à un ensemble de I-termes (supposé déjà construit) et soit t un I-terme éligible. Soit $IG(t) = (S_t, \Sigma_t, \delta_t, F_t)$ un graphe d'index construit pour t. L'objectif est d'intégrer le graphe $IG(t)$ dans IG en fusionnant certains de ces états avec les états existants de IG. Sans perte de généralité, nous pouvons supposer que $\Sigma \cap \Sigma_t = \emptyset$ (quitte à procéder à un renommage des états). Nous définissons une suite $(IG_i)_{i\in\mathbb{N}}$ de graphes index isomorphes à $IG(t)$ en remplaçant itérativement les états de $IG(t)$ par des états de IG. À

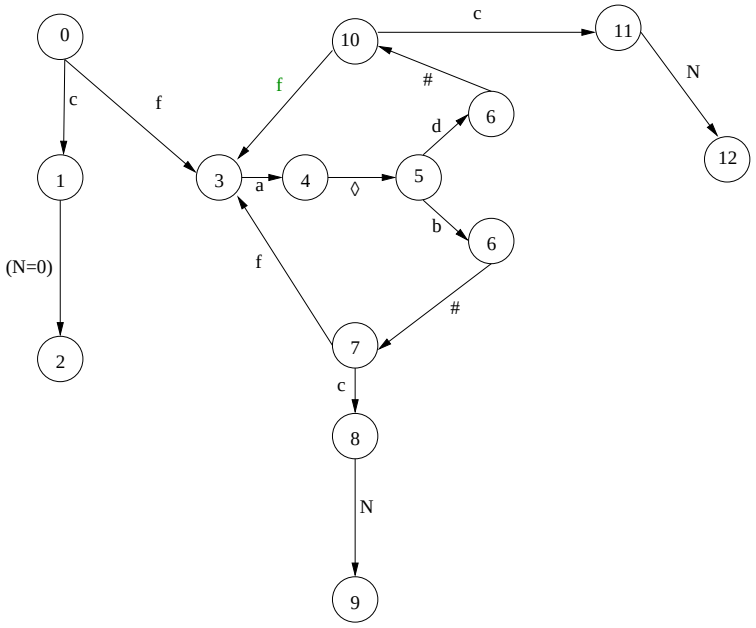

FIG. 6.2 – Graphe d'index pour indexer les deux I-termes $f(a, \diamond, b)^N.c$ et $f(a, \diamond, d)^N.c$

chaque étape, Φ_i désigne l'ensemble des couples $(T, T') \subseteq \Sigma_t \times \Sigma$ tels que T peut être remplacé par T'.

- $IG_0 = IG(t)$, $\Phi_0 = \{(S_t, S)\}$.
- $IG_{i+1} = IG_i[p_1/q_1, \ldots, p_{n_i}/q_{n_i}]$ où $\Phi_i = \{(p_j, q_j)\}_{j \in [1..|\Phi_i|]}$.
- $\Phi_{i+1} = \{(p_t, p) \in \Sigma_t \times \Sigma \mid \exists \mu \in \mathcal{L}, \exists (q, r) \in \Phi_i, \delta_i(q, \mu) = p_t \wedge \delta(r, \mu) = p\}$.

La proposition suivante montre que ce processus termine toujours.

Proposition 27. *Les suites $(IG_i)_{i \in \mathbb{N}}$ et $(\Phi_i)_{i \in \mathbb{N}}$ sont convergentes.* ■

Preuve. Soit $n_i = |\Sigma_t \cap \Sigma_i|$, c'est à dire le nombre d'états appartenant en même temps à Σ_i et Σ_t. La suite $(n_i)_{i \in \mathbb{N}}$ est décroissante (puisqu'à chaque étape on remplace des états de Σ_t par d'autres états de Σ) est donc elle est convergeante.

Soit $n = \min_i(n_i)$ et $\hat{i} = min\{i | n = n_i\}$. Si $n = 0$ alors $\Sigma_i \cap \Sigma_t = \emptyset$ et donc $\Phi_{i+1} = \emptyset$ par définition de Φ_i. Si $n \neq 0$ alors $\Phi_{i+1} = \emptyset$ car sinon, si on considère $(q, p) \in \Phi_{i+1}$ comme par définition, $q \in \Sigma_t$ et $q \notin \Sigma_{i+1}$, alors on aura $n_{i+1} < n_i$ ce qui est absurde. Nous avons alors $IG^*(t) = IG_{\hat{i}}$ ($\forall i \geq \hat{i},\ IG_{i+1} = IG_i$ par construction). □

Proposition 28. *Les graphes d'index $IG(t)$ et $IG^*(t)$ sont isomorphes.* ■

Preuve. Par induction sur la longueur de la dérivation de $IG^*(t)$. □

Pour chaque graphe d'index IG et pour chaque état terminal p de IG, $\theta_{IG}(p)$ désigne le I-terme correspondant à l'état p.

Exemple 27. La figure 6.2 montre le graphe d'index partagé construit pour indexer les deux I-termes $f(a, \diamond, b)^N.c$ et $f(a, \diamond, d)^N.c$.

■

6.5 L'algorithme de filtrage

L'objectif de cette section est de décrire un algorithme pour retrouver, étant donné un graphe d'index $IG = (S, \Sigma, \delta, F)$, tous les termes indexés qui *filtrent* un terme candidat c'est à dire l'ensemble $\{t \mid t \in \theta_{IG}(F), \exists \sigma\, t\sigma \downarrow = s\}$ où σ est une substitution. La difficulté principale réside dans le traitement des N-termes en entrée et dans les $(\diamond, \#)$-transitions dans l'index. Les $\mathcal{F}$-termes et les $\{V_X \cup \mathcal{F}\}$-transitions sont traités exactement comme dans le cas standard.

6.5.1 Lemme de décomposition

Nous commençons par établir les propriétés théoriques sur lesquelles sont fondées l'algorithme de filtrage. Le lemme suivant introduit une propriété caractéristique des termes éligibles. Intuitivement, il affirme qu'un N-terme ne peut être filtré que par un N-terme, de plus, si $t^N.s$ filtre un N-terme $u^M.v$ alors il existe un entier i tel que t^i filtre u et s filtre v avec la même substitution σ. En particulier, après dépliage, la position du trou dans t^i doit correspondre à celle de u. Ce lemme est essentiel car il permet de « décomposer » le filtrage en considérant séparément le contexte inductif et le terme de base.

Exemple 28. Soit le N-terme (éligible) $r = f(a, f(a, \diamond))^N.g(b)$ à filtrer. Il est clair qu'un terme du type $f(a, x)$ ne peut pas filtrer r puisque $ph(f(a, X)) = \{f\}$ alors que $ph\{r\}$ contient b. Ceci est vrai pour n'importe quel terme (éligible) n'ayant pas la structure N-terme. Donc pour filtrer un N-terme (éligible) il faut obligatoirement un N-terme (éligible). Par ailleurs, le N-terme $f(a, \diamond)^M.g(x)$ filtre r. de plus dans ce cas en prenant $i = 2$, nous avons $f(a, \diamond)^i = f(a, f(a, \diamond))$ filtre $f(a, f(a, \diamond))$ et $g(x)$ filtre $g(b)$. Cet exemple explique ce principe de décomposition dans le filtrage, i.e. filtrer séparément le contexte inductif et le terme de base. En particulier, dans ce cas, pour filtrer le contexte inductif de r, il faut déplier le contexte inductif du terme filtrant $(f(a, \diamond))$ en utilisant l'entier i. À noter également la correspondance des positions des trous après dépliage. ■

Lemme 29. *Soient w et $u^M.v$ deux I-termes éligibles (non variables). S'il existe une substitution σ telle que $w\sigma \downarrow = u^M.v$ alors :*
- *w est de la forme $t^N.s$.*
- *il existe $i \in \mathbb{N}$ tel que $N\sigma = i.M$.*
- *$t^i\sigma \downarrow = u$ et $s\sigma \downarrow = v$.*

■

Le lemme 29 n'est pas valable si w n'est pas éligible. En effet, le terme non éligible $f(\diamond)^N.f(a)$ peut être filtré par exemple aussi bien par le N-terme $f(\diamond)^M.a$ que par le terme (qui n'est pas un N-terme) $f(f(\diamond)^M.a)$. Une conséquence du lemme 29 est que si le terme en entrée est un N-terme $u^M.v$ alors l'état courant dans le graphe d'index doit également correspondre à un N-terme $t^N.s$[2]. De plus comme expliqué plus haut, le filtrage peut être décomposé en deux parties, qui correspondent respectivement au contexte inductif u et au terme de base v. La valeur de l'exposant N est alors égale à $i.M$, où i est le nombre de dépliages de t nécessaires pour filtrer u.

Preuve. Nous prouvons successivement les différentes propriétés :
- **w est un N-terme de la forme $t^N.s$.** Comme $u^M.v$ est un I-terme éligible, son symbole de tête dépend de M : si M est instancié à 0 alors le symbole de tête est celui de v sinon, c'est celui de u. De plus, ces deux symboles sont différents puisque les I-termes considérés sont exclusivement éligibles. Si w n'est pas un N-terme alors quelle que soit la substitution

[2] À noter qu'à cause du partage de termes communs, l'état courant peut en fait correspondre à *plusieurs* termes, incluant à la fois des N-termes et des $\mathcal{F}$-termes.

θ choisie, le symbole de tête de $w\theta$ est indépendant de M ce qui est impossible. Donc, w est de la forme $t^N.s$.

- ***N*σ est multiple de *M***. Supposons que $N\sigma$ n'est pas multiple de M, c'est à dire, qu'il existe une expression arithmétique T indépendante de M telle que $N\sigma = i.M + T$ où $i \in \mathbb{N}$. Soit ρ une substitution qui assigne la valeur 1 à toutes les variables arithmétiques à l'exception de M. Nous avons $N\sigma\rho = (i.(M\sigma) + T\rho)$ où $T\rho$ est un entier (c'est à dire $T\rho \in \mathbb{N}$). Le symbole de tête de $((t^N.s)\sigma\rho)\downarrow$ ne dépend pas dans ce cas de M. Ceci est impossible car le symbole de tête de $(u^M.v)\rho$ dépend effectivement de M. Ainsi, il existe $i \in \mathbb{N}$ tel que $N\sigma = i.M$.
- ***t*^*i*^σ ↓= *u* et *s*σ ↓= *v***. Nous avons $(t^N.s)\sigma\downarrow = u^M.v$. Soit θ une substitution qui assigne des termes fermés à toutes les variables de $u^M.v$ et telle que $M\theta > 0$. Soit $k = M\theta - 1$ ($k \in \mathbb{N}$).
  Par définition, le terme $(u^k.v)\theta$ apparaît dans $(u^M.v)\theta = (u[u^k.v]_p)\theta$ (à une position p, qui correspond à la position du trou de u). En fait, $(u^k.v)\theta$ apparaît une *seule fois* dans $(u^M.v)\theta$. En effet, $(u^k.v)\theta$ ne peut pas apparaître à l'intérieur de lui même et si $(u^k.v)\theta$ apparaît dans $(u[u^k.v]_p)\theta$ à une position $q \neq p$ alors $(u^k.v)\theta$ apparaîtrait dans $u\theta$ ce qui est impossible puisque $(u^k.v)\theta$ contient tous les sous-termes de $u\theta$.
  Comme $(t^N.s)\sigma = u^M.v$, nous avons $(t^N.s)\sigma\theta = (u^M.v)\theta = (u[u^k.v]_p)\theta$. Donc, $(u^k.v)\theta$ apparaît une seule fois dans $(t^N.s)\sigma\theta$ à une position p. Nous montrons qu'en fait cette position correspond à un terme (déplié) $t^l.s$ où $l \in \mathbb{N}$, c'est à dire que $p = q^i$, où q est la position du trou dans t et $i \in \mathbb{N}$.
  Nous pouvons écrire p comme $p = q^i.r$ où q n'est pas un préfixe de r (avec éventuellement $i = 0$ ou $r = \epsilon$). Nous voulons prouver qu'en fait $r = \epsilon$, c'est à dire que $p = q^i$. Supposons que ce ne soit pas le cas, il y a alors deux possibilités :
  - r est un préfixe strict de q. Notons que le symbole de tête de $(u^k.v)\theta$ dépend de la valeur de k (si $k = 0$ alors le symbole de tête est celui de v, sinon, c'est celui de u et ces deux symboles sont différents puisqu'on considère uniquement des I-termes éligibles). Dans ce cas en revanche (r est un préfixe strict de q), le terme $(u^k.v)\theta$ à la position $p = q^i.r$ dans $(t^N.s)\sigma\theta$ apparaît le long du chemin inductif de t mais non à la position du trou. Donc, son symbole de tête est le même indépendemment de la valeur de k, ce qui est impossible.
  - r n'est pas un préfixe de q. Nous choisissons alors $M\theta$ telle que $M\theta > 1$ et $M\theta > i$. Comme $N\sigma$ est un multiple de M, nous avons $N\sigma\theta > i$. Ainsi, t est déplié dans $(t^N.s)\sigma\theta$ au moins i fois. Le terme à la position q^i dans $(t^N.s)\sigma\theta$ est nécessairement alors de la forme $(t^l.s)\sigma\theta$ où $l = (N\sigma\theta) - i > 0$. Comme r n'est pas un préfixe de q (ni q n'est un préfixe de r par définition de r), le terme à la position r dans $(t^l.s)\sigma\theta$ appartient au contexte inductif de $t\sigma\theta$. Dans ce cas, ce terme doit apparaître plusieurs fois dans $(t^N.s)\sigma\theta$ (une occurrence par dépliage, c'est à dire au moins $(N\sigma\theta)$ occurrences) ce qui est impossible puisque $(u^k.v)\theta$ apparaît une seule fois dans $(t^N.s)\sigma\theta$.

