INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
UNIVERSITÉ JOSEPH FOURIER

SCHÉMAS DE FORMULES ET DE PREUVES : VERS PLUS DE STRUCTURE ET D'EFFICACITÉ EN DÉDUCTION AUTOMATIQUE

Mémoire de Master 2 Recherche III

Vincent ARAVANTINOS

12 juin 2007

Dirigé par :

Ricardo CAFERRA
Nicolas PELTIER

Résumé

L'OBJECTIF de notre travail était de définir un langage permettant d'exprimer et de prouver des schémas de formules, par exemple des formules logiques dépendant d'un ou plusieurs paramètres entiers.

Pour dénoter ces schémas, nous utilisons les *grammaires primales* introduites par M. Hermann. Nous montrons que ce formalisme est le seul parmi les langages de schématisations existants qui possède le pouvoir d'expression suffisant.

De par leur pouvoir d'expression, les grammaires primales sont particulièrement lourdes et difficiles à utiliser. En effet, les restrictions nécessaires à la décidabilité de l'unification sont nombreuses et peu intuitives. Dans un premier temps nous introduisons une extension des grammaires primales, avec des contraintes moins fortes sur les constructions, ce qui facilite grandement leur utilisation. Nous montrons que le pouvoir d'expression de ce nouveau formalisme est le même que celui du formalisme de départ – en particulier toute grammaire primale étendue peut-être transformée de manière automatique en une grammaire primale standard, et toutes les propriétés de décidabilité sont conservées.

En nous appuyant sur ce nouveau formalisme nous définissons ensuite un langage permettant de définir des clauses et des conjonctions itérées, proche du langage mathématique usuel, mais possédant exactement le même pouvoir d'expression que les grammaires primales.

Enfin, nous proposons un ensemble de règles d'inférence agissant sur ces schémas de clauses et de formules, et permettant d'engendrer des schémas de preuves associés.

Nous concluons en montrant quelques perspectives ouvertes par ce travail.

Remerciements

Je tiens avant tout à remercier mes tuteurs Ricardo CAFERRA et Nicolas PELTIER pour leurs encouragements, leurs remarques diverses, leur soutien, et surtout pour la stimulation et le goût de la recherche qu'ils ont su me transmettre.

Je remercie également et évidemment Anne-Lucile pour son soutien régulier, pour sa présence permanente, mes parents pour m'avoir encouragé et écouté, mes frères et mes amis pour m'avoir enduré et supporté malgré tout.

à mon arrière grand-mère, « Mémère » .

Table des matières

1 Introduction **1**
1.1 Contexte du projet . . . 1
1.2 Déduction automatique . . . 1
1.2.1 Preuves . . . 2
1.2.2 Preuves formelles . . . 2
1.2.3 Preuves automatisées . . . 2
1.3 Schémas . . . 3
1.3.1 Les schémas en tant que langage . . . 3
1.3.2 Les schémas dans la description du raisonnement . . . 4
1.3.3 Utilisation dans notre contexte . . . 4

2 Notions et définitions de base **5**
2.1 Termes . . . 5
2.2 Formules du premier ordre . . . 7
2.3 Clauses . . . 8

3 Schématisations : les grammaires primales **9**
3.1 Schématisations de termes . . . 9
3.2 Spécificité des grammaires primales . . . 10
3.3 Syntaxe . . . 10
3.4 Sémantique . . . 12
3.5 Exemple important : représentation des entiers naturels . . . 17

4 Grammaires primales étendues **19**
4.1 Définition . . . 19
4.2 Équivalence . . . 21

5 Schémas de formules **25**
5.1 Préambule : signature clausale . . . 25
5.2 Syntaxe . . . 27
5.3 Sémantique . . . 28
5.4 Exemple : le problème des pigeonniers . . . 29

6 Schémas itérés **33**
6.1 Clauses itérées . . . 33
6.1.1 Syntaxe . . . 33
6.1.2 Sémantique . . . 35
6.2 Clauses primales et clauses itérées . . . 36
6.2.1 Sémantique de l'équivalence entre une clause primale et une clause itérée . . . 36
6.2.2 Toute clause primale est équivalente à une clause itérée . . . 37
6.2.3 Tout clause itérée est équivalente à une clause primale . . . 40
6.3 Schémas itérés . . . 41
6.4 Exemple : le problème des pigeonniers . . . 41
6.5 Relaxation des contraintes : notation allégée . . . 43
6.6 Exemple : la k-coloriabilité . . . 44

7 Schémas de preuves **47**
7.1 Exemple des pigeonniers 47
7.2 Généralisation 49
7.2.1 Résolution 50
7.2.2 Factorisation 51
7.2.3 Récurrence 52

8 Considérations diverses, perspectives **55**

Plan de ce document

Le chapitre 1 page 1 présente *le contexte, les motivations et les concepts* sous-jacents à cette étude.

Le chapitre 2 page 5 rappelle les notions de base en logique du première ordre : *termes, formules, clauses*. Le lecteur averti pourra aisément se passer de cette lecture.

Après une brève introduction sur les schémas de termes, le chapitre 3 page 9 rappelle la définition des *grammaires primales*, la schématisation utilisée dans ce document.

Les grammaires primales étant un objet très formel et très lourd, le chapitre 4 page 19 présente le premier résultat original de ce mémoire : les *grammaires primales étendues*. Ces objets sont équivalents aux grammaires primales en termes de pouvoir d'expression mais présentent l'avantage d'être moins contraignants à utiliser.

Le chapitre 5 page 25 entre enfin dans le vif du sujet : on définit dans ce chapitre les *schémas de formules* en se servant des grammaires primales. Les schémas de formules ainsi obtenus héritent par conséquent de la lourdeur de ces dernières.

On introduit ensuite les *schémas itérés* au chapitre 6 page 33, un formalisme bien plus proche de l'intuition que ne le sont les schémas de formules à base de grammaires primales. Nous démontrons l'équivalence entre les deux formalismes, ce qui permet aux schémas itérés de bénéficier des avantages des grammaires primales tout en étant bien plus pratiques d'utilisation.

Le chapitre 7 page 47 introduit un système déductif sur les schémas itérés. L'exemple récurrent des pigeonniers est utilisé pour donner l'intuition des *schémas de preuves* que l'on envisage de construire. À l'heure actuelle ce système n'est qu'une ébauche — en particulier, la correction même du système est à prouver —, il permet cependant d'envisager de nombreuses pistes à explorer.

En guise de conclusion le chapitre 8 page 55 présente les *perspectives* de ce travail, et les pistes pour des travaux futurs.

Remarque : Les objets traités dans ce mémoire — en particulier les grammaires primales — sont *très formels* et très lourds à la lecture comme à l'emploi. Par conséquent une attention toute particulière a été apporté à la lisibilité et à la transmission des *idées sous-jacentes*. En particulier :

- les exemples sont souvent parsemés d'indications visuelles ;
- les points les plus formels sont suivis d'un paragraphe expliquant en français — quite à approximer la réalité — le point en question.

Ces paragraphes sont présentés de la façon suivante :

. . . . . .

Idée sous-jacente.

Ici, l'explication en français…

. . . . . .

Ainsi, nous conseillons en première lecture de ne pas se se soucier des détails formels, en suivant le « fil rouge » donné en particulier par ces paragraphes — qui peuvent être vus comme un « abstract réparti ».

Les chapitres 1 et 7 échappent à cette règle. En effet les notions introduites dans le premier sont classiques. D'autre part le chapitre 7 est plus facile à lire car il n'y a pas de définitions « très » formelles.

Chapitre 1

Introduction

1.1 Contexte du projet

NOTRE OBJECTIF est de définir une logique (i.e. un langage formel et un ensemble de règles d'inférence) permettant de raisonner sur des schémas de formules.

Un exemple typique de ce que l'on entend par schéma de formules est le principe dit du « pigeonnier » (pigeonhole) qui énonce qu'il n'existe pas de fonction injective de $[1..n]$ dans $[1..n-1]$. Ce principe peut être exprimé en logique propositionnelle (où il est souvent utilisé comme « benchmark ») pour un certain n fixé, ici l'objectif est de définir un langage permettant de raisonner directement sur l'énoncé paramétré par une variable entière n.

Les motivations de ce travail sont multiples. Nous en citons une directement liée à la théorie de la preuve, et qui voit son origine dans le cadre d'une coopération internationale du groupe qui m'accueille. Puis nous citons une application naturelle et capitale, d'un point de vue pratique, en informatique.

ORIGINELLEMENT, l'objectif était de permettre la formalisation de preuves mathématiques non exprimables en logique classique du premier ordre. Le système CERES (`http://www.logic.at/ceres/intro.html`) développé par l'Université Technique de Vienne (Autriche) implémente un algorithme d'élimination des coupures fondées sur la méthode de résolution [BL06]. L'élimination des coupures est un problème fondamental en logique, qui correspond intuitivement à l'élimination de tous les lemmes intermédiaires apparaissant dans une preuve. Elle permet de rendre explicite certains aspects « cachés » dans la preuve, par exemple de trouver des bornes supérieures pour la valeur des variables existentielles.

CERES a été utilisé pour analyser et comparer des preuves mathématiques [BHL$^+$04]. Puisque CERES utilise le calcul par résolution pour éliminer les coupures, les preuves doivent être exprimées en logique du premier ordre, ce qui se révèle souvent difficile, voire impossible, à cause des limites du pouvoir d'expression de cette logique (cela en particulier pour les preuves faisant intervenir l'induction mathématique). Actuellement un éditeur de preuve intégrant l'utilisation de schémas de preuves existe, mais il n'est utilisable que pour des commodités d'écriture : le système instantie ensuite la preuve pour un n entier fixé avant de procéder à l'élimination des coupures. Utiliser un langage de schéma plus expressif, mais en même temps suffisamment proche de la logique d'ordre 1 pour garder certaines de ses propriétés, permettrait de contourner ce problème. C'est l'objectif originel de ce travail, qui constitue un premier pas vers l'extension de la méthode d'élimination des coupures de CERES à une logique plus expressive que la logique d'ordre 1.

UNE APPLICATION naturelle de ces travaux concerne la preuve de programmes ou de circuits, dans laquelle on a souvent à considérer des formules paramétrées par un ou plusieurs entiers n (taille du circuit, des données, longueur de la liste, etc...). Les preuves sont réalisées par induction sur n, que ce soit de manière explicite (à l'aide de démonstrateurs par induction tels que SPIKE [BKR92]) ou implicite [Com01].

1.2 Déduction automatique

NOUS COMMENÇONS par une présentation générale du domaine de recherche dans lequel s'inscrit ce projet : la déduction automatique, dont le sujet d'études est la mécanisation de preuves (réfutations) de conjectures mathématiques. Ce détour préliminaire a pour objet de mieux situer notre travail.

1.2.1 Preuves

LA NOTION de *preuve* est présente dans de nombreux et divers domaines. Cela se justifie par le fait qu'une grande partie de la communication entre êtres humains a pour but de transmettre une idée. La *preuve* est là pour appuyer le discours et permettre cette transmission.

Évidemment le discours scientifique en est encore plus imprégné que le sont les dialogues de la vie courante. D'où l'importance particulière des preuves en mathématiques. Son association à l'expérimentation et à l'observation est en partie à l'origine de ce que l'on appelle la science moderne.

Il n'est donc pas surprenant que la preuve soit un sujet d'études à part entière, et ce depuis, au moins, l'Antiquité grecque. Cette étude couvre des champs divers et passionnants, tous fondamentalement liés à l'étude du raisonnement humain : quelle est la forme générale d'une preuve ? comment obtient-on une preuve ? peut-on généraliser une preuve ? comment présenter une preuve ?

1.2.2 Preuves formelles

L'ÉTUDE formelle de la preuve constitue un des domaines les plus importants de la logique, depuis les années 1930 (en particulier avec Gentzen). Avec Hilbert, les preuves deviennent même des objets mathématiques à étudier en tant que tels (*théorie de la preuve*).

La volonté de formalisation mène naturellement à la question « *Qu'est-ce qu'une preuve ?* » Et comme souvent lorsque l'on tente de formaliser une notion du sens commun, la question est simple mais les réponses sont bien plus profondes et subtiles que la question ne le suggère.

IL EST IMPORTANT de distinguer une *preuve* et la *représentation* de cette preuve.

- une preuve a comme l'un de ses buts principaux de *convaincre* (ou, tout au moins, de ne pas être rejetée) ; cette notion de conviction n'est mesurable que par l'humain à qui l'on montre la preuve.
- la théorie de la preuve, en tant que théorie mathématique, nécessite quant à elle une représentation systématique des preuves afin de pouvoir les *manipuler* en toute généralité.

Le but recherché n'est pas le même. La volonté de manipuler, transformer, comparer des preuves ne va pas forcément de pair avec l'idée de convaincre. La notion de conviction est intrinséquement subjective ce qui rend difficile la généralisation nécessaire à une formalisation.

À titre d'exemple, nous constatons que les preuves produites par les mathématiciens mettent en jeu des caractéristiques cruciales pour la notion de conviction : aspects visuels, analogies, prise en compte de l'expérience du lecteur, ... Ces preuves sont informelles.

En théorie de la preuve, cette différence est capturée par la distinction entre *preuve* et *dérivation*. Une *preuve* contient l'information computationnelle dont un ordinateur a besoin pour vérifier une proposition, c'est une preuve formelle. Une *dérivation* est ce qui *convainc* un humain de la vérité d'une proposition, c'est une preuve informelle.

LA POSSIBILITÉ d'obtenir des preuves en forme normale, l'étude de la complexité des preuves, de différents types de preuve (constructives ou classiques), de la généralisation de preuves, sont les problèmes qui intéressent les logiciens. Ces problèmes sont clairement distincts de ceux du mathématicien qui souhaite se convaincre et convaincre ses pairs. D'ailleurs les études fondamentales de la théorie de la preuve n'aident pas à démontrer des théorèmes, à *réfuter* des non-théorèmes, à *apprendre* ces pratiques, ou à *conjecturer* des propriétés générales à partir de cas particuliers. Autrement dit, elles ne servent pas au mathématicien.

Il est cependant intéressant de noter que Gentzen conçut son système formel en analysant la *pratique* des mathématiciens. Mais les preuves ainsi obtenues n'en restent pas moins très différentes des preuves produites par un humain.

1.2.3 Preuves automatisées

L'IDÉE D'AUTOMATISER les preuves émergea naturellement à la fin des années 1950 avec l'arrivée de l'ordinateur. Ce champ d'étude, plus modeste, montre cependant une importance croissante dûe à l'intégration sans cesse grandissante de l'informatique dans la vie courante. Ceci implique la nécessité de *prouver* qu'un logiciel fait bien ce que l'on attend de lui. D'autre part ce peut être un outil d'expérimentation très instructif pour le théoricien.

Nous regroupons dans ce domaine les *démonstrateurs automatiques* (Otter, Vampire, etc...) et les assistants de preuve (dans la lignée du projet AUTOMATH, tels que Coq, etc...), aussi appelés *vérificateurs formels*.

Ce champ d'étude a aussi gagné en popularité grâce à quelques succès mieux connus du « grand » public : preuve du théorème des 4 couleurs, ou plus récemment, aide à la preuve formelle de la conjecture de Kepler.

Ce type de résultats a inspiré des réflexions très profondes aux philsophes des mathématiques et à certains mathématiciens.

En particulier, ils mettent encore plus en exergue les différences entre preuve formelle et informelle. En effet ces résultats sont remarquables car ils répondent à des questions auxquels l'humain ne savait pas répondre. Par conséquent ils suscitent l'intérêt d'un public plus large que celui des logiciens. Dès lors le fait que les preuves obtenues par ce biais ne « convainquent » pas saute encore plus aux yeux.

LES SYSTÈMES de démonstration automatique ou d'aide à la preuve, même très généraux montrent des possibilités encore limitées. Nous donnons en langue naturelle un aperçu de ces limites :

1. **Manque de recul**
   - Le système est incapable d'analyser une preuve obtenue ;
   - Il est d'incapable de dégager les *pas importants*. Il est incapable de donner une *structure* d'une preuve ;
   - Pour lui, une preuve est une preuve. L'idée qu'une preuve peut être plus facile à comprendre qu'une autre lui est inaccessible ;
2. **Approche uniforme**
   Chaque problème est « attaqué » de la même manière. Évidemment : le système formel est inhérent à la définition de preuve formelle.
3. **Ils exigent trop de l'utilisateur**
   Les assistants de preuve payent leur puissance et leur généralité par leur absence « d'autonomie ». Ils ne sont que des vérificateurs (des « formalisateurs »).
4. **Absence d'apprentissage**
   Ces systèmes n'ont pas de mémoire, les échecs ou les succès rencontrés sur des problèmes déjà traités ne modifient en rien son comportement futur.
5. **Manque « d'initiative »**
   Ils sont incapables, lors d'une démonstration, de détecter un schéma possiblement généralisable. En particulier, ils ne sont d'aucune aide dans une activité *très importante* des mathématiciens : celle de « jouer » avec de petits exemples en essayant de *découvrir* des lois générales dont ces exemples seraient des cas particuliers.

S'attaquer à ces limitations est un enjeu particulièrement important si l'on souhaite que les avancées en théorie de la preuve ne soient pas confinées au petit monde des logiciens théoriques.

L'apport de notre travail concerne essentiellement le point 1.

1.3 Schémas

LA NOTION de schéma est fondamentale, et comme toute notion fondamentale, elle ne peut être approchée qu'à partir d'une caractérisation informelle.

> **SCHÉMA**
> **1.** Représentation graphique réduite à l'essentiel, et souvent symbolique, mais où toutes les informations se trouvent données de façon précise.
> **2.** Représentation mentale d'un principe de déroulement, de la structure générale d'une entité plus ou moins abstraite ou conçue abstraitement ; type structurel général dont relève quelque chose.
>
> *Le Trésor de la Langue Française*

Cette définition met en lumière deux acceptions du mot schéma :
- les schémas en tant que langage,
- l'utilisation des schémas dans la description du raisonnement.

1.3.1 Les schémas en tant que langage

DANS UN CONTEXTE FORMEL, la connotation graphique de la première acception se traduit en une connotation syntaxique. Ainsi, en première approximation un schéma est une expression avec une partie figée et une partie variable, permettant de dénoter une pluralité d'objets suivant un *modèle* (« template »). Le rôle des schémas est essentiellement d'exprimer de manière finie un ensemble infini.

Plus précisément un schéma est un triplet $(descripteur, langage, contrainte)$ où le descripteur est une expression avec des « trous » (des « blancs »), qui doivent être complétés avec des mots du langage en accord avec la contrainte, donnant l'ensemble des *instances*.

D'une manière plus générale, un langage de schémas (ou « schématisation ») peut se définir comme un triplet :
- langage de descripteurs,
- langage d'instances,
- mécanisme de passage d'un descripteur à une instance particulière (mécanisme d'*instanciation*).

1.3.2 Les schémas dans la description du raisonnement

SI L'ON SE FOCALISE sur la logique, qui est une des possibilités de formalisation du raisonnement, la notion de schéma y est très importante : schémas d'axiome, schémas de règles d'inférence (e.g. modus ponens), principe du tiers exclu, schémas de termes, etc... On se reportera à [Cor06] pour plus de détails sur l'utilisation, des schémas, d'une manière générale en logique.

Il est intéressant de noter que ces *schémas de raisonnement* sont exprimés par des *schémas de langage*.

1.3.3 Utilisation dans notre contexte

DANS LE CADRE de cette étude, on utilise les schémas pour représenter un ensemble de formules, par exemple :

$$\bigvee_{i=1}^{n} P(s^i(x)) \text{ où } n \text{ est une variable entière}$$

représente l'ensemble de formules suivant :

$$\{P(x),\ P(x) \vee P(s(x)),\ P(x) \vee P(s(x)) \vee P(s(s(x))),\ \dots\}$$

Ici, n joue le rôle de « trou » et la contrainte est que ce trou soit instancié (i.e. remplacé) par un entier.

La schématisation peut être informellement décrite de la manière suivante :
- le langage de descripteurs est celui des disjonctions « itérées » ;
- le langage des instances est celui des formules usuelles du premier ordre ;
- le mécanisme d'instanciation consiste en la substitution des trous par des entiers et en la règle de réécriture :

$$\bigvee_{i=1}^{n} C \to \underbrace{C\{_{i\leftarrow 1}\} \vee \cdots \vee C\{_{i\leftarrow n}\}}_{n \text{ fois}} \qquad \text{où } n \text{ est un entier}$$

LA MAJEURE partie de ce mémoire consiste en une description formelle de ces schémas de formules. Nous utiliserons pour ce faire un langage de schématisation de termes — les formules sont vues comme des termes. Nous introduirons plus en détail ces schématisations au chapitre 3 page 9.

Précisons simplement ici que le formalisme utilisé sera celui des grammaires primales introduites par Miki HERMANN [Her94]. La raison pour laquelle cette schématisation a été choisie est simplement qu'il s'agit de la plus puissante en termes de pouvoir d'expression. Elle a en outre l'avantage d'admettre une unification décidable (voir chapitre 3 page 9).

La partie la plus importante de notre travail a porté sur le formalisme lui-même. Nous montrons comment le rendre plus accessible et plus facile à utiliser, tout en conservant ses propriétés. Ces résultats ont naturellement une application en schématisation de termes, indépendamment de l'utilisation que nous en faisons dans ce projet.

LE CHAPITRE 7 PAGE 47 est consacré à l'ébauche d'un système déductif sur les schémas de formules introduits. Ce sont les preuves issues de ce système que nous appellerons « schémas de preuves ». Suivant la définition informelle des schémas présentée plus haut, ces schémas constituent les « schémas de raisonnement » associés aux schémas de langage que sont les schémas de formules.

À l'heure actuelle, le calcul que nous avons choisi de schématiser est la résolution pour les raisons suivantes :
- les démonstrateurs les plus efficaces et les plus connus (Prover9, Vampire, Spass, etc...) l'utilisent
- il s'agit d'un calcul *complet* (pour la réfutation) utilisant *une seule règle* (ou deux si on combine résolution binaire et factorisation).
- le procédure d'élimination des coupures de Baaz et Leitsch utilise la résolution (cf. §1.1 page 1).

Nos schémas de preuves sont ainsi des schémas de preuves par résolution (et donc des schémas de réfutations).

Chapitre 2

Notions et définitions de base

ON INTRODUIT ICI les définitions usuelles en logique du premier ordre :
- termes, e.g. :
$$f(s(a), x)$$
- substitutions, e.g. :
$$\left\{ \begin{array}{l} x \leftarrow s(y) \\ z \leftarrow x \end{array} \right\}$$
- formules, e.g. :
$$\forall x, \exists y : P(x) \implies Q(s(x), y)$$
- ensembles de clauses, e.g. :
$$\begin{array}{c} P(a) \\ \neg P(x) \vee R(x, f(x)) \\ \neg R(y, f(x)) \vee P(s(x)) \end{array}$$

Les notations utilisées sont classiques et le lecteur averti pourra aisément se passer de cette lecture. Précisons juste que tous les objets définis ici sont multisortés dans un souci de généralité.

Lorsque, dans la suite du document, nous parlerons de « termes usuels », « clauses usuelles » ou « ensembles usuels de clauses », nous ferons référence à ces définitions.

2.1 Termes

DÉFINITION 1 (SIGNATURE).
Soient :
- $\mathcal{S}$ *un ensemble fini de* symboles de sortes — *ou* types —,
- $\mathcal{F}$ *un ensemble au plus infini dénombrable de* symboles de fonction *tel que :*

$$\mathcal{F} = \bigcup_{\substack{u \in \mathcal{S}^* \\ \mathsf{s} \in \mathcal{S}}} \mathcal{F}_{u.\mathsf{s}}$$

Alors le couple $(\mathcal{S},\mathcal{F})$ *est une* signature (multisortée).

Remarque 1 *(Constante).*
En particulier, $\mathcal{F}_{\mathsf{s}}$ est l'ensemble des *constantes* de sorte s.

DÉFINITION 2 (ARITÉ, PROFIL).
Soit $f \in \mathcal{F}_{u.\mathsf{s}}$, *alors on note* $f : u \to \mathsf{s}$.
D'autre part,
- u *est le* domaine *de* f, *notée* $\mathcal{D}om(f)$,
- $|u|$ *est l'*arité *de* f, *notée* $\mathcal{A}r(f)$,
- s *est le* codomaine *de* f, *notée* $\mathcal{R}an(f)$ (comme $\mathcal{R}an$ge),
- $u.\mathsf{s}$ *est le* profil *de* f.