Ceci prouve qu'il existe $i \in \mathbb{N}$ tel que $p = q^i$ et $(t^N.s)\sigma = ((t^i)\sigma)^M.(s\sigma) = u^M.v$ et donc $(t^i)\sigma = u$ et $s\sigma = v$. □

6.5.2 Description informelle de l'algorithme

Nous proposons à présent une description informelle de l'algorithme de filtrage qui est fondé sur cette décomposition. Les variables et les symboles de fonctions sont traités exactement comme pour le cas standard. Quand un terme lu en entrée est de la forme $u^N.v$, Nous stockons le terme de base v ainsi que l'exposant N dans une pile `BT` ensuite nous essayons de filtrer le contexte inductif u. Lors du traitement de u, une $\diamond$-transition doit être rencontrée. Le I-terme correspondant à

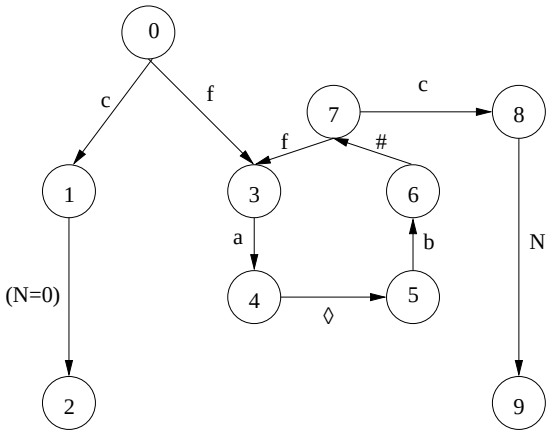

FIG. 6.3 – Graphe d'index de $t = f\,(a, \diamond, b)^N\,.c$

la boucle actuelle dans le graphe d'index doit alors être déplié pour filtrer le terme en entrée (par exemple, le contexte $f(\diamond)$ doit être déplié une fois pour filtrer $f(f(\diamond))$). Or le contexte ne peut pas être déplié à ce stade puisque nous n'avons pas encore atteint la #-transition qui pointe sur son état initial. Par conséquent, nous avons besoin d'utiliser une autre pile `H` pour stocker le terme en entrée et pour retarder son traitement jusqu'au filtrage complet de la partie restante du contexte inductif. Quand nous arrivons à une #-transition, un terme est dépilé de `H`. Si ce terme est différent du $\diamond$ alors le terme indexé est déplié pour filtrer la partie restante du contexte inductif. Sinon, le contexte itéré u a été entièrement filtré, et donc nous commençons le traitement du terme de base (stocké dans `BT`). Après avoir filtré avec succès le terme de base, nous appliquons la N-transition, qui nous permet de calculer la valeur de l'exposant. Le terme de base et l'exposant stockés dans `BT` sont alors dépilés.

Exemple 29. Soit le terme indexé $t = f\,(a, \diamond, b)^N\,.c$ dont le graphe d'index est donné dans la figure 85. nous décrivons le déroulement de l'algorithme pour le filtrage du terme $f\,(a, f\,(a, c, b)\,, b)$. La solution de ce problème est $\sigma = \{N \to 2\}$. ■

L'état initial est 0 et le terme à filtrer est $f\,(a, f\,(a, c, b)\,, b)$. L'état suivant la lecture du symbole f est 3 et les termes restant à traiter sont $a, f\,(a, c, b)\,, b$. Après lecture de a l'algorithme avance vers 4. À ce moment le prochain terme à traiter est $f\,(a, c, b)$ mais puisque l'algorithme rencontre la $\diamond$-transition, ce terme n'est pas traité tout de suite mais retardé pour plus tard : il est empilé dans `H` et le prochain terme à traiter devient b. Ensuite après lecture de b l'algorithme arrive à l'état 6. À ce niveau, il n'y a plus de termes à traiter mais comme l'algorithme rencontre une #-transition, il dépile $f\,(a, c, b)$ de `H` et incrémente le compteur cnt initialement nul. Le prochain terme à traiter est $f\,(a, c, b)$ et l'état courant est 7. Après lecture de f l'algorithme avance vers l'état 3 puis vers l'état 4. Il rencontre encore une fois une $\diamond$-transition et donc retarde le traitement de c en avançant vers 5 puis vers 6 en lisant b. La #-transition vers 7 dépile `H` et incrémente cnt qui devient égal à 2. Le prochain terme à traiter est c et donc l'algorithme avance vers l'état 8 puis vers l'état 9 après la N-transition qui ajoute la contrainte $N = cnt$ soit $N = 2$ qui est la solution du problème. Nous pouvons schématiser cette exécution comme suit :

$$0 \xrightarrow{f} 3 \xrightarrow{a} 4 \xrightarrow{\diamond} 5 \xrightarrow{b} 6 \underset{cnt=1}{\xrightarrow{\#}} 7 \xrightarrow{f} 3 \xrightarrow{a} 4 \xrightarrow{\diamond} 5 \xrightarrow{b} 6 \underset{cnt=2}{\xrightarrow{\#}} 7 \xrightarrow{c} 8 \underset{N=cnt}{\xrightarrow{N}} 9$$

Nous devons mémoriser le premier état correspondant à la boucle d'un terme indexé ainsi que le symbole de tête pour vérifier que la première transition rencontrée après avoir filtré le contexte inductif boucle réellement sur cet état. C'est pourquoi nous avons besoin d'une autre pile que nous notons `L` dans la suite. Plus de détails seront fournis dans l'exemple 32 page 92.

Dans le cas où deux boucles différentes partagent un préfixe commun, nous avons besoin de nous assurer que les mêmes transitions sont appliquées à chaque itération. Ceci est réalisé en marquant tous les états rencontrés (en utilisant un ensemble mk). Si un successeur de l'état courant est marqué, alors nous ignorons tout simplement toutes les autres transitions. L'exemple suivant fournit plus de détails sur l'utilité de mk.

Exemple 30. Soit le terme à filtrer $f(a, f(a, c, b), d)$ et soit un index contenant les termes (indexés) $f(a, \diamond, b)^N.c$ et $f(a, \diamond, d)^N.e$. Les deux contextes inductifs de ces deux termes partagent le préfixe commun $fa\diamond$. En commençant à partir de l'état initial du graphe d'index, nous lisons successivement les symboles f, a et nous arrivons à la $\diamond$-transition, qui pointe sur un état p. Comme expliqué plus haut, nous avons besoin de stocker le terme $f(a, c, b)$ dans `H`. Ensuite, nous lisons le symbole d et nous atteignons la #-transition. Le terme $f(a, c, b)$ est dépilé depuis `H` et le contexte inductif du graphe d'index est dépilé. Nous relisons les symboles f, a et nous arrivons à l'état p encore une fois. Le terme c est stocké et nous essayons de filtrer le terme b. Or, le contexte inductif a déjà été déplié, et nous savons que ce contexte est $f(a, \diamond, d)$ et *non pas* $f(a, \diamond, b)$. Par conséquent, la transition b (qui n'est pas marquée) peut être ignorée et l'algorithme échoue (puisqu'il ne trouve pas de transition valide). Sans utilisation des marques, la b-transition serait applicable ce qui entraînerait une solution incorrecte. ■

Pour trouver les valeurs des variables en exposant, nous avons besoin de compter le nombre de fois où nous entrons dans la boucle correspondante dans le graphe d'index. Ceci peut être fait en comptant le nombre de #-transitions. Cependant, un seul compteur n'est pas suffisant. En fait, le dépliage peut correspondre ou bien au filtrage d'un terme standard (par exemple si $f(f(f(a)))$ est filtré par $f(\diamond)^N.a$) ou bien au filtrage d'un contexte inductif (si $f(f(\diamond))^N.a$ est filtré). Nous utilisons un compteur b pour le premier type de dépliage et un autre compteur a pour le deuxième type. Par exemple si le terme indexé est $t = f(e, \diamond, d)^N.c$ où e, d et c sont des symboles de constantes, alors c, $f(e, c, d)$, $f(e, f(e, c, d), d)$, ... sont des instances dépliées de t. Le filtrage de ce type d'instances correspond à des dépliages successifs représentés dans cet algorithme par une boucle dans le graphe d'index. L'objectif du premier type de compteur est de compter le nombre de fois où le contexte itéré a été déplié. Un compteur de ce type est appelé un *b-compteur*. D'un autre côté, si nous considérons par exemple $t = f(\diamond)^N.c$ alors $f(f(\diamond))^N.c$, $f(f(f(\diamond)))^N.c$, ... sont également des instances de t. Dans ce cas, le contexte itéré de l'instance correspond à plusieurs dépliages successifs du contexte inductif de l'index. Lors du filtrage de ce type d'instance un seul parcours du contexte itéré de l'instance correspond en fait à plusieurs boucles faites dans l'index (le contexte itéré de l'instance correspond à plusieurs dépliages de l'index). C'est pourquoi nous introduisons les deux compteurs différents (a et b). Les deux compteurs sont stockés dans une pile (`Cnt`), avec l'état correspondant dans graphe d'index. Nous avons besoin également de stocker une variable de décision (ou tout simplement flag) pour indiquer le type du terme actuellement en traitement (I-terme ou contexte). Nous utilisons pour cela une pile `Ctx`. Les deux compteurs peuvent naturellement être utilisés en même temps. Par exemple, supposons que l'index contient le terme (indexé) $t = f(\diamond)^N.c$ et que le terme en lecture est $s = \underbrace{f(f}(\underbrace{f(f(f(\diamond)))^M}.c))$ qui est bien une instance de t. Le b-compteur est utilisé pour compter le nombre de dépliages avant d'atteindre le N-terme imbriqué (la valeur du compteur est 2 dans ce cas). Le a-compteur est quant à lui utilisé pour compter le nombre de boucles imbriquées nécessaires pour filtrer $f(f(f(\diamond)))$ en utilisant $f(\diamond)$ (la valeur du compteur est 3 dans ce cas). On a $s = t\sigma$ avec $\sigma = \{N \mapsto 3.M + 2\}$.

Finalement, nous utilisons un ensemble d'équations E pour stocker la valeur des variables apparaissant dans le graphe d'index (à la fois les variables ordinaires et les variables arithmétiques). L'algorithme échoue si cet ensemble d'équations est insatisfaisable (en considérant les variables apparaissant dans les termes en entrée comme des constantes).

6.5.3 Définition formelle

Nous donnons maintenant une description mathématique précise de l'algorithme de filtrage. Celui-ci manipule des tuples de la forme $\mathcal{C} = (q, \mathtt{W}, \mathtt{BT}, \mathtt{H}, \mathtt{Cnt}, \mathtt{Ctx}, \mathtt{L}, \mathrm{E}, mk)$ où :

- $q \in \Sigma$ est l'*état courant*, $\mathtt{W} \in (\Xi \cup \Xi_\diamond)^*$ est une pile utilisée pour stocker les *termes restant à traiter*. Initialement $q = S$ et $\mathtt{W} = s$ où s est le terme à filtrer (ou terme requête).
- $\mathtt{BT} \in (\Xi \times V_N)^*$ est une pile utilisée pour stocker les termes de base et les exposants des I-termes éligibles.
- $\mathtt{H} \in (\Xi \cup \Xi_\diamond)^*$ est une pile utilisée pour stocker les termes qui apparaissent aux positions des trous durant le filtrage.
- $\mathtt{Cnt} \in (\Sigma \times \mathbb{N} \times \mathbb{N})^*$ est une pile utilisée par les #-transitions pour stocker des paires d'entiers utilisés comme des compteurs. La première composante est utilisée pour identifier l'état correspondant à la dernière #-transition, afin de déterminer s'il s'agit de la première fois où nous visitons une #-transition donnée ou non.
- $\mathtt{Ctx} \in (\{0,1\} \times \mathbb{N} \times \mathbb{N})^*$ est une pile utilisée pour garder le type du contexte ouvert courant (Le I-terme courant non encore entièrement filtré) : 0 pour les $\mathcal{F}$-termes et 1 pour les N-termes. Les deuxième et troisième composants sont utilisés uniquement pour les termes fonctionnels non constants, ils désignent respectivement l'arité du symbole de fonction et le nombre d'arguments restants à traiter.
- $\mathtt{L} \in (\Sigma \times \mathcal{F})^*$ est une pile utilisée pour filtrer les N-termes.
- E est un ensemble d'équations qui sont ou bien de la forme $N = a.M + b$ où N et M sont des variables arithmétiques ($N, M \in V_N$) et a et b sont des entiers ($a, b \in \mathbb{N}$) ou bien de la forme $x = t$ où x est une variable ordinaire ($x \in V_X$) et t un terme éligible ($t \in \Xi$).
- mk est un ensemble d'états dits *marqués*.

Un tuple de ce type est appelé une *configuration*. Initialement nous avons $\mathcal{C} = \mathcal{C}_0 = (S, s, \epsilon, \epsilon, \epsilon, \epsilon, \emptyset, \{S\})$, où ϵ désigne une pile vide. $\mathcal{CF}$ désigne l'ensemble des configurations.

Soient *empty*, *top*, *pop*, et *push* des fonctions opérant sur les piles avec la sémantique usuelle : *empty* retourne un booléen qui indique si une pile est vide, *top* retourne le premier élément d'une pile non vide, *pop* retourne le premier élément et dépile la pile et *push* empile un nouvel élément sur la pile.

La définition suivante établit des conditions suffisantes pour assurer qu'une transition est valide dans une configuration donnée. Les transitions étiquetées avec un symbole de fonction ou une variable ordinaire sont traitées comme dans le cas standard, alors que les transitions étiquetées par $\diamond, \#$, par une variable arithmétique ou par une équation $N = 0$ sont considérées comme des ϵ-transitions (dans le sens où il n'y a pas de conditions sur l'entrée). De plus, l'état atteint doit être marqué *quand cela est possible*.

Définition 67. Soit $\mathcal{C} = (q, \mathtt{W}, \mathtt{BT}, \mathtt{H}, \mathtt{Cnt}, \mathtt{Ctx}, \mathtt{L}, \mathrm{E}, mk)$ une configuration. Une transition $(q, \mu) \mapsto p$ est dite *valide* dans $\mathcal{C}$ si et seulement si : ■

1. $\delta(q, \mu) = p$ et une des conditions suivantes est valide :
   - $\mu \in \{\diamond, \#\} \cup V_N \cup V_N^0$.
   - $\mu \in V_X$, $top(\mathtt{W}) = t$ où $t \in \Xi$ et $t \notin \Xi_\diamond$.
   - $\mu = f$, $top(\mathtt{W}) = t$ où $t \in (\Xi \cup \Xi_\diamond)$ et t est de la forme $f(t_1, \ldots, t_n)$ ou de la forme $f(t_1, \ldots, t_n)^N.s$.

2. Ou bien $p \in mk$ ou bien $\Lambda \cap mk = \emptyset$ où Λ est un ensemble d'états définis par :
   - Λ est l'ensemble de *tous* les successeurs de q si q n'apparaît pas juste après une #-transition et sinon,
   - l'ensemble de tous les successeurs p' de q tels que $(q, \nu) \mapsto p'$ et $top(\mathtt{L}) \neq (p', \nu)$ (ainsi, le premier état déjà marqué de la boucle du contexte itératif est ignoré et on ne considère que les états appartenant aux termes de base).