Exemple 1.
On prend :
- $\mathcal{S} = \{\mathsf{nat}, \mathsf{bool}\}$,

– $\mathcal{F} = \{0, s, true, false, eq_{\mathsf{nat}}, eq_{\mathsf{bool}}\}$ où :

$$\begin{array}{rr} 0 : & \longrightarrow \mathsf{nat} \\ s : & \mathsf{nat} \longrightarrow \mathsf{nat} \\ true : & \longrightarrow \mathsf{bool} \\ false : & \longrightarrow \mathsf{bool} \\ eq_{\mathsf{nat}} : & \mathsf{nat} \times \mathsf{nat} \longrightarrow \mathsf{bool} \\ eq_{\mathsf{bool}} : & \mathsf{bool} \times \mathsf{bool} \longrightarrow \mathsf{bool} \end{array}$$

DÉFINITION 3 (TERME).
Soient :
– $\Sigma = (\mathcal{S}, \mathcal{F})$ *une signature,*
– *pour chaque sorte* $\mathsf{s} \in \mathcal{S}$ *un ensemble infini dénombrable, noté* $\mathcal{X}_{\mathsf{s}}$*, de* variables de type s *(les* $\mathcal{X}_{\mathsf{s}}$ *sont disjoints deux à deux). On note* $\mathcal{X} = \bigcup_{\mathsf{s}\in\mathcal{S}} \mathcal{X}_{\mathsf{s}}$ *(*$\mathcal{X} \cap \Sigma = \emptyset$*).*

L'ensemble des termes de type s*, noté* $\mathcal{T}_{\mathsf{s}}(\Sigma, \mathcal{X})$ *est défini inductivement par :*
– $\mathcal{X}_{\mathsf{s}} \subseteq \mathcal{T}_{\mathsf{s}}(\Sigma, \mathcal{X})$ *;*
– *si* $\forall i \in 1..n : t_i \in \mathcal{T}_{\mathsf{s_i}}(\Sigma, \mathcal{X})$*, et* $f : \mathsf{s}_1, \ldots, \mathsf{s}_n \to \mathsf{s}$*, alors* $f(t_1, \ldots, t_n) \in \mathcal{T}_{\mathsf{s}}(\Sigma, \mathcal{X})$.

L'ensemble des termes *sur* Σ *et* $\mathcal{X}$*, noté* $\mathcal{T}(\Sigma, \mathcal{X})$ *est l'ensemble* $\bigcup_{\mathsf{s}\in\mathcal{S}} \mathcal{T}_{\mathsf{s}}(\Sigma, \mathcal{X})$.

On note $\mathcal{V}ar(t)$ *l'ensemble des variables apparaissant dans* t*. Les variables seront généralement notées* x, y, z *ou* $x_1, x_2, etc \ldots$.

Exemple 2.
En reprenant la signature de l'exemple 1 page précédente, alors :
– 0, $true$, $false$,
– $s(0)$, $s(s(0))$, $s(s(s(s(0))))$,
– $eq_{\mathsf{nat}}(0, 0)$, $eq_{\mathsf{nat}}(s(0), s(s(s(s(0)))))$
– $eq_{\mathsf{bool}}(true, false)$, $eq_{\mathsf{bool}}(eq_{\mathsf{nat}}(s(0), s(s(s(s(0))))), true)$.

appartiennent à $\mathcal{T}(\Sigma, \mathcal{X})$.
En revanche :
– $s(true)$, $s(s(s(s(false))))$,
– $eq_{\mathsf{nat}}(true, 0)$, $eq_{\mathsf{nat}}(true, false)$, $eq_{\mathsf{bool}}(0, true)$.

ne sont pas des termes.

DÉFINITION 4 (POSITION).
On définit l'ensemble des positions d'un terme t *de la façon suivante :*

1. *si* t *est une constante ou une variable alors* $\mathcal{P}os(t) = \{\varepsilon\}$
2. *si* $t = f(t_1, \ldots, t_n)$ *alors* $\mathcal{P}os(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{i.p \mid p \in \mathcal{P}os(t_i)\}$

De plus :
– ε *est la position* racine.
– *Le symbole fonctionnel à la racine de* t *est appelé la* tête *de* t.
– *Le* sous-terme *de* $t = f(t_1, \ldots, t_n)$ *à la position* p*, noté* $t|_p$ *est défini par :*
  1. $t|_\varepsilon = t$
  2. $t|_{i.p} = t_i|_p$
– *On note* $s[t]_p$ *le terme obtenu à partir de* s *en remplacant son sous-terme* $s|_p$ *par* t*. On dit que* $s[t]_p$ *est obtenu par* greffe *de* t *sur* s *à la position* p.
– *Quand* A *est un ensemble de positions parallèles* $\{p_1, p_2, \ldots, p_n\}$ *:*
  $s[t]_A$ *est défini comme le terme* $(((s[t]_{p_1})[t]_{p_2}) \ldots)[t]_{p_n}$ (bien défini et unique car les positions sont parallèles)
– *Soient* p *et* q *deux positions d'un terme* t.
  Alors :
  1. $p \leq q$ *si* p *est un préfixe de* q *;*
  2. $q \parallel p$ *si* $p \not< q$ *et* $q \not< p$.
     On dit alors que p *et* q *sont deux positions* parallèles *de* t.

DÉFINITION 5 (SUBSTITUTION).
Une substitution *est une fonction* $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F}, \mathcal{X})$ *telle que* $x\sigma \neq x$ *seulement pour un ensemble fini de variables.*
La substitution σ *notée* $\left\{\begin{smallmatrix} x_1 \leftarrow t_1 \\ \ldots \\ x_n \leftarrow t_n \end{smallmatrix}\right\}$ *est la substitution telle que* $x_i\sigma = t_i$.

Les substitutions s'étendent naturellement sur les termes de la façon suivante :
- $f\sigma = f$ si $f \in \mathcal{F}$, d'arité nulle ;
- $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$.

DÉFINITION 6 (F-ALGÈBRE).
Soit $(\mathcal{S}, \mathcal{F})$ une signature.
On appelle $\mathcal{F}$-algèbre sur la signature $(\mathcal{S}, \mathcal{F})$ le couple $(\mathcal{I}_\mathcal{S}, \mathcal{I}_\mathcal{F})$ tel que :
- *$\mathcal{I}_\mathcal{S}$ associe à toute sorte un ensemble non vide ;*
- *$\mathcal{I}_\mathcal{F}$ associe à tout symbole de fonction de $\mathsf{s_1} \times \cdots \times \mathsf{s_n} \to \mathsf{s}$ une fonction de $\mathcal{I}_\mathcal{S}(\mathsf{s_1}) \times \cdots \times \mathcal{I}_\mathcal{S}(\mathsf{s_n}) \to \mathcal{I}_\mathcal{S}(\mathsf{s})$.*

2.2 Formules du premier ordre

DÉFINITION 7 (LANGAGE DU PREMIER ORDRE).
On appelle langage du premier ordre *un triplet $(\mathcal{P}, \Sigma, \mathcal{X})$ où :*
- *$\Sigma = (\mathcal{S}, \mathcal{F})$ est une signature ;*
- *$\mathcal{X}$ est un ensemble au plus infini dénombrable de variables ;*
- *$\mathcal{P}$ est un ensemble au plus infini dénombrable de* symboles de prédicats *tel que :*

$$\mathcal{P} = \bigcup_{u \in \mathcal{S}^\star} \mathcal{P}_u$$

(les symboles de prédicats de profil vide sont interprétés comme des symboles propositionnels) *et :*

$$\mathcal{P} \cap \mathcal{F} = \emptyset,\ \mathcal{P} \cap \mathcal{X} = \emptyset,\ \textit{et}\ \mathcal{P} \cap \{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\} = \emptyset$$

Pour tout $P \in \mathcal{P}_{\mathsf{s_1} \ldots \mathsf{s_n}}$, on note $P : \mathsf{s_1} \times \cdots \times \mathsf{s_n}$.

DÉFINITION 8 (ATOME).
Soit un langage du premier ordre $\mathcal{L}$.
L'ensemble des atomes sur $\mathcal{L}$ est défini comme :

$$\{P(t_1, \ldots, t_n) \mid P \in \mathcal{P}_{\mathsf{s_1} \ldots \mathsf{s_n}}, t_1 \in \mathcal{T}_{\mathsf{s_1}}(\Sigma, \mathcal{X}), \ldots, t_n \in \mathcal{T}_{\mathsf{s_n}}(\Sigma, \mathcal{X})\}$$

DÉFINITION 9 (FORMULE DU PREMIER ORDRE).
L'ensemble des formules du premier ordre *est défini inductivement comme :*
- *tout atome est une formule du premier ordre ;*
- *si F est une formule du premier ordre, alors $\neg F$ en est aussi une ;*
- *si F est une formule du premier ordre et x est une variable, alors :*
  - $\forall x F$;
  - $\exists x F$;

  sont aussi des formules du premier ordre.
- *si F_1 et F_2 sont des formules du premier ordre, alors :*
  - $F_1 \vee F_2$,
  - $F_1 \wedge F_2$,
  - $F_1 \Rightarrow F_2$,
  - $F_1 \Leftrightarrow F_2$,

  sont aussi des formules du premier ordre.

DÉFINITION 10.
On définit l'ensemble des variables libres d'une formule F, *noté $\mathcal{FV}(F)$, de la manière suivante :*

$$\begin{aligned}
\mathcal{FV}(P(t_1, \ldots, t_k)) &= \mathcal{V}ar(t_1) \cup \cdots \cup \mathcal{V}ar(t_k) \\
\mathcal{FV}(\neg F) &= \mathcal{FV}(F) \\
\mathcal{FV}(\forall x F) &= \mathcal{FV}(F) \setminus \{x\} \\
\mathcal{FV}(\exists x F) &= \mathcal{FV}(F) \setminus \{x\} \\
\mathcal{FV}(F_1 \vee F_2) &= \mathcal{FV}(F_1 \wedge F_2) \\
&= \mathcal{FV}(F_1 \Rightarrow F_2) \\
&= \mathcal{FV}(F_1 \Leftrightarrow F_2) \\
&= \mathcal{FV}(F_1) \cup \mathcal{FV}(F_2)
\end{aligned}$$

Exemple 3.

En reprenant pour les termes la signature de l'exemple 1 page 5 et en ajoutant les symboles de prédicat suivants :

$$P : \mathsf{nat}$$
$$Q : \mathsf{bool}$$
$$R : \mathsf{bool} \times \mathsf{nat}$$

Alors :

$$\forall x \exists y \, (P(x) \Rightarrow ((\neg Q(true) \wedge P(s(y))) \vee R(z, s(0))))$$

est une formule, dont z est la seule variable libre.

2.3 Clauses

DÉFINITION 11 (CLAUSE).

On appelle clause *toute formule de la forme :*

$$L_1 \vee L_2 \vee \cdots \vee L_n$$

où L_i est un littéral *i.e. un atome ou la négation d'un atome, et où les variables libres sont implicitement quantifiées* universellement.

Exemple 4.

En reprenant le langage de l'exemple 3 :

$$P(a) \vee \neg Q(s(b)) \vee P(f(x_1, x_2)) \vee \neg R(a, x_2)$$

est une clause.

Chapitre 3

Schématisations : les grammaires primales

NOUS PRÉSENTONS maintenant le langage de schématisation utilisé par la suite : les *grammaires primales* introduites par Miki HERMANN (voir [Her94]).

3.1 Schématisations de termes

LES SCHÉMATISATIONS de termes (également appelées « schémas de termes », ou « méta-termes ») sont des langages permettant d'exprimer de manière finie des ensembles infinis de termes structurellement similaires.

Ces formalismes ont été proposés à partir des années 90 essentiellement pour remédier aux problèmes de divergence des procédures de calculs symboliques telles que par exemple la méthode de Knuth-Bendix [KB70]. Différents langages de schématisation ont été définis avec des pouvoirs d'expression très divers : les *termes récurrents* (ou hyper-termes) de Chen, Hsiang et Kong [CH91], les *termes avec exposants entiers* de Comon [Com95], les R*-termes* et les R*-formes* de Salzer [Sal92, Sal91a], les *grammaires primales* de Hermann [HG97], etc... On consultera [Her94] pour un état de l'art des schématisations de termes.

Tous ces formalismes permettent de construire des schémas de termes en appliquant itérativement un certain *contexte inductif*, sur un *terme de base*. Par exemple, la séquence de termes $a, f(a), f(f(a)), \ldots, f^n(a)$ est obtenue en itérant n fois le contexte $f(\ldots)$ sur le terme de base a, ce qui est usuellement noté $f(\diamond)^n.a$ (le symbole $\diamond$ dénotant un « trou »). A noter que n dénote ici une *variable entière*, dont la valeur est inconnue, et non un entier fixé. $f(\diamond)^n.a$ n'est donc pas un terme au sens classique mais un **schéma de termes** (suivant la façon dont n est quantifiée, le terme peut dénoter la séquence entière ou encore un élément arbitraire de cette séquence).

Ces langages présentent des similitudes avec les automates d'arbres [CDG+05], mais les langages qu'ils dénotent sont très différents, en particulier grâce à la présence de variables « compteurs » et à l'absence de variables « anonymes » (i.e. de variables pouvant être instanciées de manière différente à chaque itération, comme par exemple dans le terme $f(x_1, f(x_2, f(x_3, f(\ldots, f(x_n, a)))))$). Dans le cadre des automates d'arbres, le seul moyen de décrire une instance d'un automate est de donner la dérivation suivie par l'automate pour obtenir le terme. L'intérêt des schématisations est de disposer de variables compteurs, dont la simple instanciation permet d'obtenir un des termes décrits par le méta-terme. Dans notre contexte, cela a aussi l'avantage d'exhiber explicitement les paramètres entiers d'un schéma. On consultera par exemple [AHL93] pour une étude du pouvoir d'expression relatif de ces formalismes.

L'INTÉRÊT principal des schématisations est que l'unification reste décidable, ce qui permet de les intégrer à la plupart des procédures de calcul symboliques, tel que la méthode de résolution ou la procédure de Knuth-Bendix. De plus, la théorie du premier ordre est également décidable pour les grammaires primales [Pel04], ce qui signifie qu'il existe un algorithme permettant de tester la validité de toute formule purement équationnelles construite sur l'ensemble de connectifs usuels $\vee, \wedge, \neg, \forall, \exists$, où le prédicat « $=$ » est interprété comme l'égalité syntaxique entre termes.

3.2 Spécificité des grammaires primales

L'ORIGINALITÉ principale des grammaires primales par rapport aux formalismes plus simples qui ont été introduits auparavant [CH91, Com95, Sal92] est de permettre de définir des « diagonalisations », i.e. des itérations où le « contexte itéré » dépend lui même de son « rang » dans le processus d'itération. Par exemple le terme :

$$f(g^n(a), f(g^{n-1}(a), f(g^{n-2}, f(\dots, f(g(g(a)), f(g(a), f(a,a)))))))$$

ou bien encore

$$f(a, f(g(a), f(g(g(a)), f(\dots, f(g^{n-2}(a), f(g^{n-1}(a), f(g^n(a), a)))))))$$

L'avantage est que le pouvoir d'expression est augmenté de manière significative. En contrepartie, la décidabilité de l'unification est plus difficile à établir. En particulier, les conditions garantissant que l'unification reste décidable sont assez complexes *(cf. définition formelle des grammaires primales, §3.3)*.

Les grammaires primales sont le seul formalisme offrant la possibilité de considérer des contextes variables en fonction de n c'est donc celui que nous utiliserons dans ce travail.

Soit par exemple, le schéma :

$$\bigvee_{i=1}^{n} \bigvee_{j=1}^{p} P(s^i(a), s^j(a))$$

Le schéma le plus extérieur a pour terme de base la clause vide et pour contexte le terme inductif $\diamond \vee \bigvee_{j=1}^{p} P(s^i(a), s^j(a))$ qui dépend de i. Ce type de formules — *très courant* — n'est exprimable qu'avec les grammaires primales.

3.3 Syntaxe

LA DÉFINITION que nous donnons ici est la même que celle donnée dans [Her94] à ceci près que nous l'étendons à des signatures multisortées ; afin de faciliter cette généralisation, la présentation a elle aussi été modifiée.

DÉFINITION 12 (SIGNATURE PRIMALE).
On appelle signature primale *toute signature*
– *contenant la sorte* compteur *;*
– *contenant les symboles fonctionnels suivants :*

$$\begin{aligned} \mathbf{0} : & \quad \longrightarrow \mathsf{compteur} \\ \mathbf{succ} : & \quad \mathsf{compteur} \longrightarrow \mathsf{compteur} \\ + : & \quad \mathsf{compteur} \times \mathsf{compteur} \longrightarrow \mathsf{compteur} \end{aligned}$$

– *et tel que le profil de tout autre symbole fonctionnel est de l'une des deux formes suivantes :*

$$\begin{aligned} \mathsf{s_1} \times \cdots \times \mathsf{s_k} &\longrightarrow \mathsf{s_{k+1}} \quad (1) \\ \mathsf{compteur}^n \times \mathsf{s_1} \times \cdots \times \mathsf{s_k} &\longrightarrow \mathsf{s_{k+1}} \quad (2) \end{aligned}$$

où $n \geq 0$ *et* $\forall i \in 1..k+1, s_i \neq \mathsf{compteur}$.

Les symboles dont le profil a la forme (1) sont appelés *symboles constructeurs*.

Les symboles dont le profil a la forme (2) sont appelés *symboles définis*. Ces symboles seront notés $\hat{f}, \hat{g}, \dots$ n est alors appelé l'*arité de compteurs* du symbole défini, notée $\mathcal{A}r_c(\hat{f})$.

Toute signature primale est munie d'une précédence $\prec$ sur ses symboles définis.

. . . . . .

Idée sous-jacente.

Ce qui importe ici, c'est qu'il y a deux types de symboles de fonction :
– des symboles ayant au moins un paramètre de sorte compteur ; ces symboles permettront de définir les schémas de termes (le(s) paramètre(s) de sorte compteur permet(tent) de préciser le nombre d'itérations du contexte).
– des symboles ne faisant pas intervenir la sorte compteur ; ces symboles permettront de construire les « vrais » termes, i.e. les instances d'un schéma de termes.

. . . . . .

Remarque 2.
La définition originale des gramaires primales n'inclut pas le symbole $+$ pour les termes de type compteur. Nous l'ajoutons ici pour une raison de simplicité, ce qui n'a pas de conséquence sur les propriétés des grammaires primales.

Exemple 5 *(Symboles définis et arités).*

$$\hat{f} : \overbrace{\underbrace{\mathsf{compteur} \times \mathsf{compteur} \times \mathsf{compteur}}_{\mathcal{A}r_c(\hat{f})=3} \times \mathsf{t} \times \mathsf{t}}^{\mathcal{A}r(\hat{f})=5} \longrightarrow \mathsf{t}$$

$$\hat{g} : \overbrace{\underbrace{\mathsf{compteur} \times \mathsf{compteur}}_{\mathcal{A}r_c(\hat{g})=2}}^{\mathcal{A}r(\hat{g})=2} \longrightarrow \mathsf{t}$$

DÉFINITION 13 (ALGÈBRE DES TERMES PRIMAUX).
Un terme *primal est un terme sur une signature primale, dont la sorte est différente de* compteur.

Les sous-termes de sorte compteur sont appelés *expressions compteurs.*
Les variables de sorte compteur sont appelées *variables compteurs.* On note $\mathcal{CV}ar(t)$ les variables compteurs d'un terme primal t.
Les autres variables sont appelées *variables usuelles.*
Les termes primaux construits sans symbole défini sont appelés *termes usuels.*
On appelle *termes rédex* [1] les termes dont la tête est un symbole défini. Ils seront notés :

$$\hat{f}(c_1, \ldots, c_n; t_1, \ldots, t_k)$$

où :
- $\forall i \in 1..n,\ \mathcal{S}orte(c_i) = \mathsf{compteur}$,
- $\forall i \in 1..k,\ \mathcal{S}orte(t_i) = \mathsf{s_i}$.

.....

Idée sous-jacente.
Quand un terme primal contient un sous-terme rédex, c'est qu'il contient au moins un symbole défini.
Ainsi, informellement, les termes contenant des sous-termes rédex sont des schémas de termes. Alors que les termes ne contenant pas de sous-termes rédex sont des termes usuels.

.....

Exemple 6 *(Termes primaux).*
Soient les symboles de fonction suivants :

$$\begin{array}{lr}
\hat{f} : & \mathsf{compteur} \times \mathsf{t} \longrightarrow \mathsf{t} \\
\hat{g} : & \mathsf{compteur} \times \mathsf{compteur} \times \mathsf{t} \longrightarrow \mathsf{t} \\
a : & \longrightarrow \mathsf{t} \\
s : & \mathsf{t} \longrightarrow \mathsf{t}
\end{array}$$

Alors des exemples de termes primaux sur cette signature sont :
- les termes usuels $(a, s(a), s(s(a)), x_1, s(s(s(s(x_2)))))$;
- $\hat{f}(\mathbf{succ}(\mathbf{succ}(\mathbf{0})); a)$
- $\hat{f}\big(\mathbf{0}; s(x_2)\big)$
- $\hat{g}\big(c_1, \mathbf{succ}(\mathbf{0})\ ;\ a, x_1\big)$
- $\hat{g}\big(\mathbf{succ}(c_1), c_2\ ;\ s(x_2), x_1\big)$
- $\hat{f}\big(c_1\ ;\ \hat{g}(c_1, 0; a, a)\big)$

où x_1, x_2 sont des variables usuelles et c_1, c_2 des variables compteurs.

DÉFINITION 14 (TERME RÉGULIER).
Un terme primal est dit régulier *si les positions de ses sous-termes rédex sont mutuellement parallèles.*

[1] « rédex » car nous verrons que le but de ces termes est d'être *réduit*par un système de réécriture de termes

.....

Idée sous-jacente.
Un terme régulier correspond simplement au fait que plusieurs schémas de termes sont « imbriqués ».

.....

Exemple 7.
Sont réguliers :
- $\underline{\hat{f}(c_1, c_2; x_1, a)}$
- a
- $g(\underline{\hat{f}(c_1, \mathbf{0}; x_1, a)}, \underline{\hat{f}(\mathbf{succ}(c_2), c_2; x_2, x_1)})$
- $g(\underline{\hat{f}(c_1, \mathbf{0}; x_1, a)}, \underline{\hat{h}(c_2; x_2)})$

N'en sont pas :
- $\underline{\hat{f}(c_1, c_2; x_1, \underline{\hat{h}(c_2; x_2)})}$
- $\underline{\hat{f}(c_1, c_2; \underline{\hat{f}(c_1, c_3; x_2), x_2})}$
- $g(\underline{\hat{f}(c_1, c_2; \underline{\hat{f}(c_1, c_3; x_2)}, x_2)}, a)$

3.4 Sémantique

LES TERMES PRIMAUX ont pour but d'être interprétés par des ensembles de termes usuels, mais pas n'importe quelle type d'ensembles : nous nous restreignons aux ensembles représentables par des « systèmes Presburger ». Cette restriction poursuit deux buts :
- avoir un moyen *mécanique* d'exprimer ces ensembles ;
- assurer la décidabilité de l'unification entre grammaires primales.

Systèmes de réécriture Presburger [Her94]

LES SYSTÈMES PRESBURGER ont plusieurs points communs avec des grammaires classiques. Nous retrouvons donc certaines notions des grammaires sous des dénominations différentes : En particulier, le vocabulaire *terminal* (respectivement *non terminal*) correspond ici à l'ensemble des symboles *constructeurs* (respectivement *définis*). L'*axiome* est ici un terme primal.

Nous commençons par définir la notion d'approximation, nécessaire à la définition de système Presburger.

DÉFINITION 15 (APPROXIMATION).
Soient :
- *un symbole défini* $\hat{f}$ *;*
- *des expressions compteurs* $c_1, \ldots, c_k$ *;*
- *des termes primaux* $t_1, \ldots, t_l$*.*

Alors on appelle approximation *du terme primal* $\hat{f}(\vec{c}; \vec{t})$*, l'ensemble suivant, noté* $\mathcal{A}px(\hat{f}(\vec{c}; \vec{t}))$ *:*

$$\mathcal{A}px(\hat{f}(\vec{c}; \vec{t})) = \left\{ \hat{g}(\vec{c'}; \vec{t'}) \;\middle|\; \begin{array}{l} \hat{f} \succ \hat{g} \\ \vec{c'} \text{ est une sous-suite de } \vec{c} \\ \vec{t'} \text{ est une sous-suite de } \vec{t} \end{array} \right\}$$

.....

Idée sous-jacente.
La notion d'approximation permet simplement de définir un ordre bien fondé sur les termes primaux.

.....

Exemple 8.
Soient les symboles fonctionnels suivants :

$$\begin{array}{ll} \hat{f} : & \mathsf{compteur} \times \mathsf{compteur} \times \mathsf{compteur} \times \mathsf{t} \times \mathsf{t} \longrightarrow \mathsf{t} \\ \hat{g} : & \mathsf{compteur} \times \mathsf{compteur} \times \mathsf{t} \longrightarrow \mathsf{t} \\ \hat{h} : & \mathsf{compteur} \times \mathsf{compteur} \times \mathsf{t} \longrightarrow \mathsf{t} \end{array}$$

avec la précédence $\hat{f} \succ \hat{g} \succ \hat{h}$.