La fonction `FindGeneralisation` implémente l'algorithme de filtrage.

Function `FindGeneralisation`($\mathcal{C}$)

input: $\mathcal{C} = (q, \mathtt{W}, \mathtt{BT}, \mathtt{H}, \mathtt{Cnt}, \mathtt{Ctx}, \mathtt{L}, \mathrm{E}, mk)$ la configuration pour laquelle il faut trouver une généralisation

$IG = (S, \Sigma, \delta, F)$ `/* le graphe d'index */`

output: $(\theta_{IG}(p), \mathrm{E})$, une solution du problème de recherche de généralisation. $\theta_{IG}(p)$ est le terme qui correspond à l'état terminal et E sont les contraintes sur les variables dans $\theta_{IG}(p)$.

if $(empty(\mathtt{W}))$ *and* $(q \in F)$ **then**
  `/* On est dans un état acceptant */` ;
  return $(\theta_{IG}(q)$,*E)* ;
else if *pas de transition valide* $(q, \mu) \mapsto p$ *in* $\mathcal{C}$ **then**
  fail ;
else
  $\mathcal{C}_n$ = `ComputeNextConfiguration`($\mathcal{C}$) ;
  `FindGeneralization`($\mathcal{C}_n$) ;

Elle utilise la fonction `ComputeNextConfiguration` qui retourne les configurations produites par toutes les transitions valides possibles.

`ComputeNextConfiguration` est non-déterministe : elle retourne une configuration arbitrairement choisie. Cette solution a été retenue pour la présentation théorique pour des raisons de simplicité, mais en pratique il faut évidemment utiliser un mécanisme de pile et de retour-arrière pour explorer toutes les branches possibles.

La gestion de la pile `Cnt` est assurée par la procédure `ManageCounters`.

Quand nous lisons un terme (et non un contexte), nous empilons sur `Ctx` son type : 0 s'il s'agit d'un $\mathcal{F}$-terme et 1 si c'est un N-terme. Si ce terme est un $\mathcal{F}$-terme, nous stockons également l'arité du symbole de tête ainsi que le nombre d'arguments restants. Un $\mathcal{F}$-terme est entièrement traité quand tous ses arguments le sont. Il est alors dépilé de `Ctx`. Un N-terme en revanche n'est entièrement traité que si son contexte itéré et son terme de base le sont. La procédure `PushOverOpenTerm` est utilisée pour gérer Ctx.

La gestion des N-termes est simple : ils sont dépilés de la pile quand une N-transition est rencontrée. La gestion des $\mathcal{F}$-termes est légèrement plus difficile. Quand nous lisons une constante ou une variable, le compteur des arguments restant au sommet de `Ctx` est décrémenté. Quand ce compteur atteint 0, le terme est dépilé de la pile. Cette opération est répétée jusqu'à ce que la pile soit vide ou qu'un N-terme soit trouvé. Quand un terme est dépilé de `Ctx` le nouveau compteur des arguments restants au sommet de `Ctx` doit être récursivement décrémenté. La procédure `DiscardArgOfOpenTerm` implémente cette opération.

Exemple 31. L'exemple suivant illustre l'utilité de `Ctx`. Nous considérons le terme indexé $t = f(\diamond)^N.a$ et le terme en entrée (à filtrer) $s = f(f(\ f(f(\diamond))^M.a))$. Le graphe d'index de t contient une boucle pour la partie itérative $f(\diamond)$. La première fois que la #-transition correspondante est

Function `ComputeNextConfiguration`($\mathcal{C}$)

input: $\mathcal{C}$: la configuration à traiter
`/*` $\mathcal{C} = (q, \mathtt{W}, \mathtt{BT}, \mathtt{H}, \mathtt{Cnt}, \mathtt{Ctx}, \mathtt{L}, \mathtt{E}, mk)$ `*/` ;
output: $\mathcal{C}_n$: une configuration suivante
choose une transition valide $(q, \mu) \mapsto p$;
$t = pop(\mathtt{W})$; `/* Le terme courant à traiter */` ;
$q = p$; $mk = mk \cup \{p\}$;
switch μ **do**
  case $\mu \in V_X$
    $\mathrm{E} = \mathrm{E} \cup \{\mu = t\}$;
    `DiscardArgOfOpenTerm(Ctx)` ;
    if *E n'est pas satisfaisable* **then**
      fail
  case $\mu = \diamond$
    $push(t, \mathtt{H})$;
  case $\mu = \#$
    `ManageCounters(Cnt,`q`,Ctx)` ;
    $r = pop(\mathtt{H})$;
    if $r = \diamond$ **then**
      $(s, N) = top(\mathtt{BT})$; `/* Terme de base et exposant */`
      $(r, f) = pop(\mathtt{L})$;
      $push(s, \mathtt{W})$; `/* Le prochain terme à traiter est` s `*/`
      if $\delta(p, f) \neq r$ **then**
        fail
    else
      $push(r, \mathtt{W})$;
  case $\mu \in V_N$
    $(s, N) = pop(\mathtt{BT})$;
    $(r, a, b) = pop(\mathtt{Cnt})$;
    $\mathrm{E} = \mathrm{E} \cup \{\mu = a.N + b\}$;
    $pop(\mathtt{Ctx})$;
    `DiscardArgOfOpenTerm(Ctx)` ;
    if *E n'est pas satisfaisable* **then**
      fail
  case $\mu \in V_N^0$
    $\mathrm{E} = \mathrm{E} \cup \{\mu = 0\}$;
    if *E n'est pas satisfaisable* **then**
      fail
  case $\mu \in \mathcal{F}$ `/*` $t = f(t_1, ..., t_k)$ `or` $t = f^N(t_1, ..., t_k).s$ `*/`
    $push(t_k, \mathtt{W}); push(t_{k-1}, \mathtt{W}); \cdots; push(t_1, \mathtt{W})$;
    if $t \notin \Xi_\diamond$ **then**
      `PushOverOpenTerm(`t`,Ctx);`
    if $t = f^N(t_1, ..., t_k).s$ **then**
      $push((p, f), \mathtt{L})$;
      $push((s, N), \mathtt{BT})$;
$\mathcal{C}_n = (q, \mathtt{W}, \mathtt{BT}, \mathtt{H}, \mathtt{Cnt}, \mathtt{Ctx}, \mathtt{L}, \mathrm{E}, mk)$;
return $\mathcal{C}_n$;

Procedure `ManageCounters(Cnt,`q`,Ctx)`

input: `Cnt` la pile gérant les a- et b-compteurs; q l'état courant; `Ctx` la pile des contextes ouverts
result: Modifie `Cnt` en fonction du type des termes traités et de si c'est la première visite de l'état courant ou pas
if $empty(\texttt{Cnt})$ *or* $(top(\texttt{Cnt}) = (p, a, b)$ *and* $p \neq q)$ **then** `/* C'est la première visite de cet état */`
  if $top(\texttt{Ctx}) = (0, x, y)$ **then** `/* le terme courant ouvert est un` $\mathcal{F}$`-terme. Nous devons utiliser le` b`-compteur */`
    $push((q, 0, 1), \texttt{Cnt})$;
  else
    $push((q, 1, 0), \texttt{Cnt})$;
else
  $(p, a, b) = pop(\texttt{Cnt})$;
  if $top(\texttt{Ctx}) = (0, x, y)$ **then** `/* le terme courant ouvert est un` $\mathcal{F}$`-terme. Nous devons utiliser le` b`-compteur */`
    $push((p, a, b + 1), \texttt{Cnt})$; `/* nous incrémentons le` b`-compteur */`
  else
    $push((p, a + 1, b), \texttt{Cnt})$; `/* nous incrémentons le` a`-compteur */`

Procedure `PushOverOpenTerm(Ctx,`t`)`

input: `Ctx` la pile des contextes; $t \in \Xi$ le prochain terme à traiter
if $t = f^N(t_1, ..., t_n).s$ **then**
  $push((1, 0, 0), \texttt{Ctx})$;
else
  if $t = f(t_1, ..., t_n)$ *and* $n = arity(f) > 0$ **then**
    $push((0, n, n), \texttt{Ctx})$;
  else
    `DiscardArgOfOpenTerm(Ctx)` ; `/*` t `est une constante ou une variable */`

Procedure `DiscardArgOfOpenTerm(Ctx)`

input: `Ctx` la pile des contextes
`/* nous fermons (annulons) itérativement les termes simples ouverts et imbriqués qui ont un seul argument restant */` ;
while $\exists n \in \mathbb{N},\ top(\texttt{Ctx}) = (0, n, 1)$ **do**
  $pop(\texttt{Ctx})$;
if $top(\texttt{Ctx}) = (0, n, d)$ *where* $(d > 1)$ **then**
  $(0, n, d) = pop(\texttt{Ctx})$;
  $push((0, n, d - 1), \texttt{Ctx})$;

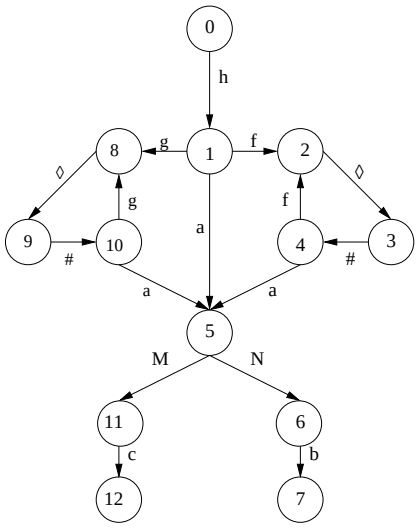

FIG. 6.4 – Indexation de $h(f(\diamond)^N.a, b)$ et $h(g(\diamond)^M.a, c)$

atteinte, le terme courant est $f(f(\ f(f(\diamond))^M.a))$. Ce terme n'est pas un N-terme. Par conséquent, nous devons utiliser le b-compteur. Le terme courant est encore déplié, et le résultat est le terme $f(\ f(f(\diamond))^M.a)$. Le b-compteur est encore utilisé puisqu'il s'agit d'un $\mathcal{F}$-terme. Après le prochain dépliage, quand nous arrivons au #, le terme courant est cette fois un N-terme $f(f(\diamond)))^M.a$, ainsi le a-compteur doit être utilisé. Le rôle de `Ctx` est de garder et de suivre le type du terme courant pour déterminer quel compteur utiliser (a ou b).

En retournant à l'exemple, juste avant la boucle, nous empilons sur `Ctx` le tuple $(0, 1, 1)$. Ceci signifie que nous sommes en train de traiter un $\mathcal{F}$-terme (0), avec un symbole de tête d'arité 1 et qu'il reste un argument à traiter. Quand le # est atteint, nous trouvons au sommet de `Ctx` un tuple $(0, x, y)$ indiquant qu'il faut utiliser le b-compteur. Le prochain terme à traiter est $f(f(f(\diamond))^M.a)$ et c'est encore un $\mathcal{F}$-terme. Nous empilons sur `Ctx` le tuple $(0, 1, 1)$. Dans la #-transition, nous utilisons encore le b-compteur (pour les même raisons que lors de la première boucle). Le prochain terme à traiter est le N-terme $f(f(\diamond))^M.a$, ainsi, nous empilons sur `Ctx` le tuple $(1, 0, 0)$. Quand nous arrivons à la #-transition, le sommet de `Ctx` indique qu'il faut utiliser cette fois le a-compteur. Le prochain terme à traiter est le contexte $f(\diamond)$, par conséquent, aucune information n'est empilée sur `Ctx` et le sommet de la pile contient toujours $(1, 0, 0)$. Après une nouvelle boucle, nous arrivons à la #-transition et nous utilisons encore le a-compteur. Ensuite il faut traiter le trou $\diamond$ qui est enlevé de `Ctx`. Le trou $\diamond$ signifie que le contexte itéré est entièrement traité. il faut considérer ensuite le terme de base a. Il s'agit d'une constante et donc il faut l'ignorer. Le sommet de `Ctx` n'est pas affecté car il correspond à un N-terme. Finalement nous arrivons à une N-transition, la pile `Ctx` est dépilée et la procédure `DiscardArgOfOpenTerm` est invoquée. *top*(`Ctx`) correspond successivement à $(0, 1, 1)$ et $(0, 1, 1)$ puis `Ctx` devient vide.

La remarque suivante explique pourquoi nous avons besoin de dupliquer le graphe d'index du terme de base à la section 6.3. ■

Remarque 1. Supposons que nous utilisions un seul graphe d'index pour le terme de base (en employant les notations de la section 6.3, nous utilisons IG_s dans les deux cas au lieu de IG_s^0) et supposons que nous devions indexer les deux termes $h(f(\diamond)^N.a, b)$ et $h(g(\diamond)^M.a, c)$. Le graphe d'index résultant est présenté dans la figure 6.4 page 91.

Si nous essayons de filtrer le terme $h(g(\diamond)^N.a, b)$ qui n'est instance ni de $h(f(\diamond)^N.a, b)$ ni de

$h(g(\diamond)^M.a, c)$, le résultat sera positif en suivant par exemple le chemin $0 \to 1 \to 8 \to 9 \to 10 \to 5 \to 6 \to 7$. L'algorithme de généralisation n'est donc pas correct si nous identifions IG_s et IG_s^0. La raison de ce comportement indésirable est que si nous utilisons un seul graphe d'index pour le terme de base, alors les graphes index de deux N-termes qui ont le même terme de base mais des contextes inductifs complètement différents vont partager un état commun (5 dans l'exemple précédent) qui peut être atteint par deux chemins différents.

Ce problème est similaire à celui qui nous a poussé à utiliser des états marqués, en l'occurrence l'existence d'états accessibles par deux chemins différents (par exemple, l'état 5 précédent). La solution adoptée dans ce cas est différente. En fait, les marques sont utiles pour les états apparaissant à l'intérieur des boucles où les états communs peuvent être visités plusieurs fois. Mais si l'état est à l'extérieur d'une boucle (comme c'est le cas dans cet exemple), les marques ne permettent pas de prévenir les incohérences puisque l'état sera visité une seule fois. La seule solution est de distinguer entre le cas où le terme de base est filtré directement (sans avoir traité le contexte itératif) et le cas où il est filtré après l'avoir traité. Ainsi, même si deux N-termes partagent un terme de base commun, ils ne vont partager que les états appartenant au graphe d'index du terme de base commun.