Alors :

$\hat{g}(\ c_1, c_3\ ;\ x_1\ ) \in \mathcal{A}px\Big(\hat{f}(c_1, c_2, c_3; x_1, x_2)\Big)$ $\curlywedge \quad \sqcap\!\!\mid \quad \sqcap\!\!\mid$ $\hat{f}(c_1, c_2, c_3\ ; x_1, x_2)$	$\hat{h}(\ c_2, c_3\ ;\ x_2\ ) \in \mathcal{A}px\Big(\hat{f}(c_1, c_2, c_3; x_1, x_2)\Big)$ $\curlywedge \quad \sqcap\!\!\mid \quad \sqcap\!\!\mid$ $\hat{f}(c_1, c_2, c_3\ ; x_1, x_2)$
$\hat{g}(c_1, c_2\ ; x_2) \notin \mathcal{A}px\Big(\hat{h}(c_1, c_2; x_2)\Big)$ $\not\curlywedge$ $\hat{h}(c_1, c_2\ ; x_2)$	$\hat{g}(\ \mathbf{0}, c_2\ ;\ x_1\ ) \notin \mathcal{A}px\Big(\hat{f}(c_1, c_2, c_3; x_1, x_2)\Big)$ $\not\sqcap\!\!\mid$ $\hat{f}(c_1, c_2, c_3\ ; x_1, x_2)$
$\hat{h}(c_2, c_1\ ;\ x_2\ ) \notin \mathcal{A}px\Big(\hat{g}(c_1, c_2; x_1, x_2)\Big)$ $\not\sqcap\!\!\mid$ $\hat{g}(c_1, c_2\ ; x_1, x_2)$	$\hat{g}(\mathbf{succ}(c_1), c_2\ ;\ x_1\ ) \notin \mathcal{A}px\Big(\hat{f}(c_1, c_2, c_3; x_1, x_2)\Big)$ $\not\sqcap\!\!\mid$ $\hat{f}(\ c_1, c_2, c_3\ ; x_1, x_2)$

DÉFINITION 16 (SYSTÈME DE RÉÉCRITURE PRESBURGER).
Un système de réécriture $\mathcal{R}$ est dit Presburger *si pour chaque symbole défini $\hat{f}$, il contient :*
– *une* règle de base *:*

$$\hat{f}(\mathbf{0}, \vec{c}; \vec{x}) \to r_1^{\hat{f}}$$

– *une* règle inductive *de l'une des deux formes suivantes :*

$$\begin{cases} \hat{f}(\mathbf{succ}(i), \vec{c}; \vec{x}) \to r_2^{\hat{f}}[\hat{f}(i, \vec{c}; \vec{x})]_A \\ ou \\ \hat{f}(\mathbf{succ}(i), \vec{c}; \vec{x}) \to r_2^{\hat{f}}[\hat{f}(i, c_1, \ldots, c_{k-1}, \mathbf{succ}(c_k), c_{k+1}, \ldots, c_n; \vec{x})]_A \end{cases}$$

tels que :

1. *A est un sous-ensemble fini des positions parallèles autres que la racine de $r_2^{\hat{f}}$;*
2. *$r_2^{\hat{f}}[\hat{f}(i, \vec{c}; \vec{x})]_A$ et $r_2^{\hat{f}}[\hat{f}(i, c_1, \ldots, c_{k-1}, \mathbf{succ}(c_k), c_{k+1}, \ldots, c_k; \vec{x})]_A$ sont des termes réguliers ;*
3. *tous les rédex de $r_1^{\hat{f}}$ et $r_2^{\hat{f}}$ appartiennent à $\mathcal{A}px(\hat{f}(i, \vec{c}; \vec{x}))$;*
4. *le symbole racine de $r_2^{\hat{f}}$ est un symbole constructeur ;*
5. *chaque variable usuelle doit apparaître dans $r_1^{\hat{f}}$ ou $r_2^{\hat{f}}$.*

i est appelé le *compteur principal* et, dans la règle inductive, c_k le *compteur auxiliaire.*

. . . . . .

Idée sous-jacente.
Cette définition constitue le point clé des grammaires primales. Le rôle d'un système Presburger est de donner l'interprétation d'un symbole défini — et donc d'un schéma de termes.

En première approximation, il s'agit simplement, comme on l'a vu précédemment, de définir un terme de base et un contexte inductif. Le schéma défini alors peut s'écrire plus concisément : $r_2^{\hat{f}}[\diamond].r_1^{\hat{f}}$

Le gain essentiel des grammaires primales par rapport aux autres schématisations est apporté par la notion de *compteur auxiliaire* introduite lorsque l'on utilise la seconde forme de règle inductive. *C'est ce compteur auxiliaire qui permet d'avoir un contexte dépendant du rang.*

La contrainte d'appartenance à l'approximation assure la terminaison.

L'interdiction des termes non réguliers est nécessaire à la décidabilité de l'unification [2].

Les deux derniers points sont accessoires, ils facilitent les démonstrations mais ne sont pas vraiment restrictifs.

. . . . . .

Exemple 9.
En reprenant la signature de l'exemple 8 page ci-contre en y ajoutant les symboles constructeurs :

$$\begin{array}{ll} r: & \mathtt{t} \times \mathtt{t} \times \mathtt{t} \longrightarrow \mathtt{t} \\ a: & \mathtt{t} \longrightarrow \mathtt{t} \end{array}$$

On peut définir pour $\hat{f}$ le système Presburger suivant :

[2] en effet dans le cas contraire la multiplication entre compteurs serait encodable dans les grammaires primales : $f^n(\diamond)^m.a$ équivaut à $f^{n.m}(\diamond).a$. Auquel cas le problème de l'unification se réduirait au 10e problème de Hilbert, indécidable.

$$\begin{array}{ll}\hat{f}(\mathbf{0}, c_2, c_3; x_1, x_2) & \to r\big(a,\ \hat{h}(c_2, c_3; x_2),\ a\big) \\ \hat{f}(\mathbf{succ}(c_1), c_2, c_3; x_1, x_2) & \to \\ & \quad r\Big(\hat{g}(c_1, c_3; x_1),\ r\big(\hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2), a, \hat{g}(c_1, c_2; x_2)\big),\ \hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2)\Big)\end{array}$$

On vérifie que les conditions sont réunies :

1. $r\Big(\hat{g}(c_1, c_3; x_1),\ r\big(\underbrace{\hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2)}_{p_1}, a, \hat{g}(c_1, c_2; x_2)\big),\ \underbrace{\hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2)}_{p_2}\Big)$:
   - $p_1 \parallel p_2$;
   - p_1 et p_2 ne sont pas la racine.
2. $r\Big(\underline{\hat{g}(c_1, c_3; x_1)},\ r\big(\underline{\hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2)}, a, \underline{\hat{g}(c_1, c_2; x_2)}\big),\ \underline{\hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2)}\Big)$
   est un terme régulier ;
3. $r\big(a,\ \underbrace{\hat{h}(c_2, c_3; x_2)}_{\in \mathcal{A}px(\hat{f}(c_1, c_2, c_3; x_1, x_2))},\ a\big)$
   (deuxième cas de l'exemple 8 page 12)
   et :
   $r\Big(\underbrace{\hat{g}(c_1, c_3; x_1),}_{\in \mathcal{A}px(\hat{f}(c_1, c_2, c_3; x_1, x_2))}\ r\big(\hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2), a, \underbrace{\hat{g}(c_1, c_2; x_2)}_{\in \mathcal{A}px(\hat{f}(c_1, c_2, c_3; x_1, x_2))}\big), \ldots\Big)$
   (premier cas de l'exemple 8 page 12 + variante de ce cas)
4. $r\Big(\hat{g}(c_1, c_3; x_1),\ r\big(\hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2), a, \hat{g}(c_1, c_2; x_2)\big), \ldots\Big)$
   $\downarrow$
   constructeur
5. $r\Big(\hat{g}(c_1, c_3; \boxed{x_1}),\ r\big(\hat{f}(c_1, c_2, \mathbf{succ}(c_3); x_1, x_2), a, \hat{g}(c_1, c_2; \boxed{x_2})\big), \ldots\Big)$
   x_1 et x_2 apparaîssent bien tous les deux.

Exemple 10.
En revanche avec la même signature, les systèmes suivants *ne sont pas* des systèmes Presburger :

$$\begin{cases}\hat{f}(\mathbf{0}, c_2, c_3; x_1, x_2) & \to a \\ \hat{f}(\mathbf{succ}(c_1), c_2, c_3; x_1, x_2) & \to \boxed{\hat{f}}(c_1, c_2, c_3; x_1, x_2)\end{cases}$$
(car A contient la racine)

$$\begin{cases}\hat{f}(\mathbf{0}, c_2, c_3; x_1, x_2) & \to x_2 \\ \hat{f}(\mathbf{succ}(c_1), c_2, c_3; x_1, x_2) & \to \underline{\hat{g}(c_1, c_2; x_1, \underline{\hat{f}(c_1, c_2, c_3; x_1, x_2)})}\end{cases}$$
(car $r_2^{\hat{f}}[\hat{f}(c_1, c_2, c_3; x_1, x_2)]_A$ n'est pas régulier ou car la racine est un symbole défini)

$$\begin{cases}\hat{f}(\mathbf{0}, c_2, c_3; x_1, x_2) & \to \hat{h}(c_2, c_1; x_1) \quad \notin \mathcal{A}px(\hat{f}(c_1, c_2, c_3; x_1, x_2)) \\ \hat{f}(\mathbf{succ}(c_1), c_2, c_3; x_1, x_2) & \to r(a, x_2, \hat{f}(c_1, c_2, c_3; x_1, x_2)\end{cases}$$

$$\begin{cases}\hat{f}(\mathbf{0}, c_2, c_3; x_1, x_2) & \to a \\ \hat{f}(\mathbf{succ}(c_1), c_2, c_3; x_1, x_2) & \to r(\underbrace{\hat{g}(\mathbf{succ}(c_1), c_2; x_1)}_{\notin \mathcal{A}px(\hat{f}(c_1, c_2, c_3; x_1, x_2))}, a, \hat{f}(c_1, c_2, c_3; x_1, x_2))\end{cases}$$
(en effet : voir le dernier cas de l'exemple 8 page 12)

$$\begin{cases}\hat{f}(\mathbf{0}, c_2, c_3; x_1, x_2) & \to a \\ \hat{f}(\mathbf{succ}(c_1), c_2, c_3; x_1, x_2) & \to r(a, x_2, \hat{f}(c_1, c_2, c_3; x_1, x_2))\end{cases}$$
(x_1 n'est pas utilisé)

$$\begin{cases}\hat{f}(\mathbf{0}, c_2, c_3; x_1, x_2) & \to x_1 \\ \hat{f}(\mathbf{succ}(c_1), c_2, c_3; x_1, x_2) & \to r(x_2, \boxed{\hat{f}(c_1, c_2, \underline{\mathbf{succ}(c_3)}; x_1, x_2)}, \boxed{\hat{f}(c_1, c_2, \underline{c_3}; x_1, x_2))}\end{cases}$$
(la règle inductive ne peut pas à la fois itérer le compteur auxiliaire d'une part et le laisser tel quel d'autre part)

Propriété 1 (Normalisation).
Tout système Presburger est nœthérien.
i.e. tout terme t possède une *forme normale.*

Démonstration. voir [Her94]. □

Propriété 2 (Confluence).
Tout système Presburger est confluent.
i.e. toute forme normale d'un terme t est *unique.*

Démonstration. Tout système Presburger est orthogonal *(i.e. sans paire critique et linéaire à gauche).* □

Corollaire 1 (Convergence).
Tout système Presburger $\mathcal{R}$ est convergent.
i.e. tout terme t possède une *unique* forme normale notée $t{\downarrow_{\mathcal{R}}}$ (ou simplement $t{\downarrow}$ lorsque $\mathcal{R}$ est sous-entendu).

Grammaires primales

DÉFINITION 17 (GRAMMAIRE PRIMALE).
On appelle grammaire primale *un triplet* $(\Sigma, \mathcal{R}, t^\star)$ *où :*
- Σ *est une signature primale,*
- $\mathcal{R}$ *est un système de réécriture Presburger,*
- $t^\star$ *est un terme primal sur* Σ*, appelé* axiome.

Remarque 3.
Lorsque cela ne présente pas d'ambiguïté, nous considérerons implicites la signature et le système Presburger — ceux-ci doivent cependant être clairs dans le contexte. Nous identifierons ainsi un terme primal à la grammaire dont il est l'axiome.

.....

Idée sous-jacente.
On assemble les briques : une grammaire primale est un couple (terme primal, interprétation).

.....

Interprétation

INTUITIVEMENT , les termes primaux permettent de « parler de plusieurs termes en même temps ». On fait ici le lien formel avec cette intuition.

DÉFINITION 18 (ÉNUMÉRATEUR, LANGAGE ENGENDRÉ).
Un énumérateur *d'un terme primal* t *est une substitution close ayant pour domaine les variables compteurs de* t *(i.e.* $\xi : \mathcal{CV}ar(t) \to \mathbb{N}$*).*
Le langage *engendré par une grammaire primale* $G = (\Sigma, \mathcal{R}, t^\star)$*, noté* $L(G)$*, est l'ensemble :*

$$L(G) = \{t^\star\xi{\downarrow_{\mathcal{R}}} \mid \xi \text{ est un énumérateur de } t\}$$

.....

Idée sous-jacente.
Le langage engendré est l'ensemble des termes représentés par le schéma de termes. On l'obtient en appliquant le système Presburger à toutes les instanciations possibles des variables compteurs.

.....

Exemple 11.
(tiré de [Her94])
Soit la grammaire primale G,
- de signature primale :

$$\begin{array}{rr}
\hat{f} : & \mathsf{compteur} \times \mathsf{compteur} \longrightarrow \mathsf{t} \\
\hat{g} : & \mathsf{compteur} \longrightarrow \mathsf{t} \\
+ : & \mathsf{t} \times \mathsf{t} \longrightarrow \mathsf{t} \\
a : & \longrightarrow \mathsf{t} \\
b : & \mathsf{t} \longrightarrow \mathsf{t} \\
c : & \longrightarrow \mathsf{t}
\end{array}$$

– de système Presburger associé *(pour des questions de lisibilité on utilise la notation infixée pour $+$)* :

$$\hat{f}(\mathbf{0}, j) \to c \qquad \hat{g}(\mathbf{0}) \to a$$
$$\hat{f}(\mathbf{succ}(i), j) \to \hat{g}(j) + \hat{f}(i, \mathbf{succ}(j)) \qquad \hat{g}(\mathbf{succ}(j)) \to b(\hat{g}(j))$$

– et d'axiome :

$$\hat{f}(n, \mathbf{0})$$

Alors :

$$L(G) \;=\; \{a + (b(a) + (\ldots + (b^n(a) + c)\ldots)) \mid n \in \mathbb{N}\}$$

Remarque 4.
Dans la continuité de la remarque 3 page précédente, lorsqu'il n'y a pas d'ambiguïté ou qu'il n'y a pas lieu de préciser la signature et le système Presburger, nous pourrons parler du langage d'un *terme primal* pour désigner en fait celui de la grammaire primale induite.
Dans ce cas la forme normale d'un terme u sera notée simplement $u\downarrow$.

Remarque 5.
Trivialement le langage engendré ne contient pas de terme rédex.

DÉFINITION 19 (ÉQUIVALENCE).
Deux grammaires primales G_1 et G_2 sont dites équivalentes*, ce qui sera noté $G_1 \equiv G_2$, si et seulement si leurs langages engendrés sont égaux.*

NOUS EN PROFITONS pour présenter la propriété suivante qui nous dit que l'on peut systématiquement supposer que c'est la deuxième forme de règle inductive qui est utilisée. Cela permet de simplifier les démonstrations ultérieures.

Propriété 3.
Soit un symbole défini sans compteur auxiliaire.
Alors il existe un symbole défini *avec compteur auxiliaire* qui lui est équivalent.

Démonstration.
– On note $\hat{f}$ le symbole défini sans compteur auxiliaire de système Presburger :

$$\hat{f}(\mathbf{0}, \vec{c}; \vec{x}) \longrightarrow r_1^{\hat{f}}$$
$$\hat{f}(\mathbf{succ}(i), \vec{c}; \vec{x}) \longrightarrow r_2^{\hat{f}}[\hat{f}(i, \vec{c}; \vec{x})]_A$$

– On note $\hat{f}'$ le symbole défini *avec compteur auxiliaire j* de système Presburger :

$$\hat{f}'(\mathbf{0}, \vec{c}, j; \vec{x}) \longrightarrow r_1^{\hat{f}}$$
$$\hat{f}'(\mathbf{succ}(i), \vec{c}, j; \vec{x}) \longrightarrow r_2^{\hat{f}}[\hat{f}'(i, \vec{c}, \mathbf{succ}(j))]_A$$

où j est une variable n'apparaîssant pas dans $r_1^{\hat{f}}$ et $r_2^{\hat{f}}$ (sinon on renomme).
– Montrons que pour tout énumérateur des deux termes, noté ξ :

$$\hat{f}(i, c_1, \ldots, c_k; \vec{x})\xi\downarrow = \hat{f}'(i, c_1, \ldots, c_k, j; \vec{x})\xi\downarrow$$

Soit un ξ un énumérateur de $\hat{f}(i, c_1, \ldots, c_k; \vec{x})$.
Alors $\xi' = \xi \cup \{j \leftarrow p\}$ est un énumérateur de $\hat{f}'(i, c_1, \ldots, c_k, j; \vec{x})$.
On note $n = i\xi (= i\xi')$ et on procède par récurrence sur n :

1. $n = 0$

$$\begin{array}{ll} \hat{f}(i, \vec{c}; \vec{x})\xi\downarrow & \hat{f}'(i, \vec{c}, j; \vec{x})\xi'\downarrow \\ = \hat{f}(\mathbf{0}, \vec{c}\xi; \vec{x})\downarrow & = \hat{f}'(\mathbf{0}, \vec{c}\xi', j\xi'; \vec{x})\downarrow \\ = r_1^{\hat{f}}\xi & = r_1^{\hat{f}}\xi' \\ & = r_1^{\hat{f}}\xi \quad (j \text{ n'apparaît pas dans } r_1^{\hat{f}}) \end{array}$$

2. $n \Rightarrow n+1$

$$\begin{array}{ll}
\hat{f}'(\mathbf{succ}(i), \vec{c}, j; \vec{x})\xi'\downarrow & \hat{f}(\mathbf{succ}(i), \vec{c}; \vec{x})\xi\downarrow \\
= \hat{f}'(\mathbf{succ}(n), \vec{c}\xi', j\xi'; \vec{x})\downarrow & = \hat{f}(\mathbf{succ}(n), \vec{c}\xi; \vec{x})\downarrow \\
= r_2^{\hat{f}}\xi'[\hat{f}'(n, \vec{c}\xi', \mathbf{succ}(j\xi'); \vec{x})\downarrow]_A & = r_2^{\hat{f}}\xi[\hat{f}(n, \vec{c}\xi; \vec{x})\downarrow]_A \\
= r_2^{\hat{f}}\xi[\hat{f}'(n, \vec{c}\xi', \mathbf{succ}(j\xi'); \vec{x})\downarrow]_A & \textit{(j n'apparaît pas dans } r_1^{\hat{f}}\textit{)} \\
= r_2^{\hat{f}}\xi[\hat{f}(n, \vec{c}\xi; \vec{x})\downarrow]_A & \textit{(hypothèse de récurrence)}
\end{array}$$

– On obtient donc que les langages engendrés sont les mêmes, et par la même l'équivalence entre $\hat{f}$ et $\hat{f}'$.

□

3.5 Exemple important : représentation des entiers naturels

BIEN QUE LA REPRÉSENTATION des entiers naturels avec les grammaires primales ne soit pas difficile, son importance réside dans le fait que nous en ferons une utilisation récurrente dans les exemples des chapitres suivants.

On utilise la définition suivante (classique) des entiers naturels, on se donne :
– une sorte entier ;
– une constante a de sorte entier qui dénote le 0 ;
– le symbole fonctionnel s : entier $\rightarrow$ entier qui dénote la fonction sucesseur.
On pose alors le symbole défini suivant :

$$\begin{aligned}
\hat{\mathcal{N}}(\mathbf{0}) &\longrightarrow a \\
\hat{\mathcal{N}}(\mathbf{succ}(i)) &\longrightarrow s(\hat{\mathcal{N}}(i))
\end{aligned}$$

qui, en fait, ne fait que traduire les compteurs en termes usuels.
Alors le langage engendré par $\hat{\mathcal{N}}(n)$ est l'ensemble $\{a, s(a), s(s(a)), \dots\}$.

Notation 1.
Par la suite, certains exemples utiliserons à outrance des termes de cette forme, ainsi nous noterons dans ces cas :

$$\hat{i} \stackrel{\text{def}}{=} \hat{\mathcal{N}}(i)$$

Chapitre 4

Grammaires primales étendues

NOUS INTRODUISONS ici les *grammaires primales étendues* dont la présentation diffère légèrement des grammaires primales usuelles mais qui leurs sont équivalentes. L'intérêt de ce formalisme est qu'il est plus pratique à utiliser, en effet les contraintes à respecter sont moins restrictives que celles des systèmes Presburger.

Ce premier résultat est de plus une première étape essentielle vers la définition des « schémas itérés » (chapitre 6 page 33). Remarquons cependant que ce résultat est un résultat général sur les grammaires primales, On pourra donc trouver des applications en schématisation de termes indépendamment de l'utilisation que nous en faisons dans ce projet.

4.1 Définition

CE FORMALISME nécessite cependant de définir une nouvelle notion : celle de compteur potentiel. Cette notion va remplacer celle d'approximation, elle est plus complexe que cette dernière mais permet d'alléger grandement la définition de système Presburger.

DÉFINITION 20 (COMPTEUR POTENTIEL D'UN TERME).
Soit un terme primal t.

Alors l'ensemble des compteurs quelconques *de* t*, noté* $\mathcal{CQ}(t)$*, et l'ensemble des* compteurs auxiliaires potentiels *sont inductivement définis de la façon suivante :*

t	$\mathcal{CQ}(t)$	$\mathcal{CA}(t)$
$\hat{f}(c_0, c_1, \ldots, c_k; \ldots)$	$\mathcal{V}ar(c_1) \cup \cdots \cup \mathcal{V}ar(c_k)$	$\left\{ i \;\middle\vert\; \begin{array}{c} \textit{il existe un unique } j \in 0..k \textit{ tq :} \\ i \in \mathcal{V}ar(c_j) \end{array} \right\}$
$f(u_1, \ldots, u_k)$	$\mathcal{CQ}(u_1) \cup \cdots \cup \mathcal{CQ}(u_k)$	$\mathcal{CA}(u_1) \cup \cdots \cup \mathcal{CA}(u_k)$

Une variable compteur est un compteur principal potentiel *de* t *si ce n'est pas un compteur quelconque de* t.

On note $\mathcal{CP}(t)$ *l'ensemble des compteurs principaux potentiels de* t.

.....

Idée sous-jacente.

Une variable compteur est *compteur principal potentiel* si ses seules occurrences sont dans les *premiers paramètres* des sous-termes rédex.

Une variable compteur est *compteur auxiliaire potentiel* si il n'a qu'une seule occurence dans chaque sous-terme rédex.

.....

Exemple 12.

Soit la signature suivante :

$$\begin{array}{ll} \hat{f} : \mathsf{compteur} \times \mathsf{compteur} \times \mathsf{t} & \longrightarrow \mathsf{t} \\ \hat{g} : \mathsf{compteur} \times \mathsf{t} & \longrightarrow \mathsf{t} \\ h : \mathsf{t} \times \mathsf{t} & \longrightarrow \mathsf{t} \\ a : & \longrightarrow \mathsf{t} \end{array}$$

$t =$
(les compteurs quelconques sont encadrés)

$h(a, x)$	n'a pas de compteur quelconque *donc toute variable compteur est compteur principal potentiel* n'a pas de compteur auxiliaire
$\hat{f}(c_1, \boxed{c_2}; x)$	$c_1 \in \mathcal{CP}(t)$ $c_1, c_2 \in \mathcal{CA}(t)$
$\hat{f}(c_1 + c_2, \boxed{c_3} + \boxed{c_2}; x)$	$c_1 \in \mathcal{CP}(t)$ $c_3 \in \mathcal{CA}(t)$ mais $c_2 \notin \mathcal{CA}(t)$
$h(\hat{f}(c_1, \boxed{c_2}; x), a)$	$c_1 \in \mathcal{CP}(t)$ $c_2 \in \mathcal{CA}(t)$
$h(\hat{f}(c_1, \boxed{c_3}; x), \hat{g}(c_2 + c_1, \boxed{c_3} + \boxed{c_2}))$	$c_1 \in \mathcal{CP}(t)$ mais $c_2 \notin \mathcal{CP}(t)$ $c_1 \in \mathcal{CA}(t)$ $c_3 \in \mathcal{CA}(t)$ car ses deux occurrences ne sont pas dans le même terme rédex mais $c_2 \notin \mathcal{CA}(t)$ car ses deux occurrences sont dans le même terme rédex

NOUS POUVONS maintenant définir les systèmes Presburger étendus, qui ont pour but de remplacer les systèmes Presburger. Très proches de ces derniers, leurs différences sont mises en valeur :

DÉFINITION 21 (SYSTÈME PRESBURGER ÉTENDU).
Un système de réécriture $\mathcal{R}$ est dit Presburger étendu *si pour chaque symbole défini $\hat{f}$, il contient :*
– *une* règle de base *:*

$$\hat{f}(\mathbf{0}, \vec{c}; \vec{x}) \to r_1^{\hat{f}}$$

– *une* règle inductive *de l'une des deux formes suivantes :*

$$\begin{cases} \hat{f}(\mathbf{succ}(i), \vec{c}; \vec{x}) \to r_2^{\hat{f}}[\hat{f}(i, \vec{c}; \vec{x})]_A \\ ou \\ \hat{f}(\mathbf{succ}(i), \vec{c}; \vec{x}) \to r_2^{\hat{f}}[\hat{f}(i, c_1, \ldots, c_{k-1}, \mathbf{succ}(c_k), c_{k+1}, \ldots, c_n; \vec{x})]_A \end{cases}$$

tels que :

1. *A est un sous-ensemble fini des positions mutuellement parallèles autres que la racine de $r_2^{\hat{f}}$;*
2. *$r_2^{\hat{f}}[\hat{f}(i, \vec{c}; \vec{x})]_A$ et $r_2^{\hat{f}}[\hat{f}(i, c_1, \ldots, c_{k-1}, \mathbf{succ}(c_k), c_{k+1}, \ldots, c_k; \vec{x})]_A$ sont des termes réguliers ;*
3. ***i est compteur principal potentiel de $r_2^{\hat{f}}$;***
4. ***j est compteur auxiliaire potentiel de $r_1^{\hat{f}}$ et $r_2^{\hat{f}}$;***
5. ***chaque symbole défini apparaîssant dans $r_1^{\hat{f}}$ ou $r_2^{\hat{f}}$ a une précédence inférieure à $\hat{f}$;***
6. *le symbole racine de $r_2^{\hat{f}}$ est un symbole constructeur ;*
7. *chaque variable usuelle doit apparaître dans $r_1^{\hat{f}}$ ou $r_2^{\hat{f}}$.*

(la notion d'approximation est remplacée par les notions de compteur principal potentiel et de compteur auxiliaire potentiel)

Une grammaire primale étendue *est un triplet $(\Sigma, \mathcal{R}, t^\star)$ où :*
– *Σ est une signature primale ;*
– ***$\mathcal{R}$ est un système Presburger étendu ;***
– *$t^\star$ est un terme primal sur Σ.*
La définition de langage engendré s'étend naturellement.