Finalement, l'utilité de la pile `L` est illustrée à travers l'exemple suivant. ■

Exemple 32. Soient les termes $t = f(g(\diamond)^N.a)$ et $s = f(g(\diamond))^N.a$. Il apparaît clairement que t n'est pas une généralisation de s. Cependant, si nous représentons les deux termes sans parenthèse et sans la transition qui correspond à la boucle, nous obtenons la même représentation, en l'occurrence $fg \diamond \#aN$ et $fg \diamond \#aN$. Pour résoudre ce problème, nous avons besoin de prendre en considération l'état pointé par la #-transition. En utilisant ce état, les deux représentations peuvent être distinguées si nous connaissons l'état initial de la boucle et le symbole de tête correspondant. C'est la raison pour laquelle la pile `L` est utilisée. Dans notre exemple, après avoir déplié le contexte itératif $f(g(\diamond))$, le sommet de la pile `L` contient le couple (p, f) or dans le graphe d'index, il n'y a aucune transition valide étiquetée avec f (le contexte itéré de l'index est $g(\diamond)$ et donc la seule transition valide est étiquetée avec g) et le filtrage échoue. ■

La complexité de la fonction `FindGeneralisation` est clairement linéaire par rapport à la taille du terme à filtrer.

Par construction des index de graphes, si IG est un graphe d'index et t un terme éligible alors pour qu'il existe un état S dans IG et une substitution σ tels que $\theta_{IG}(S)\sigma \downarrow = t$ il faut et il suffit qu'il existe une exécution de $FindGeneralisation(\mathcal{C}_0)$ qui retourne $(\theta_{IG}(S), \mathrm{E})$, où σ est une solution de E.

Chapitre 7

Implémentation et expérimentations

C'est par l'expérience que la science et l'art font leur progrès chez les hommes.

ARISTOTE

Dans ce chapitre nous présentons le démonstrateur automatique DEI qui permet d'effectuer des inférences directement sur des I-clauses. DEI a été implémenté comme une extension du démonstrateur automatique E-prover [Sch04, Scha]. Nous avons choisi E-prover comme base de développement parce qu'il jouit d'une bonne notoriété, il est largement distribué, son code source est disponible et en plus il est très performant. Notre objectif à travers ce travail était principalement motivé par le besoin de permettre des expérimentations pratiques en présence de I-clauses. Ceci peut aider indéniablement à pousser les recherches sur le sujet en détectant d'autres applications potentielles de la schématisation de termes et en améliorant la compréhension et la maîtrise de ces formalismes (au moins celui des I-termes).

Code source

Le code source du système DEI est disponible sous licence GPL à partir de la page http ://capp.imag.fr/dei.html. Il est fondé sur la version 0.999 − 004 « Longview2 » de E-prover. Le codage des extensions a été réalisé en ANSI C étant donné que le système d'origine (E-prover) est codé dans ce langage. Nous avons utilisé le compilateur gcc (versions 4.2.4 et 4.3) et nous avons testé avec succès le système sur une plateforme GNU Linux x86 (Ubuntu 8.04 Hardy Heron et Ubuntu 9.04 - Jaunty Jackalope).

7.1 Syntaxe en entrée

E-prover peut utiliser deux langages en entrée :

1. Une adaptation du langage LOP [Schb] comme syntaxe en entrée par défaut[1].
2. Le langage TPTP [SSCVG06] (Thousands of Problems for Theorem Provers) utilisé pour définir une librairie de problèmes de tests pour les démonstrateurs automatiques [SS, SS96, SS95].

[1] E prover partage avec le démonstrateur SETHEO [LGM+, LSBB92] le langage LOP.

```
CLAUSE        ::= AXIOME
                |   REQUETE
                |   REGLE

AXIOME        ::= <- PREMISSES .
                |   CONCLUSIONS <- PREMISSES .
                |   CONCLUSIONS <- .

REQUETE       ::= ?- PREMISSES .

REGLE         ::= LITTERAL .
                |   LITTERAL :- PREMISSES .

CONCLUSIONS   ::= LITTERAL
                |   LITTERAL ; CONCLUSIONS
                |   LITTERAL OU CONCLUSIONS

OU            ::= |


PREMISSES     ::= LITTERAL
                |   LITTERAL , PREMISSES
                |   LITTERAL ET PREMISSES

ET            ::= &

LITTERAL      ::= ATOME
                |   ~ ATOME

ATOME         ::= CONSTANTE
                |   CONSTANTE§LISTE_TERMES


LISTE_TERMES  ::= ( TERMES )

TERMES        ::= TERME
                |   TERME , TERMES

TERME         ::= CONSTANTE
                |   VARIABLE
                |   LISTE
                |   CONSTANTE§LISTE_TERMES

LISTE         ::= [ ]
                |   [ TERME | LISTE ]
                |   [ TERMES ]

CONSTANTE     ::= MINUSCULE§MOT
                |   NAT

VARIABLE      ::= MAJUSCULE§MOT
                |   _§MOT

MOT           ::= §
                |   MOT§MINUSCULE
                |   MOT§MAJUSCULE
                |   MOT§CHIFFRE
                |   MOT§_

MINUSCULE     ::= a | b | c | ... | x | y | z

MAJUSCULE     ::= A | B | C | ... | X | Y | Z

ENTIER        ::= CHIFFRE
                |   CHIFFRE^ENTIER

CHIFFRE       ::= 0 | 1 | 2 | ... | 7 | 8 | 9

COMMENTAIRE   ::= /*§!§*/
                | \N§#§!
                | \T§%§!
```

FIG. 7.1 – Syntaxe de E prover

Dans le système DEI nous utilisons une extension de la syntaxe LOP[2]. Ainsi, dans la figure 7.1, nous donnons une version « simplifiée » de la syntaxe LOP de E prover. Pour plus de détails (notamment la version complète de la syntaxe), le lecteur pourra consulter [Scha]. Le méta-symbole `§` désigne la concaténation de chaînes, en particulier, `§` utilisé tout seul désigne la chaîne vide, le méta-symbole `!` désigne une chaîne arbitraire ne contenant pas le symbole `/`, `\N` désigne un retour à la ligne et `\T` désigne une tabulation.

Dans notre extension, nous avons ajouté les règles nécessaires pour formaliser des I-termes (I-clauses) en entrée. En particulier nous avons modifié la règle de définition des termes `TERME : :=` pour tenir compte des I-termes. Ainsi, un I-terme est défini par la syntaxe `$iterm(TERME,TERME)^{EXP}` où `EXP` est une nouvelle règle de production pour décrire les exposants des I-termes. Nous utilisons donc le mot clé `$iterm` comme symbole (spécial) de fonction pour introduire des I-termes. Ce symbole de fonction prend deux arguments : le contexte itératif où le trou est considéré comme un symbole de constante et le terme de base. Un contrôle supplémentaire est nécessaire pour s'assurer que le contexte itératif contient exactement une seule occurrence d'un trou non appartenant à un I-terme. L'exposant est une combinaison linéaire de variables arithmétiques pondérées par des coefficients entiers. Ces exposants peuvent être représentés par les règles de production décrites dans la figure 7.2.

[2]L'extension de la syntaxe TPTP est également possible bien évidemment.

```
EXP          ::= NAT
               | NAT + VARS_ARITH
VARS_ARITH   ::= VARIABLE
               | NAT . VARIABLE + VARS_ARITH
```

FIG. 7.2 – Définition des I-termes

Par exemple, 3 + 2.N1 +4.N2 est un exposant valable dans la syntaxe de DEI. Il n'est pas nécessaire de vérifier que les variables arithmétiques apparaissant dans un exposant ne sont pas également utilisées en tant que variables standards. En effet, l'implémentation utilisée sépare les variables standards de celles arithmétiques même si en entrée le même nom de variable est utilisée : Les variables standards sont automatiquement transformées en variables s'appelant X1, X2, ... alors que les variables arithmétiques sont transformées en variables s'appelant N1, N2, ... Finalement le trou est représenté par le caractère `@`.

Exemple 33. `$iterm(s(@),0)^{1}`, `$iterm(s(@),0)^{N1}` et `$iterm(s(@),0)^{1+N1+3.N2}` sont des exemples de I-termes représentés dans la syntaxe de DEI. ■

7.2 DEI en action

Dans cette section nous présentons un exemple simple d'utilisation de DEI, pour illustrer les possibilités du langage (contextes inductifs contenant des variables, variables arithmétiques partagées, etc). Des expérimentations plus complètes sont présentées à la section 7.4.

Soit l'ensemble de I-clauses S :

$$S = \left\{ \begin{array}{c} P(f(x, g(y, \diamond))^n.u, s(\diamond)^n.0) \vee \neg P_1(u) \\ P_1(a) \\ Q(g(u, f(v, \diamond)^n.g(x, v)), s(\diamond)^n.0) \vee \neg Q_1(x) \\ Q_1(b) \\ R(g(y, x), z) \vee \neg P(x, z) \\ \neg Q(x, y) \vee \neg R(x, y) \end{array} \right.$$

S est insatisfaisable comme le montre la dérivation suivante :

(1)	$P(f(x, g(y, \diamond))^n.u, s(\diamond)^n.0) \vee \neg P_1(u)$			
(2)	$P_1(a)$			
(3)	$Q(g(u, f(v, \diamond)^n.g(x, v)), s(\diamond)^n.0) \vee \neg Q_1(x)$			
(4)	$Q_1(b)$			
(5)	$R(g(y, x), z) \vee \neg P(x, z)$			
(6)	$\neg Q(x, y) \vee \neg R(x, y)$			
(7)	$P(f(x, g(y, \diamond))^n.a, s(\diamond)^n.0)$	$RES(1, 2)$	$\sigma =$	$\{u \to a\}$
(8)	$Q(g(u, f(v, \diamond)^n.g(b, v)), s(\diamond)^n.0)$	$RES(3, 4)$	$\sigma =$	$\{x \to b\}$
(9)	$R(g(u, f(x, g(y, \diamond))^n.a), s(\diamond)^n.0)$	$RES(5, 7)$	$\sigma =$	$\{z \to s(\diamond)^n.0, x \to f(x, g(y, \diamond))^n.a\}$
(10)	$\neg Q(g(u, f(x, g(y, \diamond))^n.a), s(\diamond)^n.0)$	$RES(6, 9)$	$\sigma =$	$\{y \to s(\diamond)^n.0, x \to f(x, g(y, \diamond))^n.a\}$
(11)	$\square$	$RES(8, 10)$	$\sigma =$	$\{v \to a, y \to b, x \to v, n \to 1\}$

La figure 7.3 donne un codage du problème S dans le langage d'entrée de DEI.

```
p($iterm(f(X,g(Y,@)),U)^{N}  ,   $iterm(s(@),0)^{N} ) <- p1(U).
p1(a).
q($iterm(g(U,f(V,@)),g(X,V))^{M}  , $iterm(s(@),0)^{M}) <- q1(X).
q1(b).
r(g(Y,X),Z) <- p(X,Z).
<- q(X,Y), r(X,Y).
```

FIG. 7.3 – Codage du problème S

L'exécution de DEI sur ce problème retourne le résultat affiché dans la figure 7.4.

7.3 Implémentation

Pour préserver la modularité et permettre des extensions futures, nous avons essayé de restreindre autant que possible les modifications au niveau du code d'origine. DEI implémente l'algorithme d'unification des I-termes présenté au chapitre 3 et notre implémentation bénéficie de la technique de partage de termes et plus particulièrement de variables utilisée par le code de E-prover. Dans cette technique les termes sont représentés sous la forme d'un arbre où les constantes et les variables ayant le même nom sont représentées par un noeud unique de l'arbre.

Ceci permet (entre autres) d'éviter la résolution du système diophantien de l'algorithme d'unification. En effet, le partage des variables permet de résoudre à la volée les contraintes arithmétiques. La difficulté majeure est que comme rappelé au chapitre 3 l'unification des I-termes retourne plusieurs unificateurs les plus généraux. Ainsi, une adaptation du code d'origine basé sur l'unicité de l'unificateur est nécessaire.

Même si la syntaxe en entrée permet de coder des exposants sous la forme de combinaison linéaire de variables arithmétiques, l'implémentation transforme systématiquement ces termes en des termes avec exposants sous la forme de variables arithmétiques en se basant sur la transformation $\mathcal{T}$ introduite à la section 3.6. Ceci permet d'utiliser le partage de variables même pour les variables arithmétiques. La figure 7.5 illustre le partage de termes entre deux termes $t = f(x, g(x, b))$ et $s = g(g(x, b), x)$. Dans cet exemple, t contient deux occurrences de la variable x cependant, dans la représentation de t nous avons une seule instance de x qui est partagée entre les différentes occurrences de x. De même, le sous-terme $g(x, b)$ est partagé entre t et s. Le même principe est utilisé également pour l'extension aux I-termes et notamment les variables arithmétiques comme l'illustre la figure 7.6. Dans cet exemple, la variable arithmétique N est partagée entre les deux N-termes $f(\diamond)^N.a$ et $g(\diamond)^N.a$ notés dans la syntaxe de DEI respectivement $\$iterm(f(\diamond), a)^N$ et $\$iterm(g(\diamond), a)^N$. La constante a et le trou sont également partagés entre les deux. Le lien entre le symbole $\$iterm$ et la variable arithmétique est représenté dans la figure avec une flèche en tirets pour distinguer ce lien avec celui correspondant à la relation entre un symbole de fonction et ses fils directs.