.....

Idée sous-jacente.

La notion d'approximation interdit, dans un système Presburger, d'utiliser dans $r_1^{\hat{f}}$ et $r_2^{\hat{f}}$ des termes rédex dont les paramètres soient des termes *autres que des variables.*

Un système Presburger étendu n'impose pas (moins) de conditions sur la forme des paramètres des termes rédex.

.....

LE GAIN en « ergonomie » se voit particulièrement sur les exemples suivants.

Exemple 13.

Soit la définition de $\hat{\mathcal{N}}$ en §3.5 page 17 et un symbole défini $\hat{g}$: $\mathsf{compteur} \times \mathsf{compteur} \to \mathsf{entier}$ de précédence supérieure à $\hat{\mathcal{N}}$ dont on suppose donné un système Presburger non précisé.

On souhaite définir le système Presburger d'un symbole défini $\hat{f}$ de précédence supérieure à $\hat{g}$. Alors *(les causes de non système Presburger sont soulignées, celles de non Presburger étendu sont encadrées)* :

1. si le système contient la règle :
$$\hat{f}(\mathbf{0}, j) \to \hat{\mathcal{N}}(\underline{\mathbf{succ}(j)})$$
ce n'est pas un système Presburger, en effet : $\underline{\mathbf{succ}(j)}$ n'est pas une sous-suite de i, j (en revanche j seul le serait).
Mais ce peut être un système Presburger étendu car j est bien compteur auxiliaire potentiel de $\hat{\mathcal{N}}(\mathbf{succ}(j))$ (j n'y apparaît qu'une fois).
2. si le système contient la règle :
$$\hat{f}(\mathbf{0}, j) \to \hat{g}(\underline{\boxed{j}, \boxed{j}})$$
ce n'est pas un système Presburger, en effet : $\underline{j, j}$ n'est pas une sous-suite de i, j.
Ce n'est pas non plus un système Presburger étendu car j n'est pas compteur auxiliaire potentiel de $\hat{g}(\boxed{j}, \boxed{j})$: en effet il apparaît *dans les deux paramètres* (en revanche ce serait possible avec $\hat{g}(i, \mathbf{succ}(j))$ ou $\hat{g}(j, \mathbf{succ}(k))$).
3. si le système contient la règle :
$$\hat{f}(\mathbf{succ}(i), j) \to r(\hat{g}(\underline{\mathbf{succ}(i), j}), \hat{f}(i, \mathbf{succ}(j))) \quad (\text{où } r : \mathsf{entier} \times \mathsf{entier} \to \mathsf{entier})$$
ce n'est pas un système Presburger, en effet : $\underline{\mathbf{succ}(i), j}$ n'est pas une sous-suite de i, j.
En revanche ce peut être un système Presburger étendu car :
   – i est bien compteur principal potentiel de $\hat{g}(\mathbf{succ}(i), j)$ (car sa seule occurence est dans le premier paramètre),
   – j est bien compteur auxiliaire potentiel de $\hat{g}(\mathbf{succ}(i), j)$ (pour les mêmes raisons que précédemment).
4. si le système contient la règle :
$$\hat{f}(\mathbf{succ}(i), j) \to r(\hat{g}(\underline{\mathbf{succ}(i), \boxed{i}}), \hat{f}(i, \mathbf{succ}(j)))$$
ce n'est pas un système Presburger, en effet : $\underline{\mathbf{succ}(i), i}$ n'est pas une sous-suite de i, j.
Ce ne peut pas non plus être un système Presburger étendu car i n'est pas compteur principal potentiel de $\hat{g}(\mathbf{succ}(i), \boxed{i})$ (c'est un compteur quelconque).
5. le système suivant n'est pas Presburger mais est Presburger étendu :
$$\hat{f}(\mathbf{0}, j) \to r(\hat{\mathcal{N}}(\underline{\mathbf{succ}(j)}), \hat{g}(j, \underline{\mathbf{0}}))$$
$$\hat{f}(\mathbf{succ}(i), j) \to r\big(\hat{g}(\underline{i+j}, \underline{\mathbf{succ}(\mathbf{0})}), r(\hat{\mathcal{N}}(\underline{\mathbf{succ}(i)}), \hat{f}(i, \mathbf{succ}(j)))\big)$$

4.2 Équivalence avec les grammaires primales

LE BUT ICI est de démontrer que les grammaires primales étendues sont équivalentes aux grammaires primales.

On a d'abord trivialement la propriété suivante :

Propriété 4.
Toute grammaire primale est équivalente à une grammaire primale étendue.

Démonstration. En effet, tout système Presburger est trivialement un système Presburger étendu *(on vérifie facilement que chaque terme rédex appartenant à l'approximation, i — resp. j — est compteur principal — resp. auxiliaire — potentiel)*. Par conséquent toute grammaire primale est une grammaire primale étendue. □

MONTRONS MAINTENANT que toute grammaire primale étendue est équivalente à une grammaire primale.

Lemme 1.
Soit un terme rédex $\hat{f}(n, \vec{p}; \vec{x})$ où $n, p_1, \ldots, p_k$ sont des *expressions compteurs*.
Alors il existe un terme primal (classique) équivalent dont tous les sous-termes de type compteur sont des *variables* compteurs *(i.e. pas des expressions compteurs quelconques)*.

.....

Idée sous-jacente.
Ce qui compte ici, c'est que les expressions compteurs — donc des termes quelconques — peuvent être « décomposées » pour se ramener au cas de variables compteurs seules.

.....

Démonstration.

.....

Idée de la preuve.
Pour tout i, on note $\{c'_{i,1}, \ldots, c'_{i,m_i}\} = \mathcal{CV}ar(p_i)$ *(ordonnés arbitrairement)* et $\{c'_{0,1}, \ldots, c'_{0,m_0}\} = \mathcal{CV}ar(n)$; nous allons construire un $\hat{f}'$ tel que :

$$\hat{f}'(c'_{0,1}, \ldots, c'_{0,m_0}, c'_{1,1}, \ldots, c'_{1,m_1}, \ldots; \vec{x}) \equiv \hat{f}(p_1, \ldots, p_k; \vec{x})$$

Schématiquement :

$$\begin{array}{ccccc} \hat{f}(& n, & \ldots & p_k & ; \vec{x}) \\ & \downarrow & & \downarrow & \\ \hat{f}'(& \overbrace{c'_{0,1}, \ldots, c'_{0,m_0}}, & \ldots & \overbrace{c'_{k,1}, \ldots, c'_{k,m_k}} & ; \vec{x}) \end{array}$$

La structure de chaque expression compteur est encodée dans la *structure* de $\hat{f}'$. e.g. chaque addition dans une expression compteur est remplacé par un symbole défini dont la structure encode l'addition correspondante.

.....

– On note u le terme $\hat{f}(n, \vec{p}; \vec{x})$, u' le terme recherché, et le système Presburger de $\hat{f}$ de la façon suivante :

$$\hat{f}(\mathbf{0}, \vec{c}; \vec{x}) \longrightarrow r_1^{\hat{f}}$$
$$\hat{f}(\mathbf{succ}(i), \vec{c}; \vec{x}) \longrightarrow r_2^{\hat{f}}[\hat{f}(i, c_1, \ldots, c_{j-1}, \mathbf{succ}(c_j), c_{j+1}, \ldots)]_A$$

(on se place dans le cas le plus général où $\hat{f}$ a un compteur auxiliaire)
– On procède par induction nœthérienne sur l'extension lexicographique $(\prec, \leq)$, *où :*
  - $\prec$ *désigne la précédence entre symboles définis ;*
  - $\leq$ *désigne l'ordre sur la hauteur du terme n ;*

  On note $\mathcal{HI}(t)$ le terme obtenu par application de l'$\mathcal{H}$ypothèse d'$\mathcal{I}$nduction au terme t.

  1. si $n = \mathbf{0}$ ou plus généralement, n est une constante :
     alors on peut réécrire le terme avec le système Presburger de $\hat{f}$ et on applique l'hypothèse d'induction à tous les éventuels termes rédex ainsi introduits *(et dont la tête est nécessairement de précédence inférieure)* ;
  2. si n est de la forme $n' + n''$:
     - On suppose que n contient au moins une variable — sinon, n est une constante, cf. 1 — on peut donc se ramener par associativité de l'addition à un terme équivalent tel que n' est une variable. Alors on note :
       – $\hat{g_1}(\vec{d_1}; \vec{y_1}), \ldots, \hat{g_p}(\vec{d_p}; \vec{y_p})$ les termes rédex apparaîssant dans $r_2^{\hat{f}}$ et $r_1^{\hat{f}}$;

– $R_2^{\hat{f}}$ le terme obtenu en remplaçant dans $r_2^{\hat{f}}$ chaque $\hat{g}_i(\vec{d};\vec{y})$ par

$$\mathcal{HI}\left(\hat{g}_i(\vec{d_i};\vec{y_i})\left\{\begin{matrix} i\leftarrow i+n'' \\ c_1\leftarrow p_1 \\ \cdots \\ c_k\leftarrow p_k \end{matrix}\right\}\right)$$

– $c'_{0,1} \stackrel{\text{def}}{=} n'$.

- Puis on pose :

$$\begin{aligned}
&\hat{f}'(\mathbf{0}, c'_{0,2}, \ldots, c'_{0,m_0}, c'_{1,1}, \ldots, c'_{k,m_k}; \vec{x}) \\
&\quad \to \mathcal{HI}\left(\hat{f}(n'', \vec{p}; \vec{x})\right) \\
&\hat{f}'(\mathbf{succ}(i), c'_{0,2}, \ldots, c'_{0,m_0}, c'_{1,1}, \ldots, c'_{k,m_k}; \vec{x}) \\
&\quad \to R_2^{\hat{f}}[\hat{f}'(i, c'_{0,2}, \ldots, c'_{0,m_0}, c'_{1,1}, \ldots, c'_{j-1,m_{j-1}}, \mathbf{succ}(c'_{j,1}), c'_{j,2}, \ldots; \vec{x})]_A
\end{aligned}$$

où :
– $c'_{0,2}, \ldots, c'_{0,m_0}$ sont les variables de n'' (« récupérés » par induction sur $\hat{f}(n'', \vec{p}; \vec{x})$) ;
– on donne à $\hat{f}'$ le rang suffisant pour que les symboles définis introduits par induction soient dans l'approximation ;

Ici l'hypothèse d'induction s'applique à un symbole dont le rang ne décroît pas ; en revanche, la hauteur de n'' est bien inférieure à celle de $n' + n''$.

(*Remarque :* en toute rigueur, si un quelconque c_i n'est utilisé par aucun $\hat{g}_i(\vec{d_i};\vec{y_i})$, alors le p_i correspondant n'est pas décomposé par l'hypothèse d'induction ; cependant un tel c_i serait alors inutile *(i.e. le symbole construit en retirant ce paramètre du profil de $\hat{f}$ est équivalent à $\hat{f}$)*, on peut donc remplacer p_i par une nouvelle variable.)

- Suivant la même méthode que dans la propriété 3 page 16, (i.e. en se donnant un énumérateur, puis en procédant par récurrence sur l'assignation ainsi obtenue du compteur principal, ce qui donne l'égalité des langages engendrés), on montre :

$$\hat{f}'(n', c'_{0,2}, \ldots, c'_{k,m_k}; \vec{x}) \equiv \hat{f}(n' + n'', \vec{p}; \vec{x})$$

- On pose donc $u' \stackrel{\text{def}}{=} \hat{f}'(n', c'_{0,2}, \ldots, c'_{k,m_k}; \vec{x})$ qui vérifie bien la propriété car n' est une variable.

3. si n est une variable, cf. 2 page précédente en prenant $n' = n$ et $n'' = \mathbf{0}$ *(car $n = n + \mathbf{0}$)*;
4. si n est de la forme $\mathbf{succ}(n')$, cf. 2 page ci-contre en prenant $n'' = \mathbf{succ}(\mathbf{0})$ *(car $\mathbf{succ}(n) = n + \mathbf{succ}(\mathbf{0})$)*;

□

Propriété 5.
Toute grammaire primale étendue est équivalente à une grammaire primale.

Démonstration.

. . . . . .

Idée de la preuve.
Le but est de construire une grammaire primale à partir de la grammaire primale étendue. Vues les deux définitions, il s'agit principalement de modifier la grammaire étendue de sorte à respecter la contrainte d'appartenance à l'approximation.

Trois contraintes peuvent empêcher d'atteindre ce but :
– l'approximation n'est définie que pour un terme ayant des variables pour paramètres (i.e. pas des expressions quelconques) *Ce problème est réglé par l'application du lemme 1 page précédente.*
– le compteur principal doit être placé en première position donc impossible si jamais une variable apparaît *avant* le compteur principal. *Ce problème est réglé par la notion de compteur principal potentiel.*
– tous les paramètres du symbole défini doivent être différents donc impossible si une variable apparaît *deux fois* en tant que paramètre d'un même terme rédex. *Ceci est réglé par la notion de compteur auxiliaire potentiel.*

. . . . . .

Soit une grammaire primale étendue $(\Sigma, \mathcal{R}, t^\star)$.
Soit un symbole défini $\hat{f}$ de compteur principal i et de compteur auxiliaire j.
On note $u_1, \ldots, u_m$ les sous-termes rédex de $r_1^{\hat{f}}$ et $r_2^{\hat{f}}$.
Par définition d'un système Presburger étendu, on a pour tout $k \in 1..m$:

1. i (resp. j) est compteur principal (resp. auxiliaire) potentiel de u_k ;
2. $\mathcal{H}ead(u_k) \prec \hat{f}$.

Nous construisons un symbole défini $\hat{f}'$ « équivalent à $\hat{f}$ » mais de système Presburger classique, par induction sur la précédence de $\hat{f}$. Soit $k \in 1..m$, on note $u_k = \hat{g}(n, \vec{p}; \vec{t})$.

- L'application de l'hypothèse d'induction à u_k fournit un système Presburger pour $\hat{g}$; construisons maintenant le système Presburger de $\hat{f}$.
- On pose $u'_k = \hat{g}(n, \vec{p}; \vec{x})$ le terme tel que $u_k = u'_k \left\{ \begin{smallmatrix} x_1 \leftarrow t_1 \\ x_2 \leftarrow t_2 \\ \dots \end{smallmatrix} \right\}$; comme nous avons maintenant un système Presburger classique pour u'_k, on peut lui appliquer le lemme 1 page 22, ce qui nous donne un terme $u''_k = \hat{g}'(c_0, c_1, \dots; \vec{x})$ où $c_0, c_1, \dots$ sont des *variables compteurs*.
- Si $i \in \mathcal{V}ar(u''_k)$, alors i doit être le premier paramètre de u''_k, sinon u''_k ne peut pas appartenir à l'approximation de $\hat{f}'(i, \dots)$; or, i est compteur principal potentiel de u_k, donc :
  - soit $i \notin \mathcal{V}ar(u_k)$, et alors $i \notin \mathcal{V}ar(u''_k)$, donc le problème ne se pose pas ;
  - soit $i \in \mathcal{V}ar(n)$ (et $\forall p_l \in \vec{p} : i \notin \mathcal{V}ar(p_l)$) ; alors la démonstration de le lemme 1 page 22 montre que l'on peut imposer $c_0 = i$. De plus, comme $\forall p_l \in \vec{p} : i \notin \mathcal{V}ar(p_l)$, c_0 est la seule occurence de i, si cela n'avait pas été le cas u''_k aurait eu la forme $\hat{g}'(i, \dots, i, \dots)$ et n'aurait donc pas pu appartenir à l'approximation de $\hat{f}'(i, \dots)$.
- De même j étant compteur auxiliaire potentiel de $r_1^{\hat{f}}$ et de $r_2^{\hat{f}}$, il a au plus une occurence dans u_k.
- On peut donc distinguer deux types de variables compteurs :
  - celles qui apparaîssent entre i et j (donc *avant* j), notées $\vec{c_1}$;
  - celles qui apparaîssent *après* j, notées $\vec{c_2}$.

  Ces variables peuvent éventuellement se répéter *(i.e. il peut exister i_1, i_2 tel que $i_1 \neq i_2$ et $c_{i_1} = c_{i_2}$)*, ce qui pose problème pour l'appartenance de u''_k à l'approximation ; cependant ces variables-ci n'ayant pas de rôle particulier dans le système Presburger, on les renomme de sorte à ce qu'elles soient toutes distinctes ; les égalités entre variables seront rétablies en dehors du système Presburger.
- On pose alors :

$$\hat{f}'(\mathbf{0}, \vec{c_1}, j, \vec{c_2}; \vec{x}) \to r_1^{\hat{f}} \text{ où chaque } u_k \text{ est remplacé par } u''_k.$$

$$\hat{f}'(\mathbf{succ}(i), \vec{c_1}, j, \vec{c_2}; \vec{x}) \to r_2^{\hat{f}} \text{ où chaque } u_k \text{ est remplacé par } u''_k.$$

  Par construction, il s'agit bien d'un système Presburger (i.e. chaque $u''_k \in \mathcal{A}px(\hat{f}'(i, \vec{c_1}, j, \vec{c_2}))$).
- On pose enfin $t'^\star = t^\star$ dans lequel chaque sous-terme de la forme $\hat{f}(n, \vec{p}; \vec{x})$ est remplacé par $\hat{f}'(n, \vec{p}; \vec{t})$; alors $(\Sigma, \mathcal{R}', t'^\star)$ est bien une grammaire primale équivalente à $(\Sigma, \mathcal{R}, t^\star)$.

□

THÉORÈME 1.
Il est équivalent d'utiliser les grammaires primales étendues ou les grammaires primales.

AINSI PAR LA SUITE, le terme « grammaire primale » désignera indifféremment l'un ou l'autre formalisme.

D'une manière générale, les grammaires primales étendues sont plus pratiques lorsque la schématisation n'est qu'un outil. En revanche, lorsque l'objet d'étude est la schématisation en elle-même, les grammaires primales classiques sont plus pratiques — car plus restrictives.

Chapitre 5

Schémas de formules

NOUS POUVONS MAINTENANT (enfin) rentrer dans le vif du sujet : ce chapitre définit formellement les schémas de formules que nous allons utiliser. L'idée globale est :

1. Considérer les formules (qui sont dans, notre cas, des ensembles de clauses) comme des termes.

$$\{P(a); \neg P(x) \vee P(s(x))\}$$

   devient :

$$\cup(\{P(a)\}, \{\vee(\neg(P(x)), P(s(x)))\})$$

   (naturellement, cette écriture laborieuse ne sera pas utilisée telle quelle ; ce n'est qu'une formalisation nécessaire à l'utilisation des grammaires primales)

2. Utiliser les grammaires primales pour schématiser les termes ainsi obtenus : ce sont les « schémas de formules ».

$$\bigcup_{i=1}^{n+1} p(i)$$

   s'écrit :

$$\hat{\mathcal{F}}(\mathbf{succ}(n)) \text{ où : } \left\{ \begin{array}{c} \hat{\mathcal{F}}(\mathbf{0}) \to \square \\ \hat{\mathcal{F}}(\mathbf{succ}(i)) \to p(\hat{N}(\mathbf{succ}(i))) \vee \hat{\mathcal{F}}(i) \end{array} \right\}$$

5.1 Préambule : signature clausale

NOUS FORMALISONS ici le fait de considérer des formules comme des termes : partant d'un langage du premier ordre nous définissons une « signature clausale ». Les termes sur cette signature correspondent aux formules sur le langage considéré.

DÉFINITION 22 (SIGNATURE CLAUSALE).
Soit $(\mathcal{S}, \mathcal{F}, \mathcal{P})$ *un langage du premier ordre multisorté.*
On appelle signature clausale *sur* $(\mathcal{S}, \mathcal{F}, \mathcal{P})$ *toute signature :*
– *contenant les sortes* littéral, clause, *et* conjonction *;*
– *contenant les symboles fonctionnels suivants :*

$$\begin{array}{rrcll} P : & \mathcal{D}om(P) & \longrightarrow & \text{littéral} & (P \in \mathcal{P}) \\ \neg : & \text{littéral} & \longrightarrow & \text{littéral} & \\ (\cdot)_{\text{clause}} : & \text{littéral} & \longrightarrow & \text{clause} & \\ \square : & & \longrightarrow & \text{clause} & \\ \vee : & \text{clause} \times \text{clause} & \longrightarrow & \text{clause} & \\ (\cdot)_{\text{conjonction}} : & \text{clause} & \longrightarrow & \text{conjonction} & \\ \emptyset : & & \longrightarrow & \text{conjonction} & \\ \wedge : & \text{conjonction} \times \text{conjonction} & \longrightarrow & \text{conjonction} & \end{array}$$

ainsi que les symboles de $\mathcal{F}$.

Idée sous-jacente.
Les connectifs et les prédicats sont simplement vus comme des symboles de fonction.

Remarque 6.
Les symboles infixés $(\cdot)_{\mathsf{clause}}$ et $(\cdot)_{\mathsf{conjonction}}$ ne servent qu'à convertir les types. Nous les écrirons au début mais nous les omettrons assez rapidement par souci de lisibilité.

Exemple 14 *(Signature clausale).*
Soit le langage du premier ordre suivant :
- $\mathcal{S} = \{\mathsf{t}\}$
- $\mathcal{P}$ contient P, Q (d'arité 1) et R (d'arité 2) ;
- $\mathcal{F} = \{a, b, s, f, g\}$ où :

$$\begin{array}{rr} a: & \longrightarrow \mathsf{t} \\ b: & \longrightarrow \mathsf{t} \\ s: & \mathsf{t} \longrightarrow \mathsf{t} \\ f: & \mathsf{t} \times \mathsf{t} \longrightarrow \mathsf{t} \\ g: & \mathsf{t} \times \mathsf{t} \longrightarrow \mathsf{t} \end{array}$$

Alors la signature suivante, dont les sortes sont t, littéral, clause et conjonction, est une signature clausale :

$$\begin{array}{rrl} a: & & \longrightarrow \mathsf{t} \\ b: & & \longrightarrow \mathsf{t} \\ s: & \mathsf{t} & \longrightarrow \mathsf{t} \\ f: & \mathsf{t} \times \mathsf{t} & \longrightarrow \mathsf{t} \\ g: & \mathsf{t} \times \mathsf{t} & \longrightarrow \mathsf{t} \\ P: & \mathsf{t} & \longrightarrow \mathsf{littéral} \\ Q: & \mathsf{t} & \longrightarrow \mathsf{littéral} \\ R: & \mathsf{t} \times \mathsf{t} & \longrightarrow \mathsf{littéral} \\ \square: & & \longrightarrow \mathsf{clause} \\ \neg: & \mathsf{littéral} & \longrightarrow \mathsf{littéral} \\ \vee: & \mathsf{littéral} \times \mathsf{clause} & \longrightarrow \mathsf{clause} \\ \emptyset: & & \longrightarrow \mathsf{conjonction} \\ \wedge: & \mathsf{clause} \times \mathsf{conjonction} & \longrightarrow \mathsf{conjonction} \end{array}$$

DÉFINITION 23 (SÉMANTIQUE).
Soit une signature clausale Σ sur un langage $\mathcal{L}$, on appelle Σ-algèbre clausale *toute Σ-algèbre telle que :*
- littéral, clause, conjonction *ont pour domaine $\{\mathbf{V}, \mathbf{F}\}$;*
- *les symboles $\neg, (\cdot)_{\mathsf{clause}}, \square, \vee, (\cdot)_{\mathsf{conjonction}}, \emptyset, \wedge$ s'interprètent respectivement comme la négation, la fonction identité, la fonction constante égale à $\mathbf{F}$, la disjonction, la fonction identité, la fonction constante égale à $\mathbf{V}$, et la conjonction.*

Idée sous-jacente.
On adapte simplement la notion usuelle d'interprétation au nouvel objet.

Remarque 7.
Il est évident qu'à chaque interprétation classique $\mathcal{I}$ sur $\mathcal{L}$, on peut associer une Σ-algèbre clausale $\mathcal{I}'$ sur Σ.

On peut alors dire qu'une clause C est équivalente à un terme t sur une signature clausale si et seulement si $\mathcal{I}(C) = \mathcal{I}'(t)$ quel que soit $\mathcal{I}$.

Notation 2.
Par la suite nous garderons les notations usuelles pour les clauses et ensembles de clauses. Ainsi, au lieu d'écrire

$$\wedge(P(a), \wedge(\neg(Q(s(b))), \wedge(\vee(P(f(a,b)), \vee(\neg(R(c,a)), \square)), \emptyset)))$$

Nous écrirons plutôt :

$$\{P(a), \neg Q(s(b)), P(f(a,b)) \vee \neg R(c,a)\}$$

Plus généralement :
– $A_1 \vee A_2 \vee \cdots \vee A_n$ désigne le terme $\vee(\cdots \vee (\vee(A_1, A_2), A_3) \ldots, A_n)$,
– et $\{A_1, \ldots, A_n\}$ désigne le terme $\wedge(\cdots \wedge (\wedge(A_1, A_2), A_3) \ldots, A_n)$.
Ce qui est justifié car la définition 23 page ci-contre implique que $\vee$ et $\wedge$ sont associatives et commutatives dans toute Σ-algèbre clausale.

5.2 Syntaxe

IL S'AGIT MAINTENANT d'utiliser les grammaires primales pour schématiser les formules, préalablement représentées par des termes. La définition découle assez naturellement des définitions de signature clausale et primale.

DÉFINITION 24 (SCHÉMA DE FORMULES).
Soit $\mathcal{L} = (\Sigma, \mathcal{F}, \mathcal{P})$ un langage du premier ordre multisorté tel qu'aucun symbole de $\mathcal{F}$ ne soit de type littéral, clause, *ou* conjonction.

On appelle signature de schéma de formules *sur $\mathcal{L}$ toute signature primale et clausale sur $\mathcal{L}$.*
On appelle schéma de formules *toute grammaire primale sur une signature de schéma de formules dont l'axiome est de sorte* conjonction *; on interdit de plus les variables de types* littéral, clause *et* conjonction.

Ainsi le langage engendré par un schéma de formules est un ensemble d'ensembles de clauses.

On appelle *conjonction définie* (resp. *clause définie*) tout symbole défini de type conjonction (resp. clause).
On appelle *conjonction primale* (resp. *clause primale*) tout terme primal de type conjonction (resp. clause).
Une clause primale pourra aussi être appelée *schéma de clauses.*

. . . . . .

Idée sous-jacente.
C'est simplement un héritage double des grammaires primales et signatures clausales.

. . . . . .

Remarque 8.
Tout comme on pouvait précédemment désigner une grammaire primale par son axiome, on désignera de même, lorsque le contexte s'y prête, un schéma de formules par son axiome (qui est une conjonction primale).

Notation 3.
Les conjonctions définies seront notées [1] $\hat{\mathcal{F}}$, $\hat{\mathcal{G}}$, $\hat{\mathcal{H}}$.
Les clauses définies seront notées $\hat{F}$, $\hat{G}$, $\hat{H}$.

Exemple 15 *(Conjonction primale).*
(ou schéma de formules dont la signature et le système de réécriture ne sont pas représentés...)

$$((P(a,x_1))_{\mathsf{clause}})_{\mathsf{conjonction}}$$
$$\hat{\mathcal{F}}(c_1, c_2; b)$$
$$((A)_{\mathsf{clause}})_{\mathsf{conjonction}}$$
$$(\hat{F}(c_2; x_1))_{\mathsf{conjonction}}$$

ou

$$\left\{((P(a,x_1))_{\mathsf{clause}})_{\mathsf{conjonction}}; \hat{\mathcal{F}}(c_1, c_2; b); ((A)_{\mathsf{clause}})_{\mathsf{conjonction}}; (\hat{F}(c_2; x_1))_{\mathsf{conjonction}}\right\}$$

Ce que nous abrégerons en :

$$\begin{array}{c} P(a,x_1) \\ \hat{\mathcal{F}}(c_1, c_2; b) \\ A \\ \hat{F}(c_2; x_1) \end{array} \quad \text{ou} \quad \left\{P(a,x_1); \hat{\mathcal{F}}(c_1, c_2; b); A; \hat{F}(c_2; x_1)\right\}$$

[1] sauf pour $\hat{\mathcal{N}}$ qui reste dédié à la représentation des entiers naturels, voir §3.5 page 17

5.3 Sémantique

LA SÉMANTIQUE s'établit assez naturellement à partir des sémantiques des grammaires primales et des signatures clausales. Cependant souvenons-nous que notre but est de construire des *schémas de preuves*. Cela signifie que l'on veut une preuve pour *chaque instance* de notre schéma de formules. Dans un cadre de démonstration par réfutation il faut donc *réfuter* chaque instance de ce schéma. Ainsi l'interprétation d'un schéma sera fausse si et seulement si l'interprétation de *chaque instance* est fausse.

À noter que cette sémantique est complétement différente de celle utilisées dans les travaux antérieurs sur les schématisations de termes (cf. par exemple [Sal92, Pel97]). En effet, les variables entières représentent ici des paramètres (que l'on peut voir comme des quantificateurs existentiels) alors que dans les travaux existants, elles sont traitées exactement commme des variables usuelles (donc quantifiées universellement). Ainsi, un schéma sera dit "réfutable" si chacune de ses instances est réfutable. Par exemple, le schéma $\{P(a),\ \neg P(f^n(a))\}$, n'est pas réfutable (sauf si $n = 0$), alors que l'ensemble de clauses $\{P(a),\ \neg\forall n\ P(f^n(a))\}$ l'est.

DÉFINITION 25 (INTERPRÉTATION D'UN SCHÉMA DE FORMULES).
Soient :
- *$\mathcal{L}$ un langage du premier ordre,*
- *une Σ-algèbre clausale $\mathcal{I}$ sur $\mathcal{L}$,*
- *un schéma de formules $\mathcal{S}$ dont la signature est basée sur $\mathcal{L}$.*

On définit l'interprétation $\tilde{\mathcal{I}}$ de $\mathcal{S}$ de la façon suivante :

$$\tilde{\mathcal{I}} \not\models \mathcal{S} \text{ ssi. } \forall t \in L(\mathcal{S}), \mathcal{I} \not\models t$$

.....

Idée sous-jacente.
Un schéma de formules est réfutable si et seulement si toutes ses instances sont réfutables.

.....

Remarque 9.
On aurait pu écrire de manière équivalente :

$$\tilde{\mathcal{I}} \models \mathcal{S} \text{ ssi. } \exists t \in L(\mathcal{S}) : \mathcal{I} \models t$$

mais pas (voir le début du paragraphe) :

$$\tilde{\mathcal{I}} \models \mathcal{S} \text{ ssi. } \forall t \in L(\mathcal{S}) : \mathcal{I} \models t$$

Exemple 16.
Soit le schéma de formules suivant :

$$P(a)$$
$$\neg P(x) \vee P(f(x))$$
$$\neg P(\hat{f}(n; a))$$

où :

$$\hat{f}(\mathbf{0}; x) \rightarrow x$$
$$\hat{f}(\mathbf{succ}(n); x) \rightarrow f(\hat{f}(n; x))$$

Ce schéma est insatisfaisable si et seulement si il est insatisfaisable pour toute valeur de n.

DÉFINITION 26 (ÉQUIVALENCE DE DEUX SCHÉMAS DE FORMULES).
Deux schémas de formules seront dits équivalents (noté $\equiv$) si et seulement si ils ont même valeur de vérité quelle que soit l'interprétation.

Remarque 10.
Les schémas de formules héritent aussi de la définition d'équivalence entre grammaires primales. Cette dernière implique trivialement l'équivalence telle qu'on vient de la définir *(car les langages engendrés sont alors identiques).*

5.4 Exemple : le problème des pigeonniers

AFIN DE METTRE en pratique les définitions introduites, nous modélisons ici l'exemple récurrent du problème des pigeonniers (voir §1.1 page 1 pour une présentation générale de ce problème) *via* des schémas de formules.

Soient :
- n pigeonniers,
- $n+1$ pigeons.

On note $p(i,j)$ le prédicat « le pigeon i va dans le pigeonnier j ».

On modelise le problème par raffinements successifs depuis l'énoncé en langue naturelle jusqu'à des schémas de formules.

1. $\boxed{\text{« Chaque pigeon va dans un pigeonnier »}}$

$$\rightsquigarrow$$

$$\forall i \in 1..n+1 : \exists j \in 1..n \; tq. \; p(i,j)$$

$$\rightsquigarrow$$

$$\bigcup_{i=1}^{n+1} \bigvee_{j=1}^{n} p(i,j)$$

$$\rightsquigarrow$$

$$\hat{\mathcal{F}}_1(\mathbf{succ}(n), n)$$

$$\text{où} \quad \begin{cases} \left.\begin{array}{r} \hat{\mathcal{F}}_1(\mathbf{0}, j) \to \emptyset \\ \hat{\mathcal{F}}_1(\mathbf{succ}(i), j) \to \hat{F}(j, \mathbf{succ}(i)) \cup \hat{\mathcal{F}}_1(\mathbf{succ}(i), j) \end{array}\right\} \bigcup_{i=1}^{n+1} \cdots \\ \left.\begin{array}{r} \hat{F}(\mathbf{0}, i) \to \square \\ \hat{F}(\mathbf{succ}(j), i) \to p(\hat{\mathcal{N}}(i), \hat{\mathcal{N}}(\mathbf{succ}(j))) \vee \hat{F}(j, i) \end{array}\right\} \bigvee_{j=1}^{n} \cdots \\ \hat{\mathcal{F}}_1 \succ \hat{F} \succ \hat{\mathcal{N}} \end{cases}$$

Mais cette dernière écriture *ne correspond pas* à une grammaire primale étendue, en effet i n'est pas compteur principal potentiel de $\hat{F}(j, \boxed{\mathbf{succ}(i)})$ dans

$$\hat{\mathcal{F}}_1(\mathbf{succ}(i), j) \to \hat{F}(j, \mathbf{succ}(i))$$

Cependant, on peut modifier $\hat{\mathcal{F}}_1$ de la façon suivante :

$$\hat{\mathcal{F}}_1(\mathbf{0}, j, i) \to \emptyset$$
$$\hat{\mathcal{F}}_1(\mathbf{succ}(i'), j, i) \to \hat{F}(j, i) \cup \hat{\mathcal{F}}(i', j, \mathbf{succ}(i))$$

et on pose :

$$\hat{\mathcal{F}}_1(\mathbf{succ}(n), n, \mathbf{succ}(\mathbf{0}))$$

. . . . . .

Idée sous-jacente.
On a remplacé :

$$\bigcup_{i=1}^{n+1} \bigvee_{j=1}^{n} p(i,j)$$

par :

$$\bigcup_{\substack{i'=1..n+1 \\ i+i'=n+2}} \bigvee_{j=1}^{n} p(i,j)$$

. . . . . .

2. $\boxed{\text{« Chaque pigeonnier ne contient qu'un pigeon »}}$

$$\rightsquigarrow$$

$$\forall i \in 1..n, \forall j \in 1..n+1, \forall k \in 1..n+1, k \neq j : \; p(j,i) \Rightarrow \neg p(k,i)$$

$$\rightsquigarrow$$

$$\bigcup_{i=1}^{n} \bigcup_{j=1}^{n+1} \bigcup_{\substack{k=1..n+1 \\ k \neq j}} \neg p(j,i) \vee \neg p(k,i)$$

$$\rightsquigarrow$$

$$\bigcup_{i=1}^{n} \bigcup_{j=1}^{n+1} \bigcup_{k<j} \neg p(j,i) \vee \neg p(k,i)$$

$$\rightsquigarrow$$

$$\bigcup_{i=1}^{n} \bigcup_{j=1}^{n+1} \bigcup_{k=1}^{j-1} \neg p(j,i) \vee \neg p(k,i)$$

(élimination des redondances)

Ici, le même type de problème surgit : on ne peut pas transcrire littéralement cette expression en termes de grammaires primales. Nous donnons donc une expression, équivalente à la précédente, plus complexe mais qui se traduit mieux en grammaires primales.

$$\rightsquigarrow$$

$$\bigcup_{\substack{i'=1..n \\ i+i'=n+1}} \bigcup_{j=1..n+1} \bigcup_{k=1..j-1} \neg p(j,i) \vee \neg p(k,i)$$

$$\rightsquigarrow$$

$$\hat{\mathcal{F}}_2(n, \mathbf{succ}(n), \mathbf{succ}(\mathbf{0}), \mathbf{succ}(\mathbf{0}))$$

$$\text{où :} \begin{cases} \left.\begin{array}{r} \hat{\mathcal{F}}_2(\mathbf{0}, c_1, c_2, i) \to \emptyset \\ \hat{\mathcal{F}}_2(\mathbf{succ}(i'), c_1, c_2, i) \to \hat{\mathcal{G}}(c_1, c_2, i) \cup \hat{\mathcal{F}}_2(i', c_1, c_2, \mathbf{succ}(i)) \end{array}\right\} \Leftrightarrow \bigcup_{\substack{i'=1..n \\ i+i'=n+1}} \cdots \\ \left.\begin{array}{r} \hat{\mathcal{G}}(\mathbf{0}, c_2, i) \to \emptyset \\ \hat{\mathcal{G}}(\mathbf{succ}(j), c_2, i) \to \hat{\mathcal{H}}(j, j, c_2, i) \cup \hat{\mathcal{G}}(j, c_2, i) \end{array}\right\} \Leftrightarrow \bigcup_{j=1..n+1} \cdots \\ \left.\begin{array}{r} \hat{\mathcal{H}}(\mathbf{0}, j, k, i) \to \emptyset \\ \hat{\mathcal{H}}(\mathbf{succ}(k'), j, k, i) \to \neg p(\hat{\mathcal{N}}(j), \hat{\mathcal{N}}(i)) \vee \neg p(\hat{\mathcal{N}}(k), \hat{\mathcal{N}}(i)) \\ \cup \hat{\mathcal{H}}(k', \mathbf{succ}(k), i) \end{array}\right\} \Leftrightarrow \bigcup_{k=1..j-1} \cdots \\ \hat{\mathcal{F}}_2 \succ \hat{\mathcal{G}} \succ \hat{\mathcal{H}} \succ \hat{\mathcal{N}} \end{cases}$$

Mais une fois encore, on n'a pas exactement une grammaire primale, en effet j n'est pas compteur principal potentiel dans :

$$\hat{\mathcal{G}}(\mathbf{succ}(j), c_2, i) \to \hat{\mathcal{H}}(j, \boxed{j}, c_2, i) \cup \hat{\mathcal{G}}(j, c_2, i)$$

En fait, si j est utilisé dans la borne supérieure de k, on ne peut pas réutiliser j dans le « corps » de l'itération ; on modifie donc $\hat{\mathcal{G}}$ et $\hat{\mathcal{H}}$ de façon que j n'apparaisse pas dans le corps, ce qui donne :

$$\hat{\mathcal{G}}(\mathbf{0}, c_2, i) \to \emptyset$$

$$\hat{\mathcal{G}}(\mathbf{succ}(j), c_2, i) \to \hat{\mathcal{H}}(j, c_2, i) \cup \hat{\mathcal{G}}(j, c_2, i)$$

et :

$$\hat{\mathcal{H}}(\mathbf{0}, k, i) \to \emptyset$$

$$\hat{\mathcal{H}}(\mathbf{succ}(k'), k, i) \to \neg p(\hat{t}(k, k'), \hat{\mathcal{N}}(i)) \vee \neg p(\hat{\mathcal{N}}(k), \hat{\mathcal{N}}(i)) \cup \hat{\mathcal{H}}(k', \mathbf{succ}(k), i)$$

où :

$$\begin{array}{rl} \hat{t}(\mathbf{0}, d_2) \to \hat{\mathcal{N}}(d_2) & \textit{(additionne les deux compteurs, puis} \\ \hat{t}(\mathbf{succ}(d_1), d_2) \to s(\hat{t}(d_1, d_2)) & \textit{traduit le résultat en un entier)} \end{array}$$

Enfin on pose :

$$\hat{\mathcal{F}}_2(n, \mathbf{succ}(n), \mathbf{succ}(\mathbf{succ}(\mathbf{0})), \mathbf{succ}(\mathbf{0}))$$

. . . . . .

Idée sous-jacente.
On a remplacé :

$$\bigcup_{\substack{i'=1..n \\ i+i'=n+1}} \bigcup_{j=1..n+1} \bigcup_{k=1..j-1} \neg p(j,i) \vee \neg p(k,i)$$

par :

$$\bigcup_{\substack{i'=1..n \\ i+i'=n+1}} \bigcup_{j=1..n+1} \bigcup_{\substack{k=1..j-1 \\ k+k'=j}} \neg p(k+k',i) \vee \neg p(k,i)$$

. . . . . .

En conclusion, le schéma de formules à réfuter est le suivant :

$$\boxed{\hat{\mathcal{F}}_1(\mathbf{succ}(n), n, \mathbf{succ}(\mathbf{0})) \cup \hat{\mathcal{F}}_2(n, \mathbf{succ}(n), \mathbf{succ}(\mathbf{succ}(\mathbf{0})), \mathbf{succ}(\mathbf{0}))}$$

Chapitre 6

Schémas itérés

LA NOTION de schéma itéré est née en voulant caractériser précisément ce que les schémas de formules pouvaient représenter. L'objectif est de fournir un langage de représentation des schémas équivalent à celui que nous venons d'introduire, mais plus proche de la notation mathématique standard. En fin de compte, ces schémas itérés se sont progressivement substitués aux schémas de formules originels. En effet, ils sont beaucoup plus simples que ces derniers, et correspondent d'avantage à une écriture « naturelle » des schémas de formules.

Par exemple le schéma informel :

$$\bigcup_{i=1}^{n} \bigvee_{j=1}^{p} P(i,j)$$

s'écrit simplement :

$$\bigcup_{i=1}^{n} \bigvee_{j=1}^{p} P(\hat{i},\hat{j})$$

(qui est un schéma itéré).

Nous démontrons que cette définition est équivalente à la précédente en termes de pouvoir d'expression.

6.1 Clauses itérées

NOUS CONSIDÉRONS en premier lieu des *clauses* itérées i.e. des objets ne faisant pas intervenir la conjonction, e.g. :

$$\bigvee_{i=1}^{n} P(\hat{i})$$

Les résultats valables sur les clauses s'étendront naturellement aux schémas itérés en toute généralité, comme nous le verrons en §6.3 page 41.

6.1.1 Syntaxe

DÉFINITION 27 (LANGAGE DU 1ER ORDRE PRIMAL).

Soit :

- *un langage du 1er ordre $\mathcal{L} = (\mathcal{F}, \mathcal{P})$,*
- *une signature primale Σ dont les symboles constructeurs sont les éléments de $\mathcal{F}$.*

Alors on appelle langage du premier ordre primal *basé sur $\mathcal{L}$, le langage dont les prédicats sont dans $\mathcal{P}$ mais dont les termes peuvent appartenir à Σ* [1].

. . . . . .

Idée sous-jacente.

Il s'agit d'un langage du 1er ordre classique à ceci près que les termes peuvent être des termes primaux.

. . . . . .

On ne définit pas de sémantique, il s'agit juste d'une « brique » pour la définition suivante :

[1] i.e. ici, la signature primale ne sert *que pour les termes*, pas pour les clauses ou les conjonctions ; il peut donc y avoir des termes primaux, mais pas de clauses ou de conjonctions primales au sens défini plus haut.

DÉFINITION 28 (CLAUSE ITÉRÉE).
Soient
- *un langage du premier ordre $\mathcal{L}$,*
- *un langage du premier ordre primal basé sur $\mathcal{L}$, noté $\mathcal{LP}$.*

*On définit l'*ensemble des clauses itérées, *noté $\mathcal{C}onv(\mathcal{LP})$, comme le plus petit ensemble contenant $\square$ et les littéraux sur $\mathcal{LP}$ clos par disjonction et par l'opération suivante :*

$$\textit{si} \begin{cases} C \in \mathcal{C}onv(\mathcal{LP}), \\ i \textit{ est un compteur } \text{principal} \textit{ potentiel de } C, \\ j \textit{ est un compteur } \text{auxiliaire} \textit{ potentiel de } C, \\ n, p \textit{ sont des expressions compteurs.} \end{cases} \quad \textit{alors :} \bigvee_{\substack{i+j=n+p \\ i=1..n}} C \in \mathcal{C}onv(\mathcal{P}, \Sigma)$$

Ce qui nécessite de définir, pour toute clause itérée C, l'ensemble de ses compteurs quelconques, *noté $\widetilde{\mathcal{CQ}}(C)$ et l'ensemble de ses* compteurs auxiliaires potentiels, *noté $\widetilde{\mathcal{CA}}(C)$:*

C	$\widetilde{\mathcal{CQ}}(C)$	$\widetilde{\mathcal{CA}}(C)$
$\square$	$\emptyset$	$\emptyset$
$P(t_1, \ldots, t_k)$	$\mathcal{CQ}(t_1) \cup \cdots \cup \mathcal{CQ}(t_k)$	$\mathcal{CA}(t_1) \cup \cdots \cup \mathcal{CA}(t_k)$
$\neg P(t_1, \ldots, t_k)$	$\mathcal{CQ}(t_1) \cup \cdots \cup \mathcal{CQ}(t_k)$	$\mathcal{CA}(t_1) \cup \cdots \cup \mathcal{CA}(t_k)$
$C_1 \vee C_2$	$\widetilde{\mathcal{CQ}}(C_1) \cup \widetilde{\mathcal{CQ}}(C_2)$	$\widetilde{\mathcal{CA}}(C_1) \cup \widetilde{\mathcal{CA}}(C_2)$
$\bigvee_{\substack{i+j=n+p \\ i=1..n}} C$	$\widetilde{\mathcal{CQ}}(C) \cup \mathcal{V}ar(p)$	$\left\{ c \;\middle\vert\; \begin{array}{l} - \textit{ soit } c \in \mathcal{V}ar(n) \\ - \textit{ soit } c \in \mathcal{V}ar(p) \\ - \textit{ soit } c \in \widetilde{\mathcal{CA}}(C) \end{array} \right\}$

Les compteurs principaux potentiels *sont définis comme les compteurs qui ne sont pas quelconques.*

. . . . . .

Idée sous-jacente.
Bien que complexes, les définitions de compteur quelconque et de compteur auxiliaire potentiel sont dans la continuation des mêmes définitions pour les termes primaux, vues en chapitre 4 page 19.

Les définitions s'étendent en assimilant n au premier paramètre d'un terme rédex et les variables de C et p aux autres paramètres.

. . . . . .

Exemple 17.
Soient le langage du 1er ordre primal suivant :

	Σ	$\mathcal{P}$
$\hat{f}$:	$\mathsf{compteur} \times \mathsf{t} \longrightarrow \mathsf{t}$	
$\hat{g}$:	$\mathsf{compteur} \times \mathsf{compteur} \times \mathsf{t} \longrightarrow \mathsf{t}$	$P : \mathsf{t}$
a :	$\longrightarrow \mathsf{t}$	$Q : \mathsf{t}$
s :	$\mathsf{t} \longrightarrow \mathsf{t}$	$R : \mathsf{t} \times \mathsf{t}$

Alors :

1. $$P(\hat{f}(n; x)) \vee \neg Q(\hat{g}(n, p; s(a))) \in \mathcal{C}onv(\mathcal{P}, \Sigma)$$

2. $$\left(\bigvee_{i=1}^{n} P(\hat{f}(i; x)) \right) \vee \left(\bigvee_{\substack{i=1..n \\ i+j=n+k}} \neg Q(\hat{g}(i, j; x)) \right) \in \mathcal{C}onv(\mathcal{P}, \Sigma)$$

3. $$\bigvee_{j=1}^{n+1} P(\hat{g}(i, \boxed{j}; a)) \notin \mathcal{C}onv(\mathcal{P}, \Sigma)$$

car j n'est pas compteur principal potentiel de $P(\hat{g}(i,j;a))$, mais :

$$\bigvee_{\underline{i}=1}^{n+1} P(\hat{g}(\underline{i},j;a)) \ \in \mathcal{C}onv(\mathcal{P},\Sigma)$$

4.
$$\bigvee_{\underline{i}=1}^{n} \bigvee_{\substack{j=1..p \\ j+k=p+\boxed{i}}} P(\hat{f}(j,k;a)) \ \notin \mathcal{C}onv(\mathcal{P},\Sigma)$$

car i n'est pas compteur principal potentiel de $\bigvee_{\substack{j=1..p \\ j+k=p+i}} P(\hat{f}(j,i;a))$;

$$\bigvee_{\underline{i}=1}^{n} \bigvee_{j=1}^{\underline{i}} P(\hat{f}(j,\boxed{i};a)) \ \notin \mathcal{C}onv(\mathcal{P},\Sigma)$$

car i n'est pas compteur principal potentiel de $\bigvee_{j=1}^{i} P\left(\hat{f}(j,i;a)\right)$ *(en effet, i est présent dans le corps de la clause itérée)* ; *en revanche :*

$$\bigvee_{\underline{i}=1}^{n} \bigvee_{j=1}^{p} P(\hat{f}(j,p;a)) \ \in \mathcal{C}onv(\mathcal{P},\Sigma)$$

(i n'apparaissant pas, il est bien compteur principal potentiel)

5.
$$\bigvee_{\substack{\underline{i}=1..n \\ i+j=n+k}} \left(P(\hat{f}(\underline{i};x)) \vee \neg Q(\hat{g}(j,\boxed{i};x))\right) \ \notin \mathcal{C}onv(\mathcal{P},\Sigma)$$

car i n'est pas compteur principal potentiel de $Q(\hat{g}(j,i;x))$, et donc ni de $P(\hat{f}(i;x)) \vee \neg Q(\hat{g}(j,i;x))$, mais :

$$\bigvee_{\substack{\underline{i}=1..n \\ i+j=n+k}} \left(P(\hat{f}(\underline{i};x)) \vee \neg Q(\hat{g}(\underline{i},j;x))\right) \ \in \mathcal{C}onv(\mathcal{P},\Sigma)$$

et

$$\bigvee_{\substack{\underline{i}=1..n \\ i+j=n+k}} \left(P(\hat{f}(\underline{i};x)) \vee \neg Q(\hat{g}(k,j;x))\right) \ \in \mathcal{C}onv(\mathcal{P},\Sigma)$$

6.1.2 Sémantique

FORMELLEMENT, LA SÉMANTIQUE des clauses itérées se construit de la même manière que pour les schémas de formules :

On appelle *énumérateur d'une clause itérée* C toute substitution close dont le domaine inclut toutes les variables compteurs libres de C.