7.4 Expérimentations

Nous présentons dans cette section quelques expérimentations réalisées avec DEI. Nous traitons des problèmes de difficultés diverses et nous effectuons des comparaisons de performances (en

```
hicham@hicham-laptop:~/DEI$ ./PROVER/eprover -t LPO ../exemples/exemple1.pl
# Initializing proof state
# Scanning for AC axioms
#
#p1(a) <- .
#
#q1(b) <- .
#
#r(g(X1,X2),X3) <- p(X2,X3).
#
# <- r(X1,X2), q(X1,X2).
#
#p($iterm(f(X1,g(X2,@)),X3)^N1,$iterm(s(@),0)^N1) <- p1(X3).
#
#p($iterm(f(X1,g(X2,@)),a)^N1,$iterm(s(@),0)^N1) <- .
#
#r(g(X3,$iterm(f(X1,g(X2,@)),a)^N1),$iterm(s(@),0)^N1) <- .
#
#q($iterm(g(X1,f(X2,@)),g(X3,X2))^N1,$iterm(s(@),0)^N1) <- q1(X3).
#
#q($iterm(g(X1,f(X2,@)),g(b,X2))^N1,$iterm(s(@),0)^N1) <- .
#
#r(g(X4,$iterm(f(X1,g(X2,@)),X3)^N1),$iterm(s(@),0)^N1) <- p1(X3).
#
# <- r($iterm(g(X1,f(X2,@)),g(b,X2))^N1,$iterm(s(@),0)^N1).

# Proof found!
# SZS status Unsatisfiable
# Initial clauses:                    : 6
# Removed in preprocessing            : 0
# Initial clauses in saturation       : 6
# Processed clauses                   : 11
# ...of these trivial                 : 0
# ...subsumed                         : 0
# ...remaining for further processing : 11
# Other redundant clauses eliminated  : 0
# Clauses deleted for lack of memory  : 0
# Backward-subsumed                   : 0
# Backward-rewritten                  : 0
# Generated clauses                   : 12
# ...of the previous two non-trivial  : 6
# Contextual simplify-reflections     : 0
# Paramodulations                     : 12
# Factorizations                      : 0
# Equation resolutions                : 0
# Current number of processed clauses : 11
#    Positive orientable unit clauses : 5
#    Positive unorientable unit clauses: 0
#    Negative unit clauses            : 1
#    Non-unit-clauses                 : 5
# Current number of unprocessed clauses: 1
# ...number of literals in the above  : 2
# Clause-clause subsumption calls (NU) : 0
# Rec. Clause-clause subsumption calls : 0
# Unit Clause-clause subsumption calls : 0
# Rewrite failures with RHS unbound   : 0
hicham@hicham-laptop:~/DEI$
```

FIG. 7.4 – Résultat de l'exécution de DEI sur le problème S

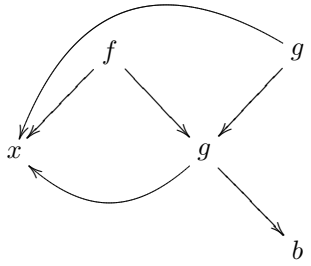

FIG. 7.5 – Exemple de partage de termes standards

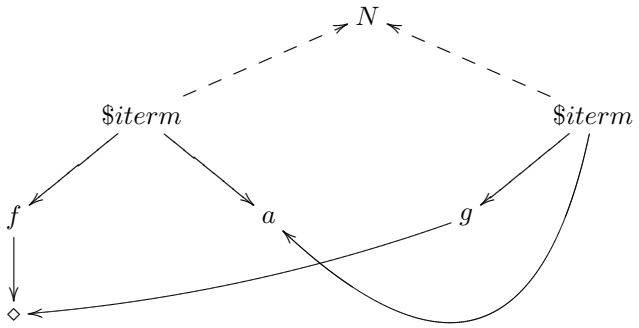

FIG. 7.6 – Exemple de partage de I-termes

terme de temps d'exécution) entre le traitement par DEI et le traitement par E-prover (version 1.2 Balasun) de problèmes équivalents formulés en logique du premier ordre (LPO). Signalons que notre objectif *n'est pas* d'établir une relation de supériorité entre DEI et E (puisque d'une part DEI est fondé sur E et d'autre part les problèmes traités même s'ils sont équivalents ne sont pas identiques) mais de comparer les performances de résolution de (certains) problèmes exprimés avec des I-termes et d'autres (équivalents) exprimés avec des termes standards. Pour chacun des problèmes traités, nous commençons par donner une formulation avec des I-termes puis nous donnons systématiquement une formulation équivalente en logique du premier ordre. Ensuite nous effectuons une série de 100 mesures du temps d'exécution et nous donnons une moyenne arrondie de ces temps.

L'implémentation actuelle de DEI n'intègre pas encore l'algorithme d'indexation des I-termes que nous avons décrit au chapitre 6, donc la détection de la redondance est très rudimentaire (la sémantique des I-termes n'est pas prise en compte, par exemple $p(s(0))$ n'est pas redondante par rapport à $p(s(\diamond)^N.0)$). Ainsi, il est très difficile, voire impossible, de converger en traitant des problèmes satisfaisables (en présence de I-termes). En revanche, puisque le calcul est complet (cf. chapitre 5), la convergence est établie (en théorie) pour des problèmes insatisfaisables (sans égalité). Dans cette partie nous considérons essentiellement des exemples insatisfaisables pour évaluer expérimentalement les possibilités de DEI. Nous présentons également un exemple (très simple) de problème satisfaisable intéressant puisqu'il s'agit d'un problème pour lequel E n'arrive pas à converger contrairement à DEI qui converge rapidement.

Les expérimentations présentées dans cette section on été réalisées sur une machine avec processeur de type Intel(R) Pentium(R) Dual Core T2330 cadencé à 1.60GHz, disposant d'un cache de 1Mo, d'une mémoire RAM de 2 Go et tournant sous Ubuntu 9.04 Jaunty Jackalope.

Les exemples formalisés dans la syntaxe de DEI ainsi que les résultats d'exécution peuvent être retrouvés sur le lien https ://sites.google.com/site/hichambensaid/home.

7.4.1 Exemple satisfaisable

Nous commençons par considérer un exemple (très simple) satisfaisable :

$$\begin{cases} \neg p(s(s(x))) \vee \neg p(s(x)) \vee \neg p(x) \\ p(s(\diamond)^{3.N}.0) \end{cases}$$

La première clause de ce problème code le fait qu'on ne peut pas avoir 3 entiers successifs ayant une certaine propriété. La seconde indique que cette propriété est vraie pour tous les multiples de 3. Ce problème est clairement satisfaisable. En effet il suffit que $p(s(\diamond)^{3.n+1}.0)$ soit évaluée à faux pour chaque $n \in \mathbb{N}$. DEI termine sur cet ensemble et donc prouve sa satisfaisabilité avec un temps moyen de 31 ms. En revanche E n'a pas été capable de converger sur la version équivalente en logique du premier ordre formalisée par :

$$\begin{cases} \neg p(s(s(x))) \vee \neg p(s(x)) \vee \neg p(x) \\ p(g(n, g(n, g(n, 0)))) \\ g(0, x) \approx x \\ g(s(y), x) \approx s(g(y, x)) \end{cases}$$

Cet exemple illustre l'intérêt d'utiliser l'algorithme d'unification des I-termes.

7.4.2 Exemples insatisfaisables

Dans la suite nous ne considérons que des exemples insatisfaisables.

Exemple P_1

Dans cet exemple, nous proposons un codage avec des I-termes des puissances de 2 et nous vérifions pour plusieurs valeurs de $n \in \mathbb{N}$ que $s(\diamond)^k.0$ est une puissance de 2, où $k = 2^n$ en procédant par réfutation. Nous donnons également une formulation du même problème en logique du premier ordre et nous comparons les performances des deux approches. Le codage avec des I-termes est :

$$\begin{cases} p(s(0)) \\ p(s(\diamond)^{2.N}.0) \vee \neg p(s(\diamond)^N.0) \\ \neg p(s(\diamond)^k.0) \end{cases} \quad n \in \mathbb{N}, k = 2^n$$

La formulation équivalente en logique du premier ordre est :

$$\begin{cases} p(s(0)) \\ p(g(n, g(n, 0))) \vee \neg p(g(n, 0)) \\ g(0, x) \approx x \\ g(s(y), x) \approx s(g(y, x)) \\ \neg p(s^k(0)) \end{cases} \quad i \in \mathbb{N}, k = 2^i$$

Le tableau 7.1 donne un comparatif des temps d'exécution pour $k \in \{2, 4, 8, 16, 32, 512, 1024, 2048\}$. Comme nous pouvons le remarquer sur le tableau, E est plus rapide que DEI pour toutes les valeurs de k jusqu'à 1024. Dans ce cas, nous pensons que la complexité introduite dans DEI pour gérer la possibilité d'avoir plusieurs unificateurs est pénalisante par rapport à E. La parité n'est atteinte que pour $k = 2048$ où la longueur des dérivations est beaucoup plus courte, ce qui compense le premier facteur.

k	Temps d'exécution du problème *avec* I-termes avec *DEI*	Temps d'exécution du problème *sans* I-termes avec *E 1.2*
2	$28ms$	$12ms$
4	$29ms$	$12ms$
8	$29ms$	$14ms$
16	$29ms$	$14ms$
32	$32ms$	$16ms$
512	$410ms$	$170ms$
1024	$1,5s$	$880ms$
2048	$5,7s$	$5,8s$

TAB. 7.1 – Comparaison des temps d'exécution sur P_1 entre DEI et E.

k	Temps d'exécution du problème *avec* I-termes avec *DEI*	Temps d'exécution du problème *sans* I-termes avec *E 1.2*
3	$26ms$	$13ms$
9	$27ms$	$17ms$
27	$35ms$	$18ms$
81	$125ms$	$69ms$
243	$900ms$	$1,2s$

TAB. 7.2 – Comparaison des temps d'exécution sur P_2 entre DEI et E.

Exemple P_2

Nous proposons également un exemple similaire au problème P_1 mais avec des contextes inductifs plus compliqués. Le problème P_2 peut être formalisé par :

$$\begin{cases} p(f(a,b)) & & & \\ p(f(a,\diamond)^{3.N}.b) & \vee & \neg p(f(a,\diamond)^{N}.b) & \\ \neg p(f(x,\diamond)^{k}.f(y,\diamond)^{k}.f(z,\diamond)^{k}.b) & & & n \in \mathbb{N}, k = 3^n \end{cases}$$

Une formulation en logique du premier ordre est :

$$\begin{cases} p(f(a,b)) & & & \\ p(g(N,a,g(N,a,g(N,a,b)))) \vee \neg p(g(N,a,b)) & & & \\ \neg p(g(s^k(0),x,g(s^k(0),y,g(s^k(0),z,b)))) & & & n \in \mathbb{N}, k = 3^n \\ g(0,x,y) & \approx & y & \\ g(s(N),x,y) & \approx & f(x,g(N,x,y)) & \end{cases}$$

Dans cet exemple, le prédicat p code les termes du type $f(a,\diamond)^{3^n}.b$ où $n \in \mathbb{N}$ (i.e. $p(t)$ ssi $\exists n \in \mathbb{N}, t = f(a,\diamond)^{3^n}.b \downarrow$) et $g(N,x,y)$ permet de coder le I-terme $f(x,\diamond)^N.y$.

Le tableau 7.2 donne un comparatif des temps d'exécution pour $k \in \{3, 9, 27, 81, 243\}$ soit $\{3^1, 3^2, 3^3, 3^4, 3^5\}$. Comme nous pouvons le remarquer sur le tableau, E est plus rapide que DEI jusqu'à $k = 81$. Pour $k = 243$, DEI devient plus rapide.

Exemple P_3

Cet exemple formalise le fait qu'un entier est une puissance d'un autre. Nous montrons comme pour l'exemple précédent en procédant par réfutation pour certains entiers k, j que k est une

(k, j)	Temps d'exécution du problème *avec* I-termes avec *DEI*	Temps d'exécution du problème *sans* I-termes avec *E 1.2*
$(2, 2)$	$27ms$	$16ms$
$(3, 3)$	$29ms$	$15ms$
$(4, 2)$	$31ms$	$35ms$
$(8, 2)$	$33ms$	$38ms$
$(9, 3)$	$33ms$	$345ms$
$(16, 2)$	$33ms$	$310ms$
$(16, 4)$	$35ms$	$2s$
$(27, 3)$	$39ms$	$41s$
$(64, 4)$	$77ms$	$> 15mn$
$(81, 3)$	$78ms$	$> 15mn$

TAB. 7.3 – Comparaison des temps d'exécution sur P_3 entre DEI et E.

puissance de j. Ce problème peut être formalisé par :

$$\begin{cases} p(s(0), y) \\ p(y, x) \vee \neg p(s(\diamond)^N.x) \vee \neg q(s(\diamond)^N.0, x, y) \\ q(s(\diamond)^N.0, 0, 0) \\ q(s(\diamond)^N.0, s(x), s(\diamond)^N.y) \vee \neg q(s(\diamond)^N.0, x, y) \\ \neg p(s(\diamond)^k.0, s(\diamond)^j.0) \qquad i \in \mathbb{N}, k = j^i \end{cases}$$

Une formulation équivalente en logique du premier ordre est :

$$\begin{cases} p(s(0), y) \\ p(y, x) \vee \neg p(g(N, x)) \vee \neg q(g(N, 0), x, y) \\ q(g(N, 0), 0, 0) \\ q(g(N, 0), s(x), g(N, y)) \vee \neg q(g(N, 0), x, y) \\ g(0, x) \approx x \\ g(s(y), x) \approx s(g(y, x)) \\ \neg p(s^k(0), s^j(0)) \qquad i \in \mathbb{N}, k = j^i \end{cases}$$

Dans les deux cas, $p(x, y)$ signifie que x est une puissance de y (i.e. $\exists i \in \mathbb{N}$, $x = y^i$), $q(x, y, z)$ signifie que z est le résultat de la multiplication de x par y (i.e. $z = x \times y$). Ce problème est noté P_3.

Le tableau 7.3 donne un comparatif des temps d'exécution pour $(k, j) \in \{(2, 2), (3, 3), (4, 2), (8, 2), (9, 3), (16, 2), (16, 4), (27, 3), (64, 4), (81, 3)\}$. Comme nous pouvons le remarquer, E est plus rapide que DEI uniquement pour $(k, j) = (2, 2)$. En revanche, pour des valeurs supérieures de k et de j, DEI est plus performant que E (sur ces exemples). Notons que, puisque ce problème est plus difficile que les problèmes P_1 et P_2, la différence entre les deux programmes est évidemment plus marquée.

(i,j,k)	Temps d'exécution du problème *avec* I-termes avec *DEI*	Temps d'exécution du problème *sans* I-termes avec *E 1.2*
$(1,1,1)$	$27ms$	$13ms$
$(1,2,3)$	$27ms$	$19ms$
$(2,4,3)$	$37ms$	$28ms$
$(3,5,2)$	$55ms$	$19ms$
$(6,7,2)$	$55ms$	$190ms$
$(8,9,2)$	$70ms$	$850ms$
$(11,7,3)$	$400ms$	$16,9s$
$(13,5,9)$	$2,2s$	$9,3s$

TAB. 7.4 – Comparaison des temps d'exécution sur P_4 entre DEI et E.

Exemple P_4

Cet exemple (simple mais non trivial) illustre les possibilités de DEI à effectuer des unifications plus compliquées (que celles précédentes) :

$$\begin{cases} p_1(f(a,\diamond,x)^{i.N}.g(y)) \\ p_2(f(x,\diamond,y)^{j.N}.g(c)) \\ p_3(f(x,\diamond,b)^{k.N}.g(y)) \\ p_4(f(x,y,z)) \\ q(f(x,\diamond,y)^{N}.g(z)) \\ \neg q(x) \vee \neg p_1(x) \vee \neg p_2(x) \neg p_3(x) \vee \neg p_4(x) \end{cases}$$

Ce problème est insatisfaisable et la solution revient à trouver un multiple commun de i,j,k où $i,j,k \in \mathbb{N}$ tout en obtenant par unification les valeurs de x,y,z dans la cinquième clause. Une formulation équivalente en logique du premier ordre est :

$$\begin{cases} p_1(\underbrace{h(N,a,h(N,a,\ldots,h(N,a,g(y),x),\ldots,x),x)}_{i \text{ fois}}) \\ p_2(\underbrace{h(N,x,h(N,x,\ldots,h(N,x,g(c),y),\ldots,y),y)}_{j \text{ fois}}) \\ p_3(\underbrace{h(N,x,h(N,x,\ldots,h(N,x,g(y),b),\ldots,b),b)}_{k \text{ fois}}) \\ p_4(f(x,y,z)) \\ q(h(N,x,g(z),y)) \\ \neg q(x) \vee \neg p_1(x) \vee \neg p_2(x) \vee \neg p_3(x) \vee \neg p_4(x) \\ h(0,w,x,z) \approx x \\ h(s(y),x,w,z) \approx f(x,h(y,x,w,z),z) \end{cases}$$

où $h(N,t_1,t_2,t_3)$ code le I-terme $f(t_1,\diamond,t_3)^N.t_2$.

Nous avons testé ce problème pour plusieurs valeurs de i,j,k en comparant les temps d'exécution de DEI et de E. Le tableau 7.4 donne les résultats des comparaisons.

Nous remarquons dans cet exemple que pour les triplets (i,j,k) ayant un plus petit commun multiplicateur assez « grand » DEI est plus performant que E. La complexité de l'unification avec I-termes est compensée dans ce cas par le nombre élevé d'inférences à faire.

n	Temps d'exécution du problème *avec* I-termes avec *DEI*	Temps d'exécution du problème *sans* I-termes avec *E 1.2*
50	$27ms$	$19ms$
100	$21ms$	$47ms$
150	$36ms$	$102ms$
200	$42ms$	$192ms$
300	$62ms$	$510ms$
500	$114ms$	$2s$
700	$230ms$	$8m4s$

TAB. 7.5 – Comparaison des temps d'exécution sur P_5 entre DEI et E.

Exemple P_5

Dans cet exemple nous considérons un prédicat p qui code le fait qu'une liste donnée contient un seul élément (i.e. tous les membres de la liste sont égaux). p peut être définit par $p(cons(x,\diamond)^N.nil)$. Nous testons pour plusieurs entiers n que $p(cons(a,\diamond)^n.nil)$ est vrai où a est une constante. Le problème donné à DEI est donc formalisé par :

$$\begin{cases} p(cons(x,\diamond)^N.nil) \\ \neg p(cons(a,\diamond)^n.nil) \quad n \in \mathbb{N} \end{cases}$$

Une formulation en logique du premier ordre équivalente est :

$$\begin{cases} p(x) \vee \neg q(x,y) \\ q(nil,x) \\ q(cons(x,y),x) \vee \neg q(y,x) \\ \underbrace{\neg p(cons(a,cons(a,\ldots,cons(a,nil))))}_{n \text{ fois}} \end{cases}$$

Le tableau 7.5 donne un comparatif des temps d'exécution pour $n \in \{50, 100, 150, 200, 300, 500, 700\}$.

Comme nous pouvons le constater, E est plus efficace pour $n = 50$ mais pour les valeurs supérieures DEI est plus rapide.

Exemple P_6

Cet exemple illustre la capacité de DEI à détecter des cycles durant la procédure d'unification. La formulation avec des I-termes est :

$$\begin{cases} \neg p(f(x,g(f(a,\diamond))^k.g(y))) \vee \neg p(f(a,g(\diamond))^{2.N}.y)) \quad k \in \mathbb{N} \\ p(f(a,g(\diamond)^N.b) \end{cases}$$

Une formulation équivalente en logique du premier ordre est :

$$\begin{cases} \neg p(f(x,h(s^k(0),a,g(y)))) \vee \neg p(i(N,a,i(N,a,y))) & k \in \mathbb{N} \\ p(i(N,a,b)) & \\ h(0,x,y) \approx y & \\ h(s(N),x,y) \approx g(f(x,h(N,x,y))) & \\ i(0,x,y) \approx y & \\ i(s(N),x,y) \approx f(x,g(i(N,x,y))) & \end{cases}$$

k	Temps d'exécution du problème *avec* I-termes avec *DEI*	Temps d'exécution du problème *sans* I-termes avec *E 1.2*
2	$25ms$	$18ms$
4	$27ms$	$48ms$
6	$30ms$	$395ms$
8	$30ms$	$2s$
10	$30ms$	$11s$
12	$31ms$	$1m43s$

TAB. 7.6 – Comparaison des temps d'exécution sur P_6 entre DEI et E.

Dans cet exemple, $h(N, x, y)$ permet de coder le I-terme $g(x, f(\diamond))^N.y$ et $i(N, x, y)$ permet de coder le I-terme $f(x, g(\diamond))^N.y$.

Le tableau 7.6 donne un comparatif des temps d'exécution pour $k \in \{2, 4, 6, 8, 10, 12\}$.

Nous remarquons que le traitement avec DEI est largement plus performant dans cet exemple que celui avec E.

Exemple P_7

Dans cet exemple, nous proposons un problème qui permet d'engendrer des listes de la forme $[a_1, a_2, \ldots, a_n]$ avec $a_i = 1$ ou 2. La propriété à démontrer est que la somme des éléments de la liste est toujours supérieure à la longueur de la liste. Nous formalisons ce problème par :

$$\begin{cases} p(s(x), cons(s(0), y)) \vee p(s(x), cons(s(s(0)), y)) \vee \neg p(x, y) & \\ p(0, nil) & \\ q(0, nil) & \\ q(cons(s(\diamond)^N.0, x), s(\diamond)^N.y) \vee \neg q(x, y) & \\ \neg p(s(\diamond)^k.0, y) \vee \neg q(y, s(\diamond)^k.x) & k \in \mathbb{N} \end{cases}$$

Dans ce codage, $p(x, y)$ signifie que y est une liste de taille x qui ne contient que des 1 ou des 2 (ou est vide) et $q(x, y)$ signifie que la somme des éléments de la liste y est égale à x.

Une formulation équivalente en logique du premier ordre est :

$$\begin{cases} p(s(x), cons(s(0), y)) \vee p(s(x), cons(s(s(0)), y)) \vee \neg p(x, y) & & \\ p(0, nil) & & \\ q(0, nil) & & \\ q(cons(g(N, 0), x), g(N, y)) \vee \neg q(x, y) & & \\ \neg p(s^k(0), y) \vee \neg q(y, s^k(0)) & & k \in \mathbb{N} \\ g(0, x) & \approx & x \\ g(s(N), x) & \approx & s(g(N, x)) \end{cases}$$

Le tableau 7.7 donne un comparatif des temps d'exécution pour $k \in \{1, 2, 3, 4\}$.

Nous remarquons que le traitement avec DEI est largement plus performant dans cet exemple que celui avec E.

7.4.3 Commentaires

Les exemples (plus ou moins simples) présentés dans cette section ont pour objectif principal de tester le fonctionnement de DEI. Les comparaisons avec E que nous avons présentées contribuent à l'évaluation des performances de DEI. Une première remarque immédiate que nous

k	Temps d'exécution du problème *avec* I-termes avec *DEI*	Temps d'exécution du problème *sans* I-termes avec *E 1.2*
1	$26ms$	$33ms$
2	$32ms$	$290ms$
3	$57ms$	$4s$
4	$383ms$	$> 10mn$

TAB. 7.7 – Comparaison des temps d'exécution sur P_7 entre DEI et E.

pouvons faire est qu'en fonction du type du problème, parfois c'est E qui est le plus rapide et parfois c'est DEI qui l'est même si asymptotiquement, DEI est plus rapide dans tous les exemples que nous avons testés. Une deuxième remarque est que sur les exemples très simples (triviaux) avec des termes de « petites » tailles, les problèmes exprimés en logique du premier ordre standard sont plus rapides à résoudre (même si l'ordre de grandeur des différences de performance se mesure en millisecondes). Pour de tels exemples, le gain en nombre d'inférences réalisées n'est pas significatif et l'utilisation de la machinerie complexe de l'unification des I-termes se révèle pénalisante. En revanche, pour des problèmes plus difficiles (où l'espace de recherche est plus diversifié et de plus grande taille impliquant la nécessité de faire beaucoup plus d'inférences) nous observons parfois des écarts de performance très nets en faveur de DEI. Cela illustre en particulier l'efficacité de l'algorithme d'unification que nous avons implémenté.

Dans le futur, nous prévoyons d'implémenter notre algorithme d'indexation de I-termes pour pouvoir traiter plus d'exemples satisfaisables. Nous pensons que la détection de la satisfaisabilité (par saturation) pourrait constituer un atout majeur en faveur de l'utilisation des I-termes (et de DEI) comme illustré avec l'exemple P_1 sur lequel E ne termine pas (en effet l'indexation est indispensable pour implanter efficacement les règles de simplification, telle que le subsomption, qui sont nécessaires à la terminaison même dans les cas les plus simples). Nous pensons également qu'il pourrait être intéressant comme travaux futurs d'essayer de caractériser (au moins pour certains types de clauses) les problèmes pour lesquels l'utilisation de la logique du premier ordre est plus efficace et ceux sur lesquels une formulation avec des I-termes donne de meilleurs résultats. L'intérêt alors est de transformer le problème initiale vers la formulation la plus adéquate (si elle existe bien entendu) avant de le traiter par un démonstrateur.

7.5 Limites de l'implémentation

La version actuelle de DEI implémente uniquement le calcul $\mathcal{ISC}$. De plus la relation d'ordre ne prend pas en charge les N-termes qui sont considérés comme incomparables avec les autres termes. Nous prévoyons d'étendre la version actuelle en implémentant $\mathcal{IS}_H$, en étendant les ordres aux N-termes et en étendant les fonctionnalités d'indexation.

Chapitre 8

Conclusion et travaux futurs

C'est là en effet un des grands et merveilleux caractères des beaux livres que pour l'auteur ils pourraient s'appeler Conclusions et pour le lecteur Incitations.

Marcel Proust

8.1 Travaux réalisés

Dans ce mémoire nous nous sommes intéressés à l'utilisation de la schématisation de termes en déduction automatique. Notre motivation initiale découlait de notre volonté de contribuer à doter les démonstrateurs automatiques de théorèmes actuels d'améliorations qualitatives et à améliorer la terminaison. Notre contribution se décline en trois axes :

1. Une contribution dans l'analyse des espaces de recherche principalement dans le contexte des preuves par saturation. En particulier, nous avons étudié la détection des régularités dans ces espaces en se basant sur une analyse syntaxique des clauses produites. Nous avons développé l'outil DS3 dans ce sens, qui permet dans certains cas de détecter les régularités et d'inférer par raisonnement inductif des généralisations structurantes de l'espace de recherche construites avec des schématisations de termes. Cela est réalisé en utilisant la puissance de calcul d'un outil de mathématiques formelles (Mathematica). Nous avons montré l'utilité de notre approche avec des exemples issus de trois applications potentielles différentes :
   (a) La caractérisation d'une suite en fonction du rang d'apparition que nous avons illustrée avec un exemple d'énumération des nombres rationnels.
   (b) La détection de la satisfaisabilité et la construction de modèles d'un ensemble de clauses S en particulier dans le cas où S n'admet que des modèles de cardinalité infinie, impossible donc à trouver avec des constructeurs de modèles finis (i.e. les constructeurs de modèles standards).
   (c) La découverte de lemmes surtout dans le contexte des preuves inductives en utilisant la méthode d'induction sans induction. Ces lemmes sont parfois nécessaires pour la terminaison des méthodes de preuve.
   (d) La découverte d'une description structurelle d'un langage à partir d'une grammaire hors-contexte.

2. Nous avons abordé les aspects théoriques de l'intégration d'une schématisation particulière de termes, les I-termes, dans les calculs par saturation les plus utilisés. Il s'agit à notre connaissance du premier travail de ce type.
   - Pour la partie déduction, nous avons démontré que l'extension du calcul de résolution ordonnée avec des I-termes a la propriété de complétude réfutationnelle (pour des ensembles de clauses sans égalité). Nous avons également montré que l'extension directe du calcul de superposition n'a pas cette propriété et nous avons proposé une nouvelle règle d'inférence pour restaurer la complétude réfutationnelle du calcul de superposition en présence d'I-termes.
   - Nous avons également abordé le problème d'élimination de la redondance en proposant un algorithme d'indexation des I-termes. En fait, vu la difficulté du sujet, nous nous sommes limités à l'indexation d'une sous-classe des I-termes.
3. Nous avons également implémenté une extension du démonstrateur automatique E-prover permettant de gérer des I-termes que nous avons appelé DEI. Il s'agit à notre connaissance du premier démonstrateur capable de traiter directement des schémas de termes (naturellement il est possible de traduire les I-termes en logique du premier ordre, mais tous les avantages liés à l'utilisation de ce formalisme sont alors perdus). Cette implémentation a été principalement motivée par le besoin d'expérimenter l'utilité des I-termes pour assurer la convergence dans certaines preuves inductives. Nous espérons que DEI pourra contribuer également à promouvoir l'usage des schémas de termes en déduction automatique à plus grande échelle en permettant d'une part de mieux comprendre ces formalismes et d'autre part de faire des manipulations et des expérimentations qui pourraient aider à mettre en évidence d'autres applications potentielles des I-termes.

8.2 Limites actuelles et perspectives

Les travaux réalisés dans le cadre de cette thèse sont loin d'être finis, le temps limité d'une thèse ne l'aurait pas permis. En particulier, comme expliqué au chapitre 4, DS3 est incapable de trouver une généralisation pertinente dans le cas où la suite de termes (clauses) est constituée d'une imbrication de plusieurs sous-suites. Nous pensons qu'il serait intéressant de trouver des algorithmes de généralisation plus robustes capables en particulier d'effectuer ce type de décomposition. L'étude expérimentale mérite évidemment d'être poursuivie et approfondie afin de pouvoir caractériser les classes de problèmes pour lesquelles l'intérêt pratique de notre contribution est le plus intéressant. En raison des limites d'expressivité sur les I-termes (en particulier l'absence de variables indicées, cf. ci-dessous), nous pensons qu'une application privilégiée de nos travaux pourrait être la preuve par induction sur des structures simples telles que les entiers (e.g. l'exemple considéré au chapitre 4). De telles preuves sont en général hors de portée des démonstrateurs classiques (même en utilisant l'induction sans induction) car la terminaison ne peut être assurée et même les démonstrateurs par induction spécialisés s'avèrent souvent incapables de générer les lemmes inductifs indispensables à la preuve (cf. [Wal96, Ham07]). Du fait des problèmes théoriques (incomplétude, nécessité d'étendre les techniques d'indexation aux I-termes) et pratiques que nous avons rencontrés, cette étude n'a pu qu'être esquissée durant le temps imparti pour cette thèse.