Dès lors, on définit naturellement le *langage engendré par une clause itérée* C de la façon suivante :

$$L(C) = \{C\xi\downarrow_{\mathcal{R}} \mid \xi \text{ est un énumérateur de } C\}$$

où $\mathcal{R}$ désigne le système Presburger associé à $\mathcal{LP}$ *(ce dernier ne réécrit que les termes, pas les formules)*, ainsi que la règle :

$$\bigvee_{\substack{i+j=n+p \\ i=1..n}} C \ \longrightarrow \ \underbrace{C\left\{\substack{i\leftarrow 1 \\ j\leftarrow n+p-1}\right\} \vee C\left\{\substack{i\leftarrow 2 \\ j\leftarrow n+p-2}\right\} \vee \cdots \vee C\left\{\substack{i\leftarrow n \\ j\leftarrow p}\right\}}_{n \text{ fois}} \quad \text{lorsque } n \text{ est un entier.}$$

Note : il est clair que le système obtenu reste convergent.

Chaque élément de $L(C)$ est alors une clause on ne peut plus usuelle.

.....

Idée sous-jacente.

Une clause de la forme $\bigvee_{\substack{i+j=n+p \\ i=1..n}} C$ est une clause usuelle *lorsque* n *est un entier*. C'est, dans ce cas, une simple notation pour la clause :

$$\underbrace{C\left\{{}^{i\leftarrow 1}_{j\leftarrow n+p-1}\right\} \vee C\left\{{}^{i\leftarrow 2}_{j\leftarrow n+p-2}\right\} \vee \cdots \vee C\left\{{}^{i\leftarrow n}_{j\leftarrow p}\right\}}_{n \text{ fois}}$$

Les clauses itérées permettent, elles, d'utiliser pour n *une expression avec variables*. Alors, la notation :

$$\underbrace{C\left\{{}^{i\leftarrow 1}_{j\leftarrow n+p-1}\right\} \vee C\left\{{}^{i\leftarrow 2}_{j\leftarrow n+p-2}\right\} \vee \cdots \vee C\left\{{}^{i\leftarrow n}_{j\leftarrow p}\right\}}_{? \text{ fois}}$$

n'est pas une clause usuelle : les « ... » ne peuvent pas être complétés car dépendants de la *valeur* de n.

Le langage engendré par cette clause itérée est l'ensemble des clauses obtenues en donnant une valeur entière à n.

.....

PUIS LA NOTION d'*interprétation* vient de manière similaire aux schémas de formules : étant donné une interprétation $\mathcal{I}$ sur $\mathcal{L}$ (le langage du 1er ordre sur lequel est basé $\mathcal{C}onv(\mathcal{LP})$), on étend cette interprétation en une interprétation sur la clause itérée C, notée $\tilde{\mathcal{I}}(C)$, définie de la manière suivante :

$$\tilde{\mathcal{I}} \not\models C \text{ ssi. } \forall c \in L(C), \mathcal{I} \not\models c$$

Il est clair que

$$\widetilde{\mathcal{I}}_{conv} \models C_1 \vee C_2 \text{ ssi. } \widetilde{\mathcal{I}}_{conv} \models C_1 \text{ ou } \widetilde{\mathcal{I}}_{conv} \models C_2$$

.....

Idée sous-jacente.

Une clause itérée est réfutable si et seulement si toutes ses instances sont réfutables.

.....

6.2 Caractérisation des clauses primales par les clauses itérées

Le but de la suite est de montrer que toute clause primale est équivalente à une clause itérée et réciproquement.

6.2.1 Sémantique de l'équivalence entre une clause primale et une clause itérée

ON COMMENCE PAR définir l'équivalence entre une clause primale et une clause itérée.

DÉFINITION 29 (ÉQUIVALENCE ENTRE CLAUSES ITÉRÉES ET CLAUSES PRIMALES).

Soient une clause itérée C et une clause primale D.

On note :

- $\mathcal{R}_{conv}$ *le système de réécriture de termes associé à C ;*
- $\mathcal{R}_{schéma}$ *le système de réécriture de termes* et clauses *associé à D.*

Alors C et D sont dites équivalentes, *si et seulement si pour tout énumérateur ξ de C et D, on a :*

$$\underbrace{C\xi\downarrow_{\mathcal{R}_{conv}}}_{\textit{clause}} = \underbrace{D\xi\downarrow_{\mathcal{R}_{schéma}}}_{\textit{terme sur une signature clausale}}$$

.....

Idée sous-jacente.

Une clause primale et une clause itérée sont équivalentes si, en instanciant les paramètres de la même manière dans les deux cas, on obtient la même clause usuelle.

.....

On notera $C \rightleftharpoons D$ et $D \leftrightharpoons C$ au lieu de $C \equiv D$ car C et D ne sont pas des objets de même classe ce qui est contraire à la sémantique habituelle de $\equiv$.

6.2.2 Toute clause primale est équivalente à une clause itérée

Lemme 2 (Forme des clauses primales).
Pour toute clause primale C, il existe des littéraux $L_1, \dots, L_i$ et des clauses *rédex* $\hat{C}_1(\vec{p_1}; \vec{t_1}), \dots, \hat{C}_j(\vec{p_j}; \vec{t_j})$, tel que C est équivalent à :

$$\square \vee (L_1)_{\mathsf{clause}} \vee \cdots \vee (L_i)_{\mathsf{clause}} \vee \hat{C}_1(\vec{p_1}; \vec{t_1}) \vee \cdots \vee \hat{C}_j(\vec{p_j}; \vec{t_j}) \quad (i, j \in \mathbb{N})$$

.....

Idée sous-jacente.
L'idée essentielle est qu'une clause primale est une clause classique « généralisée », i.e. c'est une disjonction de littéraux (comme une clause classique) **et d'autres schémas de clauses éventuels**.

.....

Démonstration. Les seuls symboles de types clause sont les clauses définies et :

$$\begin{array}{rr} \square : & \longrightarrow \mathsf{clause} \\ (\cdot)_{\mathsf{clause}} : & \mathsf{littéral} \longrightarrow \mathsf{clause} \\ \vee : & \mathsf{clause} \times \mathsf{clause} \longrightarrow \mathsf{clause} \end{array}$$

On démontre la propriété par induction structurelle sur C.

1. $C = \square$ est bien de la forme voulue en prenant $i = j = 0$;
2. $C = (L)_{\mathsf{clause}}$ alors C *est bien de la forme voulue avec* $i = 1, j = 0$;
3. $C = \hat{C}(\vec{p}; \vec{t})$ alors C *est bien de la forme voulue avec* $i = 0, j = 1$;
4. $C = D^1 \vee D^2$ où D^1 et D^2 sont des clauses primales, alors :
   - $C = (\square \vee L_1^1 \vee \cdots \vee L_i^1 \vee \hat{D}_1^1(\vec{p_1^1}; \vec{t_1^1}) \vee \cdots \vee \hat{D}_j^1(\vec{p_j^1}; \vec{t_j^1})) \vee (\square \vee L_1^2 \vee \cdots \vee L_i^2 \vee \hat{D}_1^2(\vec{p_1^2}; \vec{t_1^2}) \vee \cdots \vee \hat{D}_j^2(\vec{p_j^2}; \vec{t_j^2}))$ *(par hypothèse de récurrence)*
   - en utilisant les *propriétés d'associativité, de commutativité et d'idempotence de la disjonction*, on obtient un terme équivalent ayant la forme voulue.

$\square$

Par la suite on ne notera plus les $(\cdot)_{\mathsf{clause}}$ par souci de lisibilité.
Ce qui donne ici :

$$\square \vee L_1 \vee \cdots \vee L_i \vee \hat{C}_1(\vec{p_1}; \vec{t_1}) \vee \cdots \vee \hat{C}_j(\vec{p_j}; \vec{t_j})$$

Corollaire 2.
Soit une clause définie $\hat{C}$, alors :

$$\begin{aligned} r_1^{\hat{C}} &\equiv \square \vee L_1^1 \vee \cdots \vee L_{i_1}^1 \vee \hat{C}_1^1(\vec{c_1^1}; \vec{x_1^1}) \vee \cdots \vee \hat{C}_{j_1}^1(\vec{c_{j_1}^1}; \vec{x_{j_1}^1}) \\ r_2^{\hat{C}}[\hat{C}(\mathbf{i}, \vec{c}; \vec{x})]_A &\equiv \square \vee L_1^2 \vee \cdots \vee L_{i_2}^2 \vee \hat{C}_1^2(\vec{c_1^2}; \vec{x_1^2}) \vee \cdots \vee \hat{C}_{j_2}^2(\vec{c_{j_2}^2}; \vec{x_{j_2}^2}) \vee \hat{C}(\mathbf{i}, \vec{c}; \vec{x}) \end{aligned}$$

ou (si $\hat{C}$ a un compteur auxiliaire) :

$$\begin{aligned} & r_2^{\hat{C}}[\hat{C}(\mathbf{i}, c_1, \dots, c_{k-1}, \mathbf{succ}(c_k), c_{k+1}, \dots, c_n; \vec{x})]_A \equiv \\ & \qquad \square \vee L_1^2 \vee \cdots \vee L_{i_2}^2 \vee \hat{C}_1^2(\vec{c_1^2}; \vec{x_1^2}) \vee \cdots \vee \hat{C}_{j_2}^2(\vec{c_{j_2}^2}; \vec{x_{j_2}^2}) \vee \hat{C}(\mathbf{i}, c_1, \dots, c_{k-1}, \mathbf{succ}(c_k), c_{k+1}, \dots; \vec{x}) \end{aligned}$$

où $i_1, i_2, j_1, j_2 \in \mathbb{N}$.

Démonstration.
- Pour $r_1^{\hat{C}}$: trivial *(lemme 2)*.
- Pour $r_2^{\hat{C}}[\hat{C}(\mathbf{i}, \dots)]_A$:
  - on a déjà : $r_2^{\hat{C}}[\hat{C}(\mathbf{i}, \dots)]_A \equiv \square \vee L_1 \vee \cdots \vee L_i \vee \hat{C}_1(\vec{c_1}; \vec{x_1}) \vee \cdots \vee \hat{C}_j(\vec{c_j}; \vec{x_j})$ *(lemme 2)* ;
  - $\hat{C}(\mathbf{i}, \dots)$ est donc un sous-terme de ce terme, mais :
    1. $\hat{C}(\mathbf{i}, \dots)$ ne peut pas être un des L_i *(par typage)* ;
    2. $\hat{C}(\mathbf{i}, \dots)$ ne peut pas être $\square$ ni un sous-terme de $\square$ *(qui est un symbole constructeur d'une part, et une constante d'autre part)* ;
    3. $\hat{C}(\mathbf{i}, \dots)$ ne peut pas être un sous-terme d'un L_i ;

       (soit L_i est rédex et c'est impossible car $r_2^{\hat{C}}[\hat{C}(\mathbf{i}, \dots)]_A$ est régulier, soit L_i a pour tête un symbole constructeur qui ne peut être que $\neg$ ou un symbole de prédicat, ce qui est impossible car $\hat{C}(\mathbf{i}, \dots)$ est de type clause*)*

4. $\hat{C}(\mathbf{i},\dots)$ ne peut pas être un sous-terme d'un $\hat{C}_i(\vec{c}_i;\vec{x}_i)$;
   ($r_2^{\hat{C}}[\hat{C}(\mathbf{i},\dots)]_A$ est un terme régulier)

- $\hat{C}(\mathbf{i},\dots)$ est donc l'un des $\hat{C}_i(\vec{c}_i;\vec{x}_i)$,
  ce qui nous donne la forme voulue.

□

Lemme 3.
Soit une clause définie $\hat{C}$ de compteur principal i et de compteur auxiliaire j.
Alors, il existe des clauses primales C_1 et C_2 telles que, pour toute clause rédex de la forme $\hat{C}(n,\vec{p};\vec{x})$ où n ne contient pas de variable :

$$\hat{C}(n,\vec{p};\vec{x}) \;\equiv\; \left(C_1\{_{j\leftarrow n+p_j}\} \vee C_2\{^{i\leftarrow 1}_{j\leftarrow p_j+n-1}\} \vee C_2\{^{i\leftarrow 2}_{j\leftarrow p_j+n-2}\} \vee \cdots \vee C_2\{^{i\leftarrow n}_{j\leftarrow p_j}\}\right)\sigma$$

avec $\sigma = \left\{\begin{smallmatrix} c_1\leftarrow p_1 \\ \cdots \\ c_k\leftarrow p_k \end{smallmatrix}\right\}$ où $\vec{c}$ sont les paramètres de $\hat{C}$ dans le système Presburger.

.....

Idée sous-jacente.
On ne fait qu'exprimer l'intuition, à savoir qu'une clause itérée représente un ensemble de clauses ayant la forme suivante :

$$\underbrace{C\{^{i\leftarrow 1}_{j\leftarrow n+p-1}\} \vee C\{^{i\leftarrow 2}_{j\leftarrow n+p-2}\} \vee \cdots \vee C\{^{i\leftarrow n}_{j\leftarrow p}\}}_{n \text{ fois}}$$

.....

Démonstration.
– du corollaire 2 page précédente on tire :

$$r_1^{\hat{C}} \equiv \overbrace{\square \vee L_1^1 \vee \cdots \vee L_{i_1}^1 \vee \hat{C}_1^1(\vec{c}_1^1;\vec{x}_1^1) \vee \cdots \vee \hat{C}_{j_1}^1(\vec{c}_{j_1}^1;\vec{x}_{j_1}^1)}^{C_1}$$

$$r_2^{\hat{C}}[\hat{C}(\mathbf{i},\dots,\mathbf{succ}(j),\dots;\vec{x})]_A \equiv$$
$$\underbrace{\square \vee L_1^2 \vee \cdots \vee L_{i_2}^2 \vee \hat{C}_1^2(\vec{c}_1^2;\vec{x}_1^2) \vee \cdots \vee \hat{C}_{j_2}^2(\vec{c}_{j_2}^2;\vec{x}_{j_2}^2)}_{C_2'} \vee \hat{C}(\mathbf{i},\dots,\mathbf{succ}(j),\dots;\vec{x})$$

– Par récurrence sur n, on obtient :

$$\hat{C}(n,\vec{p};\vec{x}) \equiv \left(C_1\{_{j\leftarrow n+p_j}\} \vee C_2'\{^{i\leftarrow 0}_{j\leftarrow p_j+n-1}\} \vee C_2'\{^{i\leftarrow 1}_{j\leftarrow p_j+n-2}\} \vee \cdots \vee C_2'\left\{^{i\leftarrow n-1}_{j\leftarrow p_j}\right\}\right)\sigma$$

– Pour avoir exactement la forme voulue, on pose $C_2 = C_2'\{_{i\leftarrow \mathbf{succ}(i)}\}$.

□

Remarque 11.
De plus toute clause rédex contenu dans C_1 ou C_2 appartient à l'approximation de $\hat{C}(i,\vec{c};\vec{x})$ ce qui est évident au regard de la démonstration du corollaire 2 page précédente.

Lemme 4.
Soient une clause définie $\hat{C}$ et les clauses primales C_1, C_2 issues du lemme 3.
S'il existe :
– une clause *itérée* C_1' tel que $C_1' \rightleftharpoons C_1$,
– une clause *itérée* C_2' tel que $\begin{cases} C_2' \rightleftharpoons C_2 \\ \widetilde{\mathcal{CP}}(C_2') = \mathcal{CP}(C_2) \\ \widetilde{\mathcal{CA}}(C_2') = \mathcal{CA}(C_2) \end{cases}$

Alors :

$$\hat{C}(c_0,\vec{p};\vec{x}) \;\rightleftharpoons\; \left(C_1'\{_{j\leftarrow c_0+p_j}\} \vee \bigvee_{\substack{i=1..c_0 \\ i+j=c_0+p_j}} C_2'\right)$$

pour toute clause primale de la forme $\hat{C}(c_0,\vec{p};\vec{x})$.

Idée sous-jacente.

Si l'on suppose connue l'équivalence entre C et C', alors :

$$\begin{pmatrix} \hat{D}(0,j) \to \square \\ \hat{D}(i+1,j) \to C[\hat{D}(i,j+1)]_A \end{pmatrix} \Leftrightarrow \begin{pmatrix} \bigvee_{\substack{i=1..n \\ i+j=n+p}} C' \end{pmatrix}$$

Démonstration. On note $\mathcal{R}_{conv}$ le système de réécriture de termes associé aux clauses primales et $\mathcal{R}_{schéma}$ le système de réécriture de termes et clauses associé aux clauses itérées.

1. On vérifie d'abord que la clause itérée présentée est bien définie, i.e. i (resp. j) est bien un compteur principal (resp. auxiliaire) potentiel de C'_2.
   - comme C_2 a été obtenu dans la démonstration précédente à partir de $r_2^{\hat{C}}$, il a en particulier i pour compteur principal potentiel, par hypothèse i est donc aussi compteur principal potentiel de C'_2 ;
   - de même pour j.
2. Puis on vérifie l'équivalence :
   - soit ξ un énumérateur de $\hat{C}(c_0, \vec{p}; \vec{x})$ et de $C'_1\{^{j \leftarrow c_0+p_j}\} \vee \bigvee_{\substack{i=0..c_0 \\ i+j=c_0+p_j}} C'_2$
   - d'après le lemme 3 page ci-contre :

$$\hat{C}(c_0\xi, p_1, \ldots, p_k; \vec{x}) \equiv C_1\{^{j \leftarrow c_0\xi+p_j}\} \vee C_2\left\{^{\substack{i \leftarrow 1 \\ j \leftarrow p_j+c_0\xi-1}}\right\} \vee \cdots \vee C_2\left\{^{\substack{i \leftarrow c_0\xi \\ j \leftarrow p_j}}\right\}$$

   - et donc :

$$\begin{gathered} C_1\{^{j \leftarrow c_0\xi+p_j}\} \vee C_2\left\{^{\substack{i \leftarrow 1 \\ j \leftarrow p_j+c_0\xi-1}}\right\} \vee \cdots \vee C_2\left\{^{\substack{i \leftarrow c_0\xi \\ j \leftarrow p_j}}\right\} \\ \rightleftharpoons \\ C'_1\{^{j \leftarrow c_0\xi+p_j}\} \vee C'_2\left\{^{\substack{i \leftarrow 1 \\ j \leftarrow p_j+c_0\xi-1}}\right\} \vee \cdots \vee C'_2\left\{^{\substack{i \leftarrow c_0\xi \\ j \leftarrow p_j}}\right\} \end{gathered}$$

     Remarque : si C est une clause itérée et D une clause primale tel que $C \rightleftharpoons D$, alors pour toute substitution σ, on a $C\sigma \rightleftharpoons D\sigma$. En effet pour tout énumérateur ξ de $C\sigma$ (resp. $D\sigma$), $\sigma\xi$ est un énumérateur de C (resp. D).
   - donc (on note $\xi' = \xi \setminus \{^{c_0 \leftarrow \cdots}\}$) :

$$\begin{aligned} & \hat{C}(c_0, p_1, \ldots, p_k; \vec{x})\xi \\ \equiv\ & \left(C_1\{^{j \leftarrow c_0\xi+p_j}\} \vee C_2\left\{^{\substack{i \leftarrow 1 \\ j \leftarrow p_j+c_0\xi-1}}\right\} \vee \cdots \vee C_2\left\{^{\substack{i \leftarrow c_0\xi \\ j \leftarrow p_j}}\right\}\right)\xi' \\ \rightleftharpoons\ & \left(C'_1\{^{j \leftarrow c_0\xi+p_j}\} \vee C'_2\left\{^{\substack{i \leftarrow 1 \\ j \leftarrow p_j+c_0\xi-1}}\right\} \vee \cdots \vee C'_2\left\{^{\substack{i \leftarrow c_0\xi \\ j \leftarrow p_j}}\right\}\right)\xi' \\ \rightleftharpoons\ & \left(C'_1\{^{j \leftarrow c_0\xi+p_j}\} \vee \bigvee_{\substack{i=1..c_0\xi \\ i+j=c_0\xi+p_j}} C'_2\right)\xi' \\ \rightleftharpoons\ & \left(C'_1\{^{j \leftarrow c_0+p_j}\} \vee \bigvee_{\substack{i=1..c_0 \\ i+j=c_0+p_j}} C'_2\right)\xi \end{aligned}$$

□

Propriété 6.

Toute clause primale est équivalente à une clause itérée.

Démonstration.

Idée de la preuve.

On applique le lemme 4 page précédente inductivement sur la clause primale ce qui construit au fur et à mesure la clause itérée équivalente.

Les cas de base sont triviaux car ce sont la clause vide et les littéraux, qui sont, l'un comme l'autre, à la fois clause primale et clause itérée.

Toute clause primale peut se représenter de la façon suivante *(lemme 2 page 37)* :

$$\square \vee L_1 \vee \cdots \vee L_i \vee \hat{C}_1(n_1, \vec{p}_1; \vec{t}_1) \vee \cdots \vee \hat{C}_j(n_p, \vec{p}_j; \vec{t}_j) \quad (i, j \in \mathbb{N})$$

On note même :

$$\square \vee L_1 \vee \cdots \vee L_i \vee \hat{C}_1(c_1, \vec{p}_1; \vec{x}_1)\sigma \vee \cdots \vee \hat{C}_j(c_p, \vec{p}_j; \vec{x}_j)\sigma$$

où $\sigma = \left\{ \begin{smallmatrix} x_1 \leftarrow t_1 & i_1 \leftarrow n_1 \\ \cdots & \\ x_k \leftarrow t_k & i_k \leftarrow n_k \end{smallmatrix} \right\}$ ce qui permet de ne pas se soucier des termes usuels et de n'avoir que des variables compteurs en positions principales.

Au regard de cette expression, on voit qu'il suffit de montrer le résultat pour les clauses rédex uniquement. Dès lors on obtiendra des clauses itérées $D_1, \ldots, D_j$ équivalentes à $\hat{C}_1(c_1, \vec{p}_1; \vec{x}_1)$, $\ldots$, $\hat{C}_j(c_j, \vec{p}_j; \vec{x}_j)$. On pourra ainsi construire la clause

$$\square \vee L_1 \vee \cdots \vee L_i \vee D_1\sigma \vee \cdots \vee D_j\sigma$$

qui sera la clause itérée recherchée.

On démontre donc que toute clause rédex de la forme $\hat{C}(c, \vec{p}; \vec{x})$ est équivalente à une clause itérée. De plus, on prouve que les ensembles de compteurs potentiels principaux et auxiliaires sont les mêmes entre la clause rédex et la clause itérée exhibée, ce qui est nécessaire à l'application du lemme 3 page 38.

Par induction sur la précédence entre symboles définis :

- on applique le lemme 3 page 38 à $r_1^{\hat{C}}$ et $r_2^{\hat{C}}$ ce qui donne les clauses primales C_1 et C_2 ; chaque clause rédex appartenant à C_1 et C_2 est dans l'approximation de $\hat{C}(c, \vec{p}; \vec{x})$, on peut donc leur appliquer l'hypothèse d'induction ;
- on construit C_1' et C_2' en remplaçant dans C_1 et C_2 chacune de ces clauses rédex par leurs équivalentes itérées obtenues par induction et les littéraux L par des termes $(L)_{\mathsf{clause}}$ (s'il n'y a pas de clause rédex on a directement une clause primale) ;
- de plus, comme les clauses rédex et leurs équivalentes itérées ont mêmes ensembles de compteurs potentiels *(par hypothèse d'induction)*, C_2 et C_2' ont aussi les mêmes compteurs potentiels ;
- on peut alors appliquer le lemme 4 page 38 ce qui donne la clause itérée $\left(C_1'\{^{j \leftarrow c+p_j}\} \vee \bigvee_{\substack{i=1..c \\ i+j=c+p_j}} C_2' \right)$ équivalente à $\hat{C}(c, \vec{p}; \vec{x})$, dont c est clairement compteur principal potentiel. Les variables utilisées étant les mêmes des deux côtés, et par définition de compteurs auxiliaires potentiels d'une clause itérée et d'un terme primal, ces derniers sont les mêmes dans les deux clauses.

□

6.2.3 Tout clause itérée est équivalente à une clause primale

Propriété 7.