Une autre limitation concerne notre algorithme d'indexation qui ne peut gérer actuellement qu'une sous-classe des I-termes que nous avons appelée termes éligibles (notée Ξ). En l'absence du prédicat d'égalité, cette restriction n'est pas très contraignante, en revanche quand l'égalité est présente et surtout avec la nouvelle règle d'inférence que nous avons proposée, la stabilité de l'ensemble Ξ n'est plus assurée avec les règles d'inférence : l'application du calcul $\mathcal{IS}_H$ sur

des clauses ne contenant que des termes éligibles ne produit pas forcément des clauses avec exclusivement des termes éligibles.

Concernant DEI, ce dernier n'implémente dans sa version actuelle que le calcul $\mathcal{ISC}$ et ne peut donc garantir la complétude réfutationnelle pour le cas équationnel.

Nous pensons que plusieurs améliorations sont possibles et nécessaires à une plus large diffusion de notre travail. Nous pouvons citer (entre autres) :

- Concernant DEI, l'implémentation de notre algorithme d'indexation ce qui permettrait également d'évaluer ses performances, l'intégration du nouveau calcul $\mathcal{IS}_H$ dans DEI pour restaurer la complétude réfutationnelle et l'étude à travers des tests comparatifs de l'efficacité de ce nouveau calcul, en particulier la nouvelle règle d'inférence H-superposition.
- Des problèmes théoriques se situant dans la continuation directe de ce travail comme l'importance de trouver des algorithmes d'indexation supportant toute la classe des I-termes et offrant plus de possibilités que la recherche de généralisations comme c'est le cas avec la version actuelle et la conception de relations d'ordre spécifiques aux I-termes, qui soient suffisamment restrictives mais en même temps décidables et calculables de manière efficace.
- D'autres travaux généralisant celui-ci comme l'extension des calculs de résolution ordonnée et de superposition à des formalismes de schématisation de termes plus expressifs comme par exemple les grammaires primales.

Par ailleurs, les schématisations de termes présentées dans ce mémoire présentent des limitations. Par exemple, si nous considérons le terme $t = f(X_1, f(X_2, \ldots, X_{n-1}, f(X_n, a)))$, il n'est pas possible de représenter t avec ces schématisations de termes, même si la structure de t présente des similitudes avec les mêmes schématisations. Étudier la possibilité de formaliser ce type de termes en traitant au moins le problème de décidabilité de l'unification nous semble intéressant. Une première tentative dans ce sens – visant à combiner le pouvoir d'expression des I-termes avec celui des automates d'arbre est proposée dans [Pel08].

Finalement, comme évoqué dans [Ara10] et dans le cadre du projet ASAP, l'intégration de la schématisation de termes avec la schématisation de formules pourrait constituer une extension intéressante à la fois du présent travail et de celui effectué par Vincent Aravantinos dans sa thèse. Cela permettrait de définir des logiques permettant de raisonner sur des schémas de formules du premier ordre (les procédures existantes ne traitent que le cas propositionnel).

Bibliographie

[ACP08] V. Aravantinos, R. Caferra et N. Peltier : More flexible term schematisations via extended primal grammars. *In Tenth International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, January 2008.

[ACP09a] V. Aravantinos, R. Caferra et N. Peltier : Complexity of the satisfiability problem for a class of propositional schemata. Research report, available at `http://membres-liglab.imag.fr/aravantinos/`, 2009.

[ACP09b] V. Aravantinos, R. Caferra et N. Peltier : Decidability and undecidability results for propositional schemata. Research report, available at `http://membres-liglab.imag.fr/aravantinos`, 2009.

[ACP09c] V. Aravantinos, R. Caferra et N. Peltier : A DPLL proof procedure for propositional iterated schemata. *In Workshop "Structures and Deduction 2009" (ESSLI)*, pages 24–38, 2009.

[ACP10a] V. Aravantinos, R. Caferra et N. Peltier : Complexity of the satisfiability problem for a class of propositional schemata. *In LATA 2010 (Language, Automata Theory and Applications*, LNCS. Springer, 2010.

[ACP10b] V. Aravantinos, R. Caferra et N. Peltier : A decidable class of nested iterated schemata. *In Automated Reasoning*, volume 6173 de *Lecture Notes in Computer Science*, pages 293–308. Springer Berlin / Heidelberg, 2010.

[ACP10c] V. Aravantinos, R. Caferra et N. Peltier : Regstab : A sat solver for propositional schemata. *In Automated Reasoning*, volume 6173 de *Lecture Notes in Computer Science*, pages 309–315. Springer Berlin / Heidelberg, 2010.

[Acz77] P. Aczel : An Introduction to Inductive Definitions. *In* K. Jon Barwise, éditeur : *Handbook of Mathematical Logic*, pages 739–782. North-Holland, Amsterdam, 1977.

[AHL93] A. Amaniss, M. Hermann et D. Lugiez : Etude comparative des méthodes de schématisation de séquences infinies de termes du premier ordre. Research Report 93-R-114, Centre de Recherche en Informatique de Nancy, 1993.

[AHL97] A. Amaniss, M. Hermann et D. Lugiez : Set operations for recurrent term schematizations. *In TAPSOFT*, pages 333–344, 1997.

[Ara10] V. Aravantinos : *Schémas de formules et de preuves en logique propositionnelle.* Thèse de doctorat, Université de Grenoble, September 2010.

[AS83] D. Angluin et C. H. Smith : Inductive inference : Theory and methods. *ACM Comput. Surv.*, 15(3):237–269, 1983.

[Aub76] R. Aubin : *Mechanizing structural induction.* Thèse de doctorat, University of Edinburgh, 1976.

[Bar77] J. Barwise : *Handbook of Mathematical Logic.* North Holland, 1977.

[Bau98] P. Baumgartner : Hyper Tableaux — The Next Generation. *In* H. de Swaart, éditeur : *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1397 de *LNAI*, pages 60–76. Springer, 1998.

[BCP07] H. Bensaid, R. Caferra et N. Peltier : Towards systematic analysis of theorem provers search spaces : First steps. *In Proc. Wollic'07 (Workshop on Logic, Language, Information and Computation)*, pages 38–52. Springer, July 2007. LNCS 4576.

[BCP09] H. Bensaid, R. Caferra et N. Peltier : DEI : A theorem prover for terms with integer exponents. *In CADE 22 (22nd International Conference on Automated Deduction)*, volume 5663 de *Lecture Notes in Computer Science*, pages 146–150. Springer, 2009.

[BCP10a] H. Bensaid, R. Caferra et N. Peltier : *I*-terms in ordered resolution and superposition calculi : Retrieving lost completeness. *In* S. Autexier, J. Calmet, D. Delahaye, P. Ion, L. Rideau, R. Rioboo et A. Sexton, éditeurs : *Intelligent Computer Mathematics*, volume 6167 de *Lecture Notes in Computer Science*, pages 19–33. Springer Berlin / Heidelberg, 2010.

[BCP10b] H. Bensaid, R. Caferra et N. Peltier : Perfect discrimination graphs : Indexing terms with integer exponents. *In* J. Giesl et R. Hähnle, éditeurs : *IJCAR 2010 (5th International Joint Conference on Automated Reasoning)*, volume 6173 de *Lecture Notes in Computer Science*, pages 369–383. Springer Berlin / Heidelberg, 2010.

[Bee85] M. J. Beeson : *Foundations of constructive mathematics. Metamathematical Studies*, volume 6 de *Modern surveys in Mathematics*. Springer-Verlag, 1985.

[Bel86] E. T. Bell : *Men of Mathematics. The lives and achievements of the great mathematicians from Zeno to Poincaré*. Simon and Schuster, 1986.

[BEL01] M. Baaz, U. Egly et A. Leitsch : Normal form transformations. *In* A. Robinson et A. Voronkov, éditeurs : *Handbook of Automated Reasoning*, volume I, chapitre 5, pages 273–333. Elsevier Science, 2001.

[BG94] L. Bachmair et H. Ganzinger : Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 3(4):217–247, 1994.

[BG01] L. Bachmair et H. Ganzinger : Resolution theorem proving. *In* A. Robinson et A. Voronkov, éditeurs : *Handbook of Automated Reasoning*, pages 19–99. North-Holland, 2001.

[Bib81] W. Bibel : On matrices with connections. *Journal of the Association of Computing Machinery*, 28:633–645, 1981.

[BKR92] A. Bouhoula, E. Kounalis et M. Rusinowitch : SPIKE, an automatic theorem prover. *In LPAR '92 : Proceedings of the International Conference on Logic Programming and Automated Reasoning*, pages 460–462, London, UK, 1992. Springer-Verlag.

[BL84] W.W. Bledsoe et D.W. Loveland : *Automated Theorem Proving after 25 years*, volume 29 de *Contemporary Mathematics*. American Mathematical Society, Providence, RI, USA, 1984.

[BR95] A. Bouhoula et M. Rusinowitch : Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14:14–189, 1995.

[BS95] M. Bernstein et N. J. A. Sloane : Some Canonical Sequences of Integers. *Linear Algebra and its Applications*, 226/228(1–3):57–72, /1995.

[Bun88] A. Bundy : The use of explicit plans to guide inductive proofs. *In Proceedings of CADE 9*. Springer Verlag, 1988. LNCS 310.

[Bun01] A. Bundy : The automation of proof by mathematical induction. *In* John Alan Robinson et Andrei Voronkov, éditeurs : *Handbook of Automated Reasoning*, pages 845–911. Elsevier and MIT Press, 2001.

[BW05] H. Barendregt et F. Wiedijk : The challenge of computer mathematics. *Philosophical Transactions of the Royal Society A*, 363:2351–2375, 2005.

[Caf09] R. Caferra : Une approche simple de la logique pour l'informatique et la formalisation de l'inférence. Polycopié, ENSIMAG, 2009.

[CCM04] C. S. Calude, E. Calude et S. Marcus : Passages of proof. *Bulletin of the EATCS*, (84):167–188, October 2004.

[CH91] H. Chen et J. Hsiang : Logic programming with recurrence domains. *In Automata, Languages and Programming (ICALP'91)*, pages 20–34. Springer, LNCS 510, 1991.

[CHK90] H. Chen, J. Hsiang et H.C. Kong : On finite representations of infinite sequences of terms. *In Conditional and Typed Rewriting Systems, 2nd International Workshop*, pages 100–114. Springer, LNCS 516, 1990.

[CLP04] R. Caferra, A. Leitsch et N. Peltier : *Automated Model Building*, volume 31 de *Applied Logic Series*. Kluwer Academic Publishers, 2004.

[CM04] C. S. Calude et S. Marcus : Mathematical proofs at a crossroad ? *In Theory is Forever (Salomaa Festschrift)*, pages 15–28. Springer-Verlag, 2004. LNCS 3113.

[CN98] H. Comon et R. Nieuwenhuis : Induction = i-axiomatixation+first-order consistency. Rapport technique LSV-98-9, ENS Cachan, october 1998.

[Com91] H. Comon : Disunification : A survey. *In* J.-L. Lassez et G. Plotkin, éditeurs : *Computational Logic : Essays in Honor of Alan Robinson*, pages 322–359. MIT Press, Cambridge, MA, 1991.

[Com94] H. Comon : Inductionless induction. *In* René David, éditeur : *2nd Int. Conf. in Logic For Computer Science : Automated Deduction. Lecture notes*, Chambéry, 1994. Univ. de Savoie.

[Com95] H. Comon : On unification of terms with integer exponents. *Mathematical System Theory*, 28:67–88, 1995.

[Com01] H. Comon : Inductionless induction. *In* A. Robinson et A. Voronkov, éditeurs : *Handbook of Automated Reasoning*, chapitre 14, pages 913–962. North-Holland, 2001.

[Con93] R. L. Constable : Formal theories and software systems : Fundamental connections between computer science and logic. Department of Computer Science Cornell University, April 1993.

[CP00] R. Caferra et N. Peltier : The connection method, constraints and model building. *In Intellectics and Computational Logic*, volume 19 de *Applied Logic Series*, pages 67–84. Kluwer, 2000.

[Der82] N. Dershowitz : Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

[DG79] B. Dreben et W. D. Goldfarb : *The Decision Problem, Solvable Classes of Quantificational Formulas*. Addison-Wesley, 1979.

[DM79] N Dershowitz et Z Manna : Proving termination with multiset orderings. *Commun. ACM*, 22(8):465–476, 1979.

[Fit90] M. Fitting : *First-Order Logic and Automated Theorem Proving*. Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[FLTZ93] C. Fermüller, A. Leitsch, T. Tammet et N. Zamov : *Resolution Methods for the Decision Problem*. LNAI 679. Springer, 1993.

[Gra96] P. Graf : *Term Indexing*, volume 1053 de *Lecture Notes in Computer Science*. Springer, 1996.

[Häh01] R. Hähnle : Tableaux and related methods. *In* A. Robinson et A. Voronkov, éditeurs : *Handbook of Automated Reasoning*, pages 100–178. North-Holland, 2001.

[Ham07] G. W. Hamilton : Distilling programs for verification. *Electron. Notes Theor. Comput. Sci.*, 190(4):17–32, 2007.

[Her94] M. Hermann : Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes. Habilitation, Université de Nancy I, and CRIN-CNRS Inria-Lorraine, Nancy, France, March 1994.

[HG97] M. Hermann et R. Galbavý : Unification of Infinite Sets of Terms schematized by Primal Grammars. *Theoretical Computer Science*, 176(1–2):111–158, 1997.

[HMU00] J. E. Hopcroft, R. Motwani et J. D. Ullman : *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, deuxième édition, 2000.

[KA99] C. Krattenthaler et G. Andrews : Advanced determinant calculus. Rapport technique, Institut für Mathematik der Universität Wien, 1999.

[Kle52] S.C. Kleene : *Introduction to Metamathematics*. North-Holland, 1952.

[LD94] N. Lavrač et S. Džeroski : *Inductive Logic Programming : Techniques and Applications*. Ellis Horwood, 1994.