Toute clause itérée est équivalente à une clause primale.

Démonstration.

.....

Idée de la preuve.

On construit la clause primale correspondante en suivant la structure de la clause itérée.

La démarche est similaire à celle de la preuve de la propriété 6 page précédente.

Notons que la démonstration est courte ; c'est en *grande partie* dû à l'utilisation des grammaires primales *étendues* qui nous permettent de construire la clause primale avec beaucoup de liberté.

.....

On montre le résultat par induction sur la structure des clauses convolutives ; de plus on impose que la clause itérée et la clause primale aient le même ensemble de compteurs potentiels.

1. $\square$ *(vue comme terme)* est déjà une clause primale ; il n'y a pas de compteurs potentiels (ni principaux ni auxiliaires).
2. Pour un littéral L, $(L)_{\mathsf{clause}}$ est aussi une clause primale *(donc trivialement équivalente)* les compteurs potentiels sont trivialement les mêmes des deux côtés.
3. L'ensemble des clauses primales et celui des clauses itérées sont tous deux clos par disjonction ; de plus les ensembles de compteurs potentiels d'une clause primale et d'un clause itérée se comportent de la même manière par rapport à la disjonction *(qui est un simple symbole constructeur pour les clauses primales)*.
4. Soit une clause itérée de la forme $\bigvee_{\substack{i=1..n \\ i+j=n+p}} C$.

On applique l'hypothèse d'induction à C, ce qui nous donne une clause primale équivalente D. De plus, toujours par induction, ces deux clauses ont mêmes compteurs potentiels, ce qui justifie la bonne définition du système Presburger étendu suivant :

$$\hat{\Delta}(\mathbf{0}, \vec{c}, j; \vec{x}) \longrightarrow \square$$
$$\hat{\Delta}(\mathbf{succ}(i), \vec{c}, j; \vec{x}) \longrightarrow D\{_{i \leftarrow \mathbf{succ}(i)}\} \vee \hat{\Delta}(i, \vec{c_1}, \mathbf{succ}(j), \vec{c_2}; \vec{x})$$

où $c_1, \ldots, c_k$ sont les compteurs apparaissant dans D.

La clause primale $\hat{\Delta}(n, \vec{c_1}, p, \vec{c_2}; \vec{x})$ est alors clairement équivalente à la clause itérée originale. De plus elles ont mêmes ensembles de compteurs potentiels *par induction* pour les compteurs de C (et D), et *par définition des différents compteurs potentiels* pour les variables de n et p.

□

Corollaire 3.
Il est équivalent d'utiliser les clauses primales ou les clauses itérées.

6.3 Schémas itérés

DÉFINITION 30 (SCHÉMA ITÉRÉE).
On définit similairement aux clauses itérées les ensembles itérés de clauses, que l'on appellera plus simplement schémas itérés *:*
- *toute clause itérée est un schéma itéré ;*
- *si $\mathcal{S}$ est un schéma itéré, alors :*

$$\bigcup_{\substack{i=1..n \\ i+j=n+p}} \mathcal{S}$$

est un schéma itéré (avec les mêmes contraintes sur i et j que pour les clauses itérées, les définitions de compteurs potentiels étant obtenues en remplaçant dans leurs définitions respectives les $\bigvee$ par des $\bigcup$).

. . . . . .

Idée sous-jacente.
C'est le même principe que pour les clauses itérées : on peut maintenant itérer sur la conjonction (l'union ensembliste) en plus de la disjonction.

. . . . . .

Lorsque l'on souhaitera mettre en valeur le fait que l'on peut itérer une conjonction en plus d'une disjonction, on appellera parfois les schémas itérés des *conjonctions itérées.*

THÉORÈME 2.
Tout schéma de formules est équivalent à un schéma itéré et réciproquement.

Démonstration. Par dualité entre la disjonction et la conjonction. □

6.4 Exemple : le problème des pigeonniers

NOUS PRENONS une fois encore l'exemple des pigeonniers. On peut ainsi, en comparant avec la modélisation précédente (sans schéma itéré), constater que les schémas itérés sont plus proches de l'intuition que les schémas de formules.

Cependant, cet exemple permet aussi de constater qu'il subsiste de (grosses) lourdeurs qui sont somme toute assez contraignantes. Heureusement une de ces lourdeurs (récurrente) peut être traitée de manière systématique, donnant lieu à la notation 4 page 43 (§6.5 page 43).

ON REPREND L'EXPRESSION en §5.4 page 29 :

1. $\boxed{\text{« Chaque pigeon va dans un pigeonnier »}}$

$$\rightsquigarrow$$

$$\forall i \in 1..n+1 : \exists j \in 1..n \; tq. \; p(i,j)$$

$$\rightsquigarrow$$

$$\bigcup_{i=1}^{n+1} \bigvee_{j=1}^{n} p(\widehat{i},\widehat{j})$$

Mais cette dernière écriture *ne correspond pas à un schéma itéré*, en effet i n'est pas compteur principal potentiel de $\bigvee_{j=1}^{n} p(\widehat{\boxed{i}},\widehat{j})$.

$$\rightsquigarrow$$

$$\bigcup_{\substack{i'=1..n+1 \\ i+i'=n+2}} \bigvee_{j=1}^{n} p(\widehat{i},\widehat{j})$$

qui est bien un schéma itéré.

2. $\boxed{\text{« Chaque pigeonnier ne contient qu'un pigeon »}}$

$$\rightsquigarrow$$

$$\forall i \in 1..n, \forall j \in 1..n+1, \forall k \in 1..n+1, k \neq j : \; p(j,i) \Rightarrow \neg p(k,i)$$

$$\rightsquigarrow$$

$$\bigcup_{i=1}^{n} \bigcup_{j=1}^{n+1} \bigcup_{k=1}^{j-1} \neg p(\widehat{j},\widehat{i}) \vee \neg p(\widehat{k},\widehat{i})$$

Mais $j-1$ n'est pas une expression compteur (pas de soustraction possible).

$$\rightsquigarrow$$

$$\bigcup_{i=1..n} \bigcup_{\substack{j'=1..n+1 \\ j'+j=n+1}} \bigcup_{k=1..j} \neg p(\widehat{\mathbf{succ}(j)},\widehat{i}) \vee \neg p(\widehat{k},\widehat{i})$$

$$\rightsquigarrow$$

Comme précédemment, i n'est pas compteur principal potentiel, on effectue donc la même transformation :

$$\rightsquigarrow$$

$$\bigcup_{\substack{i'=1..n \\ i+i'=n+1}} \bigcup_{\substack{j'=1..n+1 \\ j'+j=n+1}} \bigcup_{k=1..j} \neg p(\widehat{\mathbf{succ}(j)},\widehat{i}) \vee \neg p(\widehat{k},\widehat{i})$$

Cette fois, c'est j qui n'est pas compteur auxiliaire potentiel de $\bigcup_{k=1..\underline{j}} \neg p(\widehat{\mathbf{succ}(\underline{j})},\widehat{i}) \vee \neg p(\widehat{k},\widehat{i})$

$$\rightsquigarrow$$

$$\bigcup_{\substack{i'=1..n \\ i+i'=n+1}} \bigcup_{\substack{j'=1..n+1 \\ j'+j=n+1}} \bigcup_{\substack{k=1..j \\ k+k'=j+1}} \neg p(\widehat{k+k'},\widehat{i}) \vee \neg p(\widehat{k},\widehat{i})$$

Au final, le schéma de formules à réfuter est le suivant :

$$\boxed{\begin{array}{l} \bigcup_{\substack{i'=1..n+1 \\ i+i'=n+2}} \bigvee_{j=1}^{n} p(\widehat{i},\widehat{j}) \\ \bigcup_{\substack{i'=1..n \\ i+i'=n+1}} \bigcup_{\substack{j'=1..n+1 \\ j'+j=n+1}} \bigcup_{\substack{k=1..j \\ k+k'=j+1}} \neg p(\widehat{k+k'},\widehat{i}) \vee \neg p(\widehat{k},\widehat{i}) \end{array}}$$

On se référera à profit à l'exemple en §5.4 page 29, pour constater que :

1. le schéma obtenu est plus intuitif ;
2. il contient toute l'information dans sa seule expression (pas besoin de grammaire primale pour connaître sa signification) ;
3. une page a suffit pour l'obtenir...

6.5 Relaxation des contraintes : notation allégée

L'EXEMPLE DES PIGEONNIERS nous montre qu'un problème revient régulièrement : celui où l'on souhaite itérer sur une variable qui *n'est pas compteur principal potentiel* mais qui est *compteur auxiliaire potentiel*. Ce problème se traite de manière systématique, ce qui donne lieu à la notation 4. Généraliser ce traitement pour rendre les schémas encore plus « ergonomiques » fait partie de nos perspectives.

On justifie d'abord cette notation par les quelques propriétés suivantes :

Propriété 8.
Soit une clause C dont i est compteur auxiliaire potentiel et i' un compteur n'apparaissant pas dans C. Alors

$$\bigvee_{\substack{i'=1..n \\ i'+i=n+1}} C$$

est une clause itérée.

Démonstration. Comme i' n'apparait pas, il n'est pas compteur quelconque, il est donc compteur principal potentiel. □

Propriété 9.
Sous les mêmes conditions, lorsque n est un entier, on a :

$$\bigvee_{\substack{i'=1..n \\ i'+i=n+1}} C \;=\; C\{_{i\leftarrow 1}\} \vee C\{_{i\leftarrow 2}\} \vee \cdots \vee C\{_{i\leftarrow n}\}$$

Démonstration. Lorsque n est entier, on peut réécrire $\bigvee_{\substack{i'=1..n \\ i'+i=n+1}} C$ en :

$$C\{_{i\leftarrow n+1-1=n}^{i'\leftarrow 1}\} \vee C\{_{i\leftarrow n+1-2=n-1}^{i'\leftarrow 2}\} \vee \cdots \vee C\{_{i\leftarrow n+1-n=1}^{i'\leftarrow n}\}$$

Et comme i' n'apparait pas dans C, c'est équivalent à :

$$C\{_{i\leftarrow n}\} \vee C\{_{i\leftarrow n-1}\} \vee \cdots \vee C\{_{i\leftarrow 1}\}$$

D'où le résultat par associativité et commutativité de la disjonction. □

Notation 4.
Cette propriété justifie que l'on note :

$$\bigvee_{i=1}^{n} C \stackrel{\text{def}}{=} \bigvee_{\substack{i'=1..n \\ i'+i=n+1}} C$$

même quand i n'est « que » compteur auxiliaire potentiel.

Bien évidemment, on a le même résultat, et donc la même notation, dans le cas où l'on a un schéma itéré au lieu d'une clause itérée.

ON PEUT AINSI simplifier l'exemple des pigeonniers de la façon suivante :

$$\boxed{\begin{array}{l} \displaystyle\bigcup_{i=1}^{n+1} \bigvee_{j=1}^{n} p(\widehat{i},\widehat{j}) \\ \displaystyle\bigcup_{i=1}^{n} \bigcup_{j=1}^{n+1} \bigcup_{\substack{k=1..j \\ k+k'=j+1}} \neg p(\widehat{k+k'},\widehat{i}) \vee \neg p(\widehat{k},\widehat{i}) \end{array}}$$

NOTONS ENFIN que nous espérons à terme pouvoir généraliser ce résultat afin d'obtenir des *schémas itérés étendus* dont le pouvoir d'expression serait équivalent à celui des schémas itérés, mais sans la lourdeur des compteurs principaux/auxiliaires potentiels.

6.6 Exemple : la k-coloriabilité

CHANGEONS d'exemple. Le problème de la k-coloriabilité d'un graphe consiste à savoir s'il existe un coloriage des sommets (i.e. une fonction qui à chaque sommet associe une couleur) tel que tous les sommets reliés par une arête soient de couleurs différentes. Le nombre de couleurs est limité à k qui est un paramètre du problème.

Nous modélisons ce problème avec les schémas itérés.

Formalisation :
- les sommets du graphe sont numérotés de 1 à n ;
- la relation $a(i,j)$ signifie que les sommets i et j sont voisins ;
- les couleurs sont numérotées de 1 à k ;
- la relation $c(i,q)$ signifie que le sommet i a la couleur q.

Une fois encore, nous procédons par raffinements successifs :

$\boxed{\text{« Deux sommets voisins n'ont pas la même couleur »}}$

$\rightsquigarrow$

« Pour tout couple de sommets, s'ils sont voisins, alors ils n'ont pas la même couleur »

$\rightsquigarrow$

« Pour tout couple de sommets (i,j), si i et j sont voisins, si i a pour couleur k_1, alors j a une couleur autre que k_1 »

$\rightsquigarrow$

« Pour tout couple de sommets (i,j), si i et j sont voisins, si i a pour couleur k_1, alors j a une couleur k_2 appartenant à $[1..k_1-1] \cup [k_1+1..k]$ »

$\rightsquigarrow$

$$\forall i,j \in 1..n, \forall k_1 \in 1..k : (a(i,j) \wedge c(i,k_1)) \implies \exists k_2 \in [1..k_1-1] \cup [k_1+1..k] \; tq. \; c(j,k_2)$$

$\rightsquigarrow$

$$\forall i,j \in 1..n, \forall k_1 \in 1..k : \neg a(i,j) \vee \neg c(i,k_1) \vee \left(\bigvee_{k_2=1}^{k_1-1} c(j,k_2)\right) \vee \left(\bigvee_{k_2=k_1+1}^{k} c(j,k_2)\right)$$

$\rightsquigarrow$

(on passe maintenant aux schémas itérés)

$$\bigcup_{i=1}^{n} \bigcup_{j=1}^{n} \bigcup_{k_1=1}^{k} \left(\neg a(\widehat{i},\widehat{j}) \vee \neg c(\widehat{i},\widehat{k_1}) \vee \left(\bigvee_{k_2=1}^{\boxed{k_1-1}} c(\widehat{j},\widehat{k_2}) \right) \vee \left(\bigvee_{k_2=\boxed{k_1+1}}^{k} c(\widehat{j},\widehat{k_2}) \right) \right)$$

$\rightsquigarrow$

(la soustraction n'est pas autorisée)

$$\bigcup_{i=1}^{n}\bigcup_{j=1}^{n}\bigcup_{\substack{k_1=1..k\\k_1+k'_1=k}}\left(\neg a(\widehat{i},\widehat{j})\vee\neg c(\widehat{i},\widehat{k_1})\vee\left(\bigvee_{k_2=1}^{k'_1}c(\widehat{j},\widehat{k_2})\right)\vee\left(\bigvee_{\substack{k'_2=1..\boxed{k-k_1}\\k'_2+k_2=k+1}}c(\widehat{j},\widehat{k_2})\right)\right)$$

$$\rightsquigarrow$$

(l'itération doit commencer à partir de 1*)*

$$\bigcup_{i=1}^{n}\bigcup_{j=1}^{n}\bigcup_{\substack{k_1=1..k\\k_1+k'_1=k}}\left(\neg a(\widehat{i},\widehat{j})\vee\neg c(\widehat{i},\widehat{k_1})\vee\left(\bigvee_{k_2=1}^{k'_1}c(\widehat{j},\widehat{k_2})\right)\vee\left(\bigvee_{\substack{k'_2=1..k'_1\\k'_2+k_2=k+1}}c(\widehat{j},\widehat{k_2})\right)\right)$$

Notons que les itérations sur i et j bénéficient de la notation 4 page 43.

Il reste à exprimer le fait que chaque sommet n'a qu'une couleur ce qui s'obtient en adaptant la formalisation de « Chaque pigeonnier ne contient qu'un pigeon » :

$$\bigcup_{i=1}^{n}\bigcup_{j=1}^{k}\bigcup_{\substack{k_1=1..j\\k_1+k'_1=j+1}}\neg c(\widehat{i},\widehat{k_1+k'_1})\vee\neg c(\widehat{i},\widehat{k_1})$$

En fin de compte, le problème s'exprime de la façon suivante :

$$\boxed{\begin{array}{l}\displaystyle\bigcup_{i=1}^{n}\bigcup_{j=1}^{n}\bigcup_{\substack{k_1=1..k\\k_1+k'_1=k}}\left(\neg a(\widehat{i},\widehat{j})\vee\neg c(\widehat{i},\widehat{k_1})\vee\left(\bigvee_{k_2=1}^{k'_1}c(\widehat{j},\widehat{k_2})\right)\vee\left(\bigvee_{\substack{k'_2=1..k'_1\\k'_2+k_2=k+1}}c(\widehat{j},\widehat{k_2})\right)\right)\\ \displaystyle\bigcup_{i=1}^{n}\bigcup_{j=1}^{k}\bigcup_{\substack{k_1=1..j\\k_1+k'_1=j+1}}\neg c(\widehat{i},\widehat{k_1+k'_1})\vee\neg c(\widehat{i},\widehat{k_1})\end{array}}$$

NOUS CONSTATONS que l'expression n'est pas forcément simple. Rassurons nous en disant que le même problème exprimé avec les schémas de formules serait encore plus complexe... D'autre part, nous remarquons que plusieurs transformations sont *très similaires* à celles rencontrées avec le problème des pigeonniers. Ces problèmes ont été traités de la même manière entre les deux exemples, ils sont fortement sujets à « systèmatisation ». C'est une des perspectives majeures dans le cadre d'une poursuite de ce travail.

Chapitre 7

Schémas de preuves

DANS CE CHAPITRE , nous proposons un premier système de preuve opérant sur des ensembles itérés de clauses. Notre objectif est de permettre l'écriture de preuves formelles (en utilisant des règles d'inférence permettant de déduire de nouvelles clauses itérées à partir de clauses itérées existantes ou même de conjonctions itérées) et de poser les bases nécessaires à leur mécanisation. Nous ne chercherons pas à automatiser entièrement la recherche de preuve. En l'état, ce système de preuve est plutôt destiné à être utilisé de manière interactive (par exemple en liaison avec un assistant de preuve tel que Coq [The01]).

7.1 Exemple des pigeonniers

NOUS DÉMONTRONS « manuellement » le problème des pigeonniers. Cet exemple sert de piste pour introduire un système déductif — les notes dans la marge sont présentes pour faire le lien entre l'exemple et sa généralisation.

$$\bigcup_{i=1}^{n+1} \bigvee_{j=1}^{n} p(\widehat{i}, \widehat{j}) \tag{7.1}$$

$$\bigcup_{i=1}^{n} \bigcup_{j=1}^{n+1} \bigcup_{\substack{k=1..j \\ k+k'=j+1}} \neg p(\widehat{k+k'}, \widehat{i}) \vee \neg p(\widehat{k}, \widehat{i}) \tag{7.2}$$

Démonstration. Nous procédons par récurrence sur n.

Base $n = 0$, donc le schéma (7.1) se réécrit : (α)

$$\bigcup_{i=1}^{n+1} \underbrace{\bigvee_{j=1}^{\boxed{0}} p(\widehat{i}, \widehat{j})}_{=\square}$$

On a trivialement une contradiction.

Pas $n > 0$, le but est de se ramener au problème des pigeonniers avec $n - 1$ pigeons. (α)

1. Considérons le pigeon $n + 1$, alors, *d'après (7.1)* :

$$\bigvee_{j=1}^{n} p(\widehat{n+1}, \widehat{j})$$

Donc il existe un $j' \leq n$ tel que $p(n + 1, j')$ est vrai. On introduit donc le littéral : (γ)

$$p(\widehat{n+1}, \widehat{j'})$$

2. Or, on a aussi :

$$\bigcup_{i=1}^{n} \bigcup_{j=1}^{n+1} \bigcup_{\substack{k=1..j \\ k+k'=j+1}} \neg p(\widehat{k+k'}, \widehat{i}) \vee \neg p(\widehat{k}, \widehat{i})$$

Et donc : (ζ_1)

$$\neg p(\widehat{k+k'}, \widehat{i}) \vee \neg p(\widehat{k}, \widehat{i})$$

pour tous i, j, k, k' vérifiant :

$$\begin{cases} 1 \leq i \leq n \\ 1 \leq j \leq n \\ 1 \leq k \leq j \\ k + k' = j + 1 \\ 1 \leq k' \end{cases}$$

Il existe donc k et k' tel que $k + k' = n + 1$ (tous les couples $(1, n), (2, n-1), \ldots, (n, 1)$).
De plus, comme $j' \leq n$, il existe un i tel que $i = j'$.
On déduit donc :

$$\neg p(\widehat{n+1}, \widehat{j'}) \vee \neg p(\widehat{k}, \widehat{j'}) \forall k \leq n$$

(ϵ) 3. On peut donc appliquer la résolution, pour tout $k \leq n$:

$$\frac{\boxed{p(\widehat{n+1}, \widehat{j'})} \qquad \boxed{\neg p(\widehat{n+1}, \widehat{j'})} \vee \neg p(\widehat{k}, \widehat{j'})}{\neg p(\widehat{k}, \widehat{j'})}$$

4. De même qu'au point 1 page précédente, on a :

$$\bigvee_{j=1}^{n} p(\widehat{k}, \widehat{j})$$

(δ) On peut donc appliquer la résolution :

$$\frac{\neg p(\widehat{k}, \widehat{j'}) \qquad \bigvee_{j=1}^{n} p(\widehat{k}, \widehat{j})}{\bigvee_{j \neq j'} p(\widehat{k}, \widehat{j})}$$

Mais ce n'est pas une clause itérée... Comme dans les exemples précédents, nous procédons par raffinements jusqu'à obtenir une clause itérée. Les causes de la « non itérativité » sont encadrés dans chaque expression intermédiaire.

$$\bigvee_{j \neq j'} p(\widehat{k}, \widehat{j})$$

$$\rightsquigarrow$$

$$\Big(\bigvee_{j=1}^{\boxed{j'-1}} p(\widehat{k}, \widehat{j})\Big) \vee \Big(\bigvee_{j=\boxed{j'+1}}^{n} p(\widehat{k}, \widehat{j})\Big)$$

$$\rightsquigarrow$$

$$\Big(\bigvee_{j=1}^{j_1} p(\widehat{k}, \widehat{j})\Big) \vee \Big(\bigvee_{\substack{i'=1..j_2 \\ i'+j=j_2+i+1}} p(\widehat{k}, \widehat{j})\Big) \text{ où : } \begin{cases} j_1 = i - 1 \\ j_2 = n - i \end{cases}$$

(ζ_2) 5. Comme les déductions précédentes étaient correctes quels que soient $k \leq n$, on introduit le schéma suivant :

$$\bigcup_{k=1..n} \Big(\bigvee_{j=1}^{j_1} p(\widehat{k}, \widehat{j})\Big) \vee \Big(\bigvee_{\substack{i'=1..j_2 \\ i'+j=j_2+i+1}} p(\widehat{k}, \widehat{j})\Big) \tag{7.3}$$

où : $\begin{cases} j_1 = j' - 1 \\ j_2 = n - j' \end{cases}$

6. D'autre part, du schéma (7.2) page 47, on déduit :

$$\bigcup_{i=1}^{n-1} \bigcup_{j=1}^{n} \bigcup_{\substack{k=1..j \\ k+k'=j+1}} \neg p(\widehat{k+k'}, \widehat{i}) \vee \neg p(\widehat{k}, \widehat{i}) \tag{7.4}$$

Il s'agit du schéma (7.2) page 47 dans lequel $n-1$ est substitué à n. Le schéma obtenu est un sous-ensemble du schéma original, c'en est donc trivialement une conséquence logique.

7. On constate enfin que (7.3) page ci-contre et (7.4) nous ramène au problème original pour $n-1$ pigeonniers, ce qui nous permet d'appliquer l'hypothèse de récurrence. (β)

En particulier, (7.3) page ci-contre se ramène à (7.1) page 47 *via.* la substitution suivante :

$$\begin{cases} p(j,k) \leftarrow p(j,k) \text{ si } k < i \\ p(j,k) \leftarrow p(j,k+1) \text{ si } k \geq i \end{cases}$$

□

DE CET EXEMPLE, nous pouvons d'ores et déjà tirer quelques observations sur le système de preuve à mettre en place :

– démontrer par récurrence (voir 7.1) ; ce qui implique de pouvoir :
  1. diviser le processus en plusieurs branches — selon que l'on est dans le pas ou dans l'induction — (α) ;
  2. détecter, dans une branche inductive, que l'on s'est ramené à une instance strictement inférieure du même problème (β).

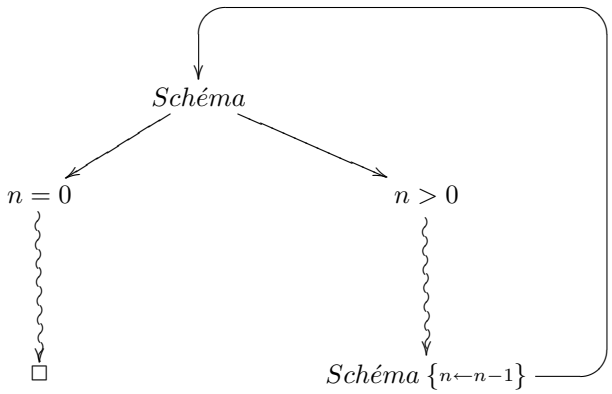

FIG. 7.1 – Gestion de la récurrence.

– « isoler » une clause (γ) ;
– appliquer la résolution à une *clause* itérée (δ) ;
– appliquer la résolution classique (ϵ) ;
– isoler une clause dans un ensemble de clauses (ζ_1), travailler sur cette clause, puis reconstituer l'ensemble des clauses obtenues (ζ_2). Ce qui revient à appliquer le système déductif directement à un ensemble de clauses ((ζ), voir §7.2) ;
– utiliser des contraintes (η).