[Lei97] A. Leitsch : *The resolution calculus*. Springer. Texts in Theoretical Computer Science, 1997.

[LGM+] R. Letz, C. Gresser, M. Moser, A. Wolf et O. Ibens : The SETHEO home page. `http://www4.informatik.tu-muenchen.de/~letz/setheo/`.

[LSBB92] R. Letz, J. Schumann, S. Bayerl et W. Bibel : SETHEO : a High-Performance Theorem Prover. *Journal of Automated Reasoning*, 8(2):183–212, 1992.

[McC92] W. McCune : Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9:147–167, 1992.

[McC95] W. McCune : *Otter 3.0 Reference Manual and Guide*. Argonne National Laboratory, août 1995. Revision A.

[McC10] W. McCune : Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[Mit97] T. M. Mitchell : *Machine Learning*. McGraw-Hill, New York, 1997.

[Pel97] N. Peltier : Increasing the capabilities of model building by constraint solving with terms with integer exponents. *Journal of Symbolic Computation*, 24:59–101, 1997.

[Pel01a] N. Peltier : A General Method for Using Terms Schematizations in Automated Deduction. *In Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, pages 578–593. Springer LNCS 2083, 2001.

[Pel01b] N. Peltier : Introduction à la déduction automatique. Note du cours "Logique et Formalisation du raisonnement (partie II : introduction à la déduction automatique).", DEA ISC. Université Joseph Fourier, Grenoble., 2001.

[Pel08] N. Peltier : A unified view of tree automata and term schematisations. *In Fifth IFIP International Conference On Theoretical Computer Science - TCS 2008*, volume 273 de *IFIP*, pages 491–505. Springer, 2008.

[Plo70] G.D. Plotkin : A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[Plo71] G.D. Plotkin : A Further Note on Inductive Generalization. *In* B. Meltzer et D. Michie, éditeurs : *Machine Intelligence 6*, chapitre 8, pages 101–124. Edinburgh University Press, 1971.

[Pol73] G. Polya : *How to Solve It, a New Aspect of Mathematical Method*. Princeton University Press. Second Edition, 1973.

[Rob65a] J. A. Robinson : Automatic deduction with hyperresolution. *Intern. Journal of Computer Math.*, 1:227–234, 1965.

[Rob65b] J. A. Robinson : A machine-oriented logic based on the resolution principle. *J. Assoc. Comput. Mach.*, 12:23–41, 1965.

[RSV01] I. V. Ramakrishnan, R. C. Sekar et A. Voronkov : Term indexing. *In* John Alan Robinson et Andrei Voronkov, éditeurs : *Handbook of Automated Reasoning*, pages 1853–1964. Elsevier and MIT Press, 2001.

[RV01a] A. Riazanov et A. Voronkov : Vampire 1.1 (system description). *In Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'01)*, pages 376–380. Springer LNCS 2083, 2001.

[RV01b] A. Robinson et A. Voronkov, éditeurs. *Handbook of Automated Reasoning*. North-Holland, 2001.

[Sak97] Y. Sakakibara : Recent advances of grammatical inference. *Theor. Comput. Sci.*, 185(1):15–45, 1997.

[Sal92] G. Salzer : The unification of infinite sets of terms and its applications. *In Logic Programming and Automated Reasoning (LPAR'92)*, pages 409–429. Springer, LNAI 624, July 1992.

[Scha] S. Schulz : The E Equational Theorem Prover. `http://www4.informatik.tu-muenchen.de/~schulz/WORK/eprover.html`.

[Schb] S. Schulz : LOP-syntax for theorem proving applications. `http://www4.informatik.tu-muenchen.de/~schulz/WORK/lop.syntax`.

[Sch04] S. Schulz : System Description : E 0.81. *In* D. Basin et M. Rusinowitch, éditeurs : *Proc. of the 2nd IJCAR, Cork, Ireland*, volume 3097 de *LNAI*, pages 223–228. Springer, 2004.

[Smu68] R. M. Smullyan : *First-Order Logic*. Springer, 1968.

[SS] Ch. Suttner et G. Sutcliffe : The TPTP problem library. Rapport technique, http ://www.cs.miami.edu/ tptp/. V-2.5.0.

[SS95] Ch. B. Suttner et G. Sutcliffe : The TPTP problem library. Rapport technique, TU München / James Cook University, 1995. V-1.2.0.

[SS96] Ch. Suttner et G. Sutcliffe : The TPTP problem library. Rapport technique, TU München / James Cook University, 1996. V-1.2.1.

[SSCVG06] G. Sutcliffe, S. Schulz, K. Claessen et A. Van Gelder : Using the TPTP language for writing derivations and finite interpretations. *In In Ulrich Fuhrbach and Natarajan Shankar, editors, Proc. of the 3rd IJCAR, Seattle, volume 4130 of LNAI*, pages 67–81. Springer, 2006.

[SW83a] J. Siekmann et G. Wrightson : *Automation of Reasoning. Classical Papers on Computational Logic 1957-1966*, volume 1. Springer-Verlag, 1983.

[SW83b] J. Siekmann et G. Wrightson : *Automation of Reasoning. Classical Papers on Computational Logic 1967-1970*, volume 2. Springer-Verlag, 1983.

[Vic10] J. Vickers : The problem of induction. *In* Edward N. Zalta, éditeur : *The Stanford Encyclopedia of Philosophy*. Fall 2010 édition, 2010.

[Wal96] T. Walsh : A divergence critic for inductive proof. *Journal of Artificial Intelligence Research*, 4:209–235, 1996.

[Wos88] L. Wos : *Automated Reasoning : 33 Basic Research Problems*. Prentice Hall, 1988.

[WRCS67] L. Wos, G. A. Robinson, D. F. Carson et L. Shalla : The concept of demodulation in theorem proving. *J. ACM*, 14(4):698–709, 1967.

Index

Annexe A

Prévention de la divergence : fichiers de sortie

A.1 Détection de la satisfaisabilité et construction de modèles

```
Printing the simplified clauses
===============================
Found 5 different form(s)
++p(X1,X1,X2)
Current Form has 1 different
instances found
Printing Instances of the current Form
++p(X1,X1,X2)
**********************
++p([f,1](X1,[g,1](X2),X1),X3,X2) V
--p(X1,[f,1](X3,[g,1](X2),X3),X2)
Current Form has 1 different
instances found
Printing Instances of the current Form
++p([f,1](X1,[g,1](X2),X1),X3,X2) V
--p(X1,[f,1](X3,[g,1](X2),X3),X2)
**********************
--p(X1,[f,1](X1,[g,1](X2),X1),X2)
Current Form has 1 different
instances found
Printing Instances of the current Form
--p(X1,[f,1](X1,[g,1](X2),X1),X2)
**********************
++p({{[f,Plus[2,Times[2,n]]]($,[g,1](X2),$)(X1)}},X1,X2)
Current Form has 7 different instances found
Printing Instances of the
current Form
++p({{[f,2]($,[g,1](X2),$)(X1)}},X1,X2)
```

```
++p({{[f,4]($,[g,1](X2),$)(X1)}},X1,X2)
++p({{[f,6]($,[g,1](X2),$)(X1)}},X1,X2)
++p({{[f,8]($,[g,1](X2),$)(X1)}},X1,X2)
++p({{[f,10]($,[g,1](X2),$)(X1)}},X1,X2)
++p({{[f,12]($,[g,1](X2),$)(X1)}},X1,X2)
++p({{[f,14]($,[g,1](X2),$)(X1)}},X1,X2)
**********************
++p({{[f,Plus[2,n]]($,[g,1](X2),$)(X1)}},X3,X2) V
--p(X1,{{[f,Plus[2,n]]($,[g,1](X2),$)(X3)}},X2)
Current Form has 9 different instances found
Printing Instances of the current Form
++p({{[f,2]($,[g,1](X2),$)(X1)}},X3,X2) V
--p(X1,{{[f,2]($,[g,1](X2),$)(X3)}},X2)
++p({{[f,3]($,[g,1](X2),$)(X1)}},X3,X2) V
--p(X1,{{[f,3]($,[g,1](X2),$)(X3)}},X2)
++p({{[f,4]($,[g,1](X2),$)(X1)}},X3,X2) V
--p(X1,{{[f,4]($,[g,1](X2),$)(X3)}},X2)
... [skipped]
**********************
End processing
Printing the simplified clauses
================================
Found 9 different form(s)
% ... [skipped]
**********************
--q([f,Plus[1,n]](X1),X1)
Current Form has 329 different instances
found
Printing Instances of the current Form
--q([f,1](X1),X1)
--q([f,2](X1),X1)
--q([f,3](X1),X1)
--q([f,4](X1),X1)
--q([f,5](X1),X1)
--q([f,6](X1),X1)
% ... [skipped]
**********************
--p([f,Plus[1,n]](X1),X1)
Current Form has 329 different instances
found
Printing Instances of the current Form
--p([f,1](X1),X1)
--p([f,2](X1),X1)
--p([f,3](X1),X1)
--p([f,4](X1),X1)
--p([f,5](X1),X1)
--p([f,6](X1),X1)
% ... [skipped]
**********************
```

```
End processing
```

A.2 Preuve par consistance et découverte de lemmes

```
Printing the simplified clauses
===============================
Found 20 different form(s)
++equal([l0,1](0),nil)
Current Form has 1 different instances found
Printing Instances of the current Form
++equal([l0,1](0),nil)
% ... [skipped]
**********************
++equal([count_aux,1]([l0,1](X1),[s,Plus[1,n]](0)),[s,Plus[1,n]](X1))
Current Form has 202 different instances found Printing Instances of
the current Form
++equal([count_aux,1]([l0,1](X1),[s,1](0)),[s,1](X1))
++equal([count_aux,1]([l0,1](X1),[s,2](0)),[s,2](X1))
++equal([count_aux,1]([l0,1](X1),[s,3](0)),[s,3](X1))
++equal([count_aux,1]([l0,1](X1),[s,4](0)),[s,4](X1))
++equal([count_aux,1]([l0,1](X1),[s,5](0)),[s,5](X1))
++equal([count_aux,1]([l0,1](X1),[s,6](0)),[s,6](X1))
++equal([count_aux,1]([l0,1](X1),[s,7](0)),[s,7](X1))
++equal([count_aux,1]([l0,1](X1),[s,8](0)),[s,8](X1))
++equal([count_aux,1]([l0,1](X1),[s,9](0)),[s,9](X1))
++equal([count_aux,1]([l0,1](X1),[s,10](0)),[s,10](X1))
++equal([count_aux,1]([l0,1](X1),[s,11](0)),[s,11](X1))
**********************
% ... [skipped]
**********************
++equal([count_aux,1](nil,[s,Plus[1,n]](0)),[s,Plus[1,n]](0))
Current Form has 202 different instances found Printing Instances of
the current Form ++equal([count_aux,1](nil,[s,1](0)),[s,1](0))
++equal([count_aux,1](nil,[s,2](0)),[s,2](0))
++equal([count_aux,1](nil,[s,3](0)),[s,3](0))
++equal([count_aux,1](nil,[s,4](0)),[s,4](0))
++equal([count_aux,1](nil,[s,5](0)),[s,5](0))
++equal([count_aux,1](nil,[s,6](0)),[s,6](0))
++equal([count_aux,1](nil,[s,7](0)),[s,7](0))
++equal([count_aux,1](nil,[s,8](0)),[s,8](0))
++equal([count_aux,1](nil,[s,9](0)),[s,9](0))
++equal([count_aux,1](nil,[s,10](0)),[s,10](0))
++equal([count_aux,1](nil,[s,11](0)),[s,11](0))
**********************
++equal([s,Plus[2,n]](X1),[count_aux,1]([l0,1](X1),[s,Plus[2,n]](0)))
Current Form has 201 different instances found Printing Instances of
the current Form
++equal([s,2](X1),[count_aux,1]([l0,1](X1),[s,2](0)))
```

```
++equal([s,3](X1),[count_aux,1]([l0,1](X1),[s,3](0)))
++equal([s,4](X1),[count_aux,1]([l0,1](X1),[s,4](0)))
++equal([s,5](X1),[count_aux,1]([l0,1](X1),[s,5](0)))
++equal([s,6](X1),[count_aux,1]([l0,1](X1),[s,6](0)))
++equal([s,7](X1),[count_aux,1]([l0,1](X1),[s,7](0)))
++equal([s,8](X1),[count_aux,1]([l0,1](X1),[s,8](0)))
++equal([s,9](X1),[count_aux,1]([l0,1](X1),[s,9](0)))
++equal([s,10](X1),[count_aux,1]([l0,1](X1),[s,10](0)))
++equal([s,11](X1),[count_aux,1]([l0,1](X1),[s,11](0)))
++equal([s,12](X1),[count_aux,1]([l0,1](X1),[s,12](0)))
**********************
++equal([s,Plus[1,n]](0),[s,Plus[1,n]](0))
Current Form has 201 different instances found Printing Instances of
the current Form ++equal([s,1](0),[s,1](0))
++equal([s,2](0),[s,2](0)) ++equal([s,3](0),[s,3](0))
++equal([s,4](0),[s,4](0)) ++equal([s,5](0),[s,5](0))
++equal([s,6](0),[s,6](0)) ++equal([s,7](0),[s,7](0))
++equal([s,8](0),[s,8](0)) ++equal([s,9](0),[s,9](0))
++equal([s,10](0),[s,10](0)) ++equal([s,11](0),[s,11](0))
**********************
End processing
```

A.3 Terme général d'un langage défini par une grammaire

```
Printing the simplified clauses
===============================
Found 12 different form(s)
++W({{[f,Plus[2,n]](a,$)({{[f,Plus[1,n]](b,$)(b)}})}})
Current Form has 177 different instances found
Printing Instances of the current Form
++W({{[f,3](a,$)({{[f,2](b,$)(b)}})}})
++W({{[f,4](a,$)({{[f,3](b,$)(b)}})}})
++W({{[f,5](a,$)({{[f,4](b,$)(b)}})}})
**********************
% ... [skipped]
**********************

++W([f,1](a,[f,1]({{[f,Plus[2,n]](a,$)({{[f,Plus[1,n]](b,$)(b)}})}},b)))
Current Form has 176 different instances found
Printing Instances of the current Form
++W([f,1](a,[f,1]({{[f,3](a,$)({{[f,2](b,$)(b)}})}},b)))
++W([f,1](a,[f,1]({{[f,4](a,$)({{[f,3](b,$)(b)}})}},b)))
++W([f,1](a,[f,1]({{[f,5](a,$)({{[f,4](b,$)(b)}})}},b)))
**********************
% ... [skipped]
**********************
End processing
```