7.2 Généralisation

NOUS ESQUISSONS à partir de cet exemple un certain nombre de règles fournissant une base pour un système déductif.

Les règles que nous proposons opèrent sur des ensembles de triplets $(S, C, \mathcal{L})$ où :

– (S, C) est une *formule itérée contrainte*, i.e. :
  • S est un ensemble de formules itérées (conjonctions ou disjonctions).

- C est une formule de l'arithmétique de Presburger [1]. C permet d'exprimer des conditions sur les variables entières de S (i.e. les paramètres).

– $\mathcal{L}$ est un ensemble de *lemmes*, i.e. de formules itérées contraintes. Comme nous allons le voir, $\mathcal{L}$ contient les schémas déjà démontrés, pouvant être instanciés au cours de la preuve pour démontrer de nouvelles instances des schémas précédents. L'ensemble E peut-être initialement vide, où bien peut contenir certains schémas dont la preuve est connue (ou admise), par exemple des lemmes proposés par l'utilisateur, des librairies de formules...

7.2.1 Résolution

Unification

L'UNIFICATION PRIMALE est décidable comme nous l'avons vu. Cela est été démontré par Hermann et Galbavý [HG97] qui proposent un algorithme permettant de transformer tout problème d'unification primal en un ensemble (fini, éventuellement vide) de formes résolues, définies comme des couples (σ, C) où σ est une substitution opérant sur les variables non entières du problème et où C est une formule de l'arithmétique de Presburger.

Exemple 18.
Par exemple, étant donné le problème :

$$\hat{f}(n) = g(x)$$

où $\hat{f}$ est défini par les règles :

$$\hat{f}(\mathbf{0}) \to a$$
$$\hat{f}(n+1) \to g(\hat{f}(n))$$

on obtient le couple

$$(\{x \leftarrow \hat{f}(m)\}, n = m + 1)$$

Cet algorithme utilise des règles d'unification classique (décomposition, test d'occurrence, ...), les règles du système de réécriture Presburger définissant les symboles primaux, ainsi qu'un mécanisme de surréduction (*narrowing*) permettant d'instancier les variables entières. Ces règles sont appliquées en mémorisant les étapes intermédiaires, de manière à éviter les boucles. Cet algorithme étant assez complexe, nous renvoyons à [HG97] pour une présentation détaillée. Nous utiliserons cet algorithme comme une simple « boite noire ».

Dans la suite, nous dénotons simplement par $p.g.u.(t, s)$ un unificateur maximal de t et s, obtenu par application de l'algorithme ci-dessus. $p.g.u.(t, s)$ est un couple composé d'une substitution et d'une formule arithmétique. A noter que $p.g.u.$ n'est pas une fonction au sens strict car il peut exister plusieurs unificateurs maximaux.

(ϵ) **Résolution classique**

NOUS REPRENONS tout d'abord la règle de résolution classique, que nous étendons aux clauses primales.

$$\frac{X \vee L \qquad Y \vee L'}{X\sigma \vee Y\sigma} \qquad (\sigma, \textit{true}) = p.g.u.(L, L')$$

où L, L' sont des littéraux.

Remarque 12.
Le seul point inhabituel dans la définition de cette règle est que les contraintes arithmétiques de l'unificateur doivent être vraies (*true*). Cela est dû au fait qu'on cherche à construire un schéma de preuve correct pour toute instantiation des variables entières du problème initial. Ce choix est directement relié à la sémantique que nous avons introduite dans les chapitres précédents : intuitivement, les variables entières sont quantifiées *existentiellement* et non universellement.

Exemple 19.
Considérons par exemple le schéma de formules (exprimé ici sous forme mathématique pour plus de lisibilité) :

$$p(a) \wedge (\forall x)(\neg p(x) \vee p(f(x))) \wedge \neg p(f^n(a))$$

L'application de la résolution entre $p(a)$ et $p(f^n(a))$ est impossible, car n n'est pas nécessairement égal à 0. De même, n n'est pas nécessairement distinct de 0, donc l'application de la résolution entre $p(f(x))$ et $p(f^n(a))$

[1] i.e. une formule construite sur les connectifs logiques usuels, sur l'ensemble infini de fonctions, $+, 0, 1, 2, \ldots$ de $\mathbb{N}$ et sur l'ensemble de prédicats $\{=, <\}$.

est également impossible (on a pas $n > 0$). La seule règle applicable ici sur $p(f^n(a))$ est la règle d'analyse de cas que nous introduirons par la suite.

Résolution sur clauses itérées (δ)

NOUS PROPOSONS maintenant une règle permettant d'appliquer la résolution sur un littéral apparaissant à l'intérieur d'une clause itérée. Le rang de ce littéral est arbitraire, et dénoté par un nouveau paramètre du problème : k.

$$\frac{\bigvee_{i=1}^{n} L \qquad \neg L'\{_{i\leftarrow k}\} \vee C}{\left(\bigvee_{i=1}^{k_1} L\sigma\right) \vee C\sigma \vee \left(\bigvee_{\substack{i'=1..k_2 \\ i'+i=n+1}} L\sigma\right)} \qquad (\sigma, \textit{true}) = p.g.u.(L, L')$$

où :
- L, L' sont des littéraux.
- les contraintes $\begin{cases} k_1 = k - 1 \\ k_2 = n - k \end{cases}$ sont ajoutées aux contraintes de la formule itérée considérée.

Remarque 13.

Notons que le langage des clauses itérées que nous avons introduit dans les chapitres précédents permet d'exprimer cette règle sous une forme très proche de l'écriture mathématique. Exprimer cette même règle en utilisant directement le formalisme des grammaires primales était notre première tentative, mais la définition se révèle beaucoup plus difficile et le résultat est lourd et peu lisible.

Exemple 20.

Soit la clause itérée $\bigvee_{i=1}^{n} p(i, i+1)$, et la clause standard $(\forall x)(p(3, x) \vee q(x))$. Nous instantions la règle ci-dessus avec $k = 3$. Nous obtenons :

$$\bigvee_{i=1}^{2} p(i) \vee q(4) \vee \bigvee_{i'=1}^{n-3} p(n+1-i', n+2-i')$$

La dernière partie de la clause étant (par commutativité) équivalente à $\bigvee_{i=4}^{n} p(i, i+1)$, le résultat obtenu correspond bien à l'application de la résolution sur le troisième littéral de la clause ci-dessus.

Résolution sur conjonctions itérées (ζ)

LA RÈGLE SUIVANTE s'applique sur une conjonction itérée. Il s'agit cette fois-ci d'appliquer la résolution entre une clause D et toutes les clauses d'une conjonction itérée. Contrairement au cas précédent, il n'y a pas besoin d'introduire de nouveau paramètre : toutes les inférences sont appliquées en parallèle sur chacune des clauses.

$$\frac{\bigcup_{\substack{i=1..n \\ i+j=n+p}} C \qquad D}{\bigcup_{\substack{i=1..n \\ i+j=n+p}} E} \qquad E = \mathcal{R}(C, D)$$

où :
- E a i (resp. j) pour compteur principal (resp. auxiliaire).
- $\mathcal{R}$ désigne le système déductif que nous sommes en train de définir et donc $\mathcal{R}(C, D)$ le résultat de l'application de $\mathcal{R}$ à C et D.

Remarque 14.

A noter que l'utilisation du système déductif à l'intérieur de la règle elle même ne pose pas de problème de terminaison, car la taille des itérations que nous considérons décroît à chaque appel de $\mathcal{R}$.

7.2.2 Factorisation

LA RÈGLE DE FACTORISATION, qui vise à regrouper deux littéraux identiques apparaîssant dans une clause, s'étend facilement aux clauses itérées :

Factorisation

$$\frac{C \vee L_1 \vee L_2}{C\sigma \vee L_1\sigma} \qquad (\sigma, \textit{true}) = p.g.u.(L_1, L_2)$$

où L_1, L_2 sont des littéraux.

(γ) **Factorisation généralisée**

NOUS INTRODUISONS également une règle de factorisation généralisée qui s'applique à des clauses itérées. L'idée est la suivante. Une clause de la forme $\bigvee_{i=1}^{n} p(i)$, peut être exprimé par une quantification existentielle : $(\exists j)(p(j) \wedge j \in [1..n])$. Ainsi, il est possible de déduire une nouvelle clause $p(j)$ en ajoutant la contrainte $j \leq n$ à la formule itérée.

$$\frac{\bigvee_{\substack{i=1..n \\ i+j=n+p}} C}{C}$$

avec la contrainte $j \leq n$.

Remarque 15.

Cette règle est importante car elle permet de réduire la taille des preuve en factorisant un certain nombre de cas (ici les cas $j = 1, \ldots, n$).

Elle est à rapprocher de la règle de F-extension introduite par Baaz et Leitsch pour réduire la taille des preuves par résolution. Cette règle permet de factoriser une clause de la forme

$$(\forall x_1, \ldots, x_n)(p(t_1) \vee p(t_2))$$

en :

$$(\forall x_1, \ldots, x_n)(\exists y)p(y)$$

qui peut ensuite par skolémisation être transformée en :

$$(\forall x_1, \ldots, x_n)p(f(x_1, \ldots, x_n))$$

Baaz, Leistch et Egly [BL92, Egl93] ont étudié diverses variantes de la règle de F-extension et ont montré que cette règle peut réduire la taille des preuves d'un facteur non élémentaire.

Notre règle peut être vue comme une généralisation de la F-extension au cas où la taille des clauses n'est pas fixée (i.e. dépend d'une variable entière).

7.2.3 Récurrence

(α) **Analyse de cas**

LA RÈGLE SUIVANTE est très différente des précédentes, car elle permet de faire une analyse de cas en fonction de la valeur donnée à une variable entière n. On considère les cas $n = 0$ et $n = m + 1$, ce qui donne deux formules itérées avec contraintes. Puisqu'on cherche à réfuter le schéma initial pour toute valeur de n, ces deux formules doivent être réfutées. Cependant, par application du principe d'induction sur n, la preuve du schéma correspondant au cas $n = m + 1$ peut utiliser des instances de ce schéma pour des valeurs de n plus petites ou égales à m. Nous ajoutons donc ce schéma dans l'ensemble de lemmes associé à la formule itérée à démontrer.

$$S \text{ dont } n \text{ est un paramètre.}$$

$$\frac{\swarrow}{S\{_{n \leftarrow 0}\}} \qquad\qquad \frac{\searrow}{S\{_{n \leftarrow n+1}\}}$$

$\swarrow$, $\searrow$ signifiant que *les deux schémas introduits sont à réfuter.*

Qui plus est, pour permettre la détection ultérieure du rebouclage, on mémorise le lemme (S', C') où :
- m est une nouvelle constante ;
- S' est un renommage de S dans lequel m est substitué à n ;
- C' est un renommage de C, l'ensemble de contraintes associé à S auquel on ajoute la contrainte $m \leq n$.

Exemple 21.

Appliquons la règle sur la formule itérée

$$p(a) \wedge (\forall x)(\neg p(x) \vee p(f(x))) \wedge \neg p(f^n(a))$$

considérée précédemment. Nous obtenons deux problèmes :
- Le premier correspond au cas $n = 0$, et s'écrit :

$$p(a) \wedge (\forall x)(\neg p(x) \vee p(f(x))) \wedge \neg p(a)$$

La preuve est obtenue par application de la règle de résolution standard sur $p(a)$ et $\neg p(a)$.

– Le second correspond au cas $n = n + 1$. Nous obtenons :

$$p(a) \wedge (\forall x)(\neg p(x) \vee p(f(x))) \wedge \neg p(f(f^n(a)))$$

De plus, le lemme

$$p'(a') \wedge (\forall x)(\neg p'(x) \vee p'(f'(x))) \wedge \neg p'(f'^m(a'))$$

est mémorisé, avec la contrainte $m \leq n$. Cela indique que toutes les instances de ce lemme pour $m \leq n$ peuvent être supposées démontrables. A noter que les p', f', a' dénotent des fonctions et prédicats arbitraires.

Réduction (β)

CETTE RÈGLE vise à réduire une formule itérée à un lemme déjà connu (en particulier pour utiliser une hypothèse d'induction) en choisissant de manière adéquate les symboles de fonctions et de prédicats de ce lemme.

$$\frac{(S, C) \cong (S', C')}{\square} \qquad (S', C') \text{ a été mémorisé}$$

Il reste à définir la relation $\cong$. Nous proposons la définition suivante.

$$(S, C) \cong (S', C')$$

si :

– il existe un système de réécriture R tel que $S' \rightarrow_R S'' \subseteq S$, où les règles sont de la forme :

$$q(x_1, \ldots, x_k) \rightarrow t \quad \text{si } \chi$$

où :
- q est un symbole de S ;
- χ est un ensemble de contraintes sur $x_1, \ldots, x_k$ (pouvant contenir des variables de S) ;
- t est un terme sur la signature de S.

– et $C' \downarrow_R$ est une conséquence de C

NOTRE OBJECTIF n'étant dans ce travail que de donner des règles permettant la formalisation, nous supposerons que le système de réécriture est donné par l'utilisateur. Un des objectifs, à terme serait d'automatiser totalement cette tâche.

On pourrait par exemple utiliser un algorithme d'ordre supérieur pour instancier les symboles de fonction et de prédicat de S' par des fonctions construites sur la signature de S. Cette possibilité donnerait lieu à des développements théoriques difficiles (comment combiner unification primale et unification du second ordre, peut-on garantir la terminaison pour certaines classes ?...) qui ne pouvaient naturellement pas être abordés dans le cadre de ce stage.

Exemple 22.

Illustrons l'application de cette règle sur l'exemple précédent. Nous appliquons la règle de résolution standard pour obtenir la clause $p(f^n(a))$. Ensuite, nous appliquons la règle de réduction avec le système de réécriture trivial

$$R = \{p'(x) \rightarrow p(x), a' \rightarrow a, f'(x) \rightarrow f(x), m \rightarrow n\}$$

Le lecteur peut vérifier que les règles de R transforment la formule itérée

$$p'(a') \wedge (\forall x)(\neg p'(x) \vee p'(f'(x))) \wedge \neg p'(f'^m(a'))$$

en

$$p(a) \wedge (\forall x)(\neg p(x) \vee p(f(x))) \wedge \neg p(f^n(a))$$

qui est contenue dans la formule à démontrer.

Par conséquent, la règle de réduction s'applique, et la clause vide est générée (ce qui termine la preuve du schéma de formules originel).

Nous admettrons la correction des règles ci-dessus.

Chapitre 8

Considérations diverses, perspectives

L'APPORT DE NOTRE TRAVAIL est double. Tout d'abord, nous avons proposé un langage de schéma, associé à une sémantique rigoureuse. Notre formalisme combine le pouvoir d'expression des grammaires primales avec une simplicité d'utilisation proche du langage mathématique. Dans un second temps, nous avons proposé un premier système d'inférence permettant de dériver des schémas de preuve.

NOUS POUVONS regrouper les perspectives selon trois axes :

Système déductif *(schémas de preuves)*

Il reste beaucoup de travail au niveau du système déductif :
- preuve de la correction ;
- étude des propriétés du système d'inférence, en particulier :
  - identification de classes de schémas pour lesquels il est complet pour la réfutation ;
  - caractérisation des problèmes que l'on peut résoudre avc notre approche et qui ne peuvent l'être (pour des raisons de complexité) avec les démonstrateurs existants ;
- automatisation de la règle de *Réduction* ;
- étude d'éventuelles stratégies pour orienter la recherche de preuve ;
- etc...

L'objectif annoncé de cet axe est de parvenir à une **implémentation** (efficace) des schémas de preuves. En effet, il faut insister sur le fait que bien que de nature essentiellement théorique, ce travail a aussi un **but pratique évident : l'amélioration qualitative des démonstrateurs existants**.

Langage *(schémas de formules)*

Les schémas itérés sont beaucoup plus intutifs que les schémas primaux, cependant il faut bien reconnaître que ce formalisme n'est pas une panacée : les notions de compteurs principaux potentiels et auxiliaires potentiels ne sont pas intuitives et très contraignantes.

À ce titre, le plan de travail suivant permettrait de rapprocher encore les schémas de formules de l'intuition (le but est de se rapprocher le plus possible de la langue naturelle...) :

1. généraliser la notation du §6.5 page 43 afin d'obtenir un langage similaire dans la forme aux schémas itérés, mais moins contraignant (plus de compteur principal potentiel), et équivalent en pouvoir d'expression ;
2. exprimer ensuite le langage obtenu en termes de quantificateurs existentiels (resp. universels) au lieu de clauses (resp. conjonctions) itérées. Par exemple :

$$\bigvee_{i=1}^{n} \ldots \quad \text{deviendrait} \quad \exists i \leq n, \ldots$$

et :

$$\bigcup_{i=1}^{n} \ldots \quad \text{deviendrait} \quad \forall i \leq n, \ldots$$

L'objectif de cet axe est d'obtenir une implémentation **ergonomique**, plus facile à utiliser et plus intuitive. Ce travail est nécessaire si l'on souhaite que *les avancées théoriques puissent réellement être bénéfiques sur le plan pratique*.

Notons que cette étude permettrait d'obtenir des connexions avec le domaine très étudié de l'arithmétique bornée (*bounded arithmetic*). Ce champ d'études va de pair avec de nombreuses études en *complexité de la preuve*. Ceci permettrait de mesurer quantitativement l'apport de notre approche par rapport aux techniques classiques.

Divers, prise de recul L'étude présentée dans ce mémoire peut mener à diverses pistes, qui s'inscrivent toutes dans une volonté de rapprocher les preuves formelles de l'humain :

- L'objectif principal est l'*implémentation* (éventuellement dans le cadre d'un assistant de preuve comme Coq) et surtout expérimentation (afin de mieux cerner les possibilités et les limites éventuelles du langage et du système de preuve).
- Il serait probablement très bénéfique de traiter d'autres calculs que la résolution.
  Notons par exemple que la déduction naturelle étant utilisé dans des outils industriels (methode B en particulier), traiter de tels calculs faciliterait une intégration concrète dans le monde industriel.
- Une autre approche intéressante serait la schématisation *directe* des preuves. Ici, nous nous sommes principalement focalisés sur la notion de schéma de formules, et nous avons appelé schémas de preuves les réfutations obtenues par le système déductif.
  Mais on peut aussi envisager de représenter directement des dérivations *via.* des grammaires primales. Par exemple [1] :

$$\hat{f}(\mathbf{0}) \to A(\widehat{\mathbf{0}})$$

$$\hat{f}(\mathbf{succ}(i)) \to \left(\frac{\hat{f}(i) \quad A(\widehat{i}) \implies A(\widehat{\mathbf{succ}(i)})}{A(\mathbf{succ}(i))} \right)$$

  représenterait l'arbre de dérivation :

$$\dfrac{\dfrac{\dfrac{A(0) \quad A(0) \implies A(1)}{A(1)} \quad A(1) \implies A(2)}{A(2)} \quad A(2) \implies A(3)}{\dots}$$

  qui schématise une application itérée de modus ponens sur un ensemble $\{A_0, A_0 \implies A_1, A_1 \implies A_2, \dots\}$.
  C'est un point essentiel si l'on s'intéresse à la représentation et à la manipulation des preuves (par exemple si on veut obtenir la preuve pour un n donné). Un premier pas dans cette voie serait de s'intéresser à la possibilité de reconstruire, à partir d'un schéma de preuves, la preuve classique correspondant à une instanciation donnée de n.
  Cette approche permettrait peut-être aussi d'établir des connexions avec les travaux sur la divergence des procédures de preuve ([Sal91b, Pel97, Pel01]).

[1] Remarquons que ceci est une grammaire primale étendue mais pas une grammaire primale...

Bibliographie

[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.

[BHL+04] Matthias BAAZ, Stefan HETZL, Alexander LEITSCH, Clemens RICHTER et Hendrik SPOHR : Cut-elimination : Experiments with ceres. *In* Franz BAADER et Andrei VORONKOV, éditeurs : *LPAR*, volume 3452 de *Lecture Notes in Computer Science*, pages 481–495. Springer, 2004.

[BKR92] Adel BOUHOULA, Emmanuel KOUNALIS et Michaël 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.

[BL92] M. BAAZ et A. LEITSCH : Complexity of resolution proofs and function introduction. *Annals of Pure and Applied Logic*, 20:181–215, 1992.

[BL06] Matthias BAAZ et Alexander LEITSCH : Towards a clausal analysis of cut-elimination. *J. Symb. Comput.*, 41(3-4):381–410, 2006.

[Boy] Thierry BOY DE LA TOUR : Logique et formalisation du raisonnement, partie i - logique classique, notes de cours. Laboratoire LEIBNIZ-IMAG.

[Caf07] Ricardo CAFERRA : Une approche simple de la logique pour l'informatique et l'automatisation de l'inférence, notes de cours. ENSIMAG, janvier 2007.

[CCC94] Hubert COMON, Serenella CERRITO et Claude CRÉPEAU : Calculabilité et complexité, notes de cours, 1993-1994.

[CDG+05] H. COMON, M. DAUCHET, R. GILLERON, F. JACQUEMARD, D. LUGIEZ, S. TISON et M. TOMMASI : Tree automata techniques and applications. Available on : `http ://www.grappa.univ-lille3.fr/tata`, septembre 2005. release October, 1rst 2002.

[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.

[CJ03] Hubert COMON et Jean-Pierre JOUANNAUD : Les termes en logique et en programmation, version préliminaire, notes de cours. Laboratoire de Recherche en Informatique, CNRS et Université de Paris Sud, décembre 2003.

[Com95] Hubert COMON : On unification of terms with integer exponents. *Mathematical System Theory*, 28:67–88, 1995.

[Com01] Hubert COMON : Inductionless induction. *In Handbook of Automated Reasoning*. North-Holland, 2001.

[Cor06] John CORCORAN : Schemata : The concept of schema in the history of logic. *The Bulletin of Symbolic Logic*, 12, juin 2006.

[Dow04] Gilles DOWEK : Théorie des types, noes de cours. INRIA-Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, septembre 2004.

[Egl93] U. EGLY : On different concepts of function introduction. *In Computational Logic and Proof Theory, KGC 93*, pages 172–183. Springer, LNCS 713, 1993.

[Her94] Miki HERMANN : *Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes*. Habilitation, Université Henri Poincaré Nancy 1, mars 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.

[KB70] D. KNUTH et P. BENDIX : Simple word problems in universal algebra. *In* John LEECH, éditeur : *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[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.

[Pel01] 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.

[Pel03] Nicolas PELTIER : Logique et formalisation du raisonnement, partie ii - introduction à la déduction automatique, notes de cours. CNRS - Laboratoire LEIBNIZ-IMAG, septembre 2003.

[Pel04] N. PELTIER : The First Order Theory of Primal Grammars is Decidable. *Theoretical Computer Science*, 323:267–320, 2004.

[Sal91a] G. SALZER : *Unification of Meta-Terms*. Thèse de doctorat, Technische Universität Wien., 1991.

[Sal91b] Gernot SALZER : Deductive generalization and meta-reasoning or how to formalize genesis. *In* Hermann KAINDL, éditeur : *ÖGAI*, volume 287 de *Informatik-Fachberichte*, pages 103–115. Springer, 1991.

[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.

[The01] THE COQ DEVELOPMENT TEAM : *The Coq Proof Assistant Reference Manual V7.1*. INRIA-Rocquencourt, CNRS-ENS Lyon (France), October 2001. `http ://coq.inria.fr/doc/main.html`.

Index

approximation, 12
arité, 5
arité (d'un prédicat), 7
arité de compteur, 10
atome, 7

clause (usuelle), 8
clause définie, 27
clause itérée, 34
clause primale, 27
codomaine, 5
compteur auxiliaire, 13
compteur auxiliaire potentiel (d'un terme primal), 19
compteur auxiliaire potentiel (d'une clause itérée), 34
compteur principal, 13
compteur principal potentiel (d'un terme primal), 19
compteur principal potentiel (d'une clause itérée), 34
compteur quelconque (d'un terme primal), 19
compteur quelconque (d'une clause itérée), 34
confluence, 15
conjonction définie, 27
conjonction itérée, 41
conjonction primale, 27
constantes, 5
convergence, 15

domaine, 5

énumérateur, 15
énumérateur (d'une clause itérée), 35
équivalence (de grammaires primales), 16
équivalence (entre clauses itérées et primales), 36
expression compteur, 11

F-algèbre, 7
fonction, 5
formule du premier ordre, 7

grammaire primale, 15
grammaire primale étendue, 20
greffe, 6

interprétation (d'un schéma de formules), 28
interprétation (d'une clause itérée), 36

k-coloriabilité, 44

langage du premier ordre primal, 33
langage engendré (par un terme), 16
langage engendré (par une grammaire), 15
langage engendré (par une clause itérée), 35
littéral, 8

nœthérien, 15

notation allégée, 43

position, 6
positions parallèles, 6
prédicat, 7
problème des pigeonniers, 1
profil, 5
profil (d'un prédicat), 7

racine, 6

Σ-algèbre clausale, 26
schéma de clauses, 27
schéma de formules, 27
schéma itéré, 41
signature, 5
signature clausale, 25
signature de schéma clausal, 27
signature primale, 10
sorte, 5
sous-terme, 6
substitution, 6
symbole constructeur, 10
symbole défini, 10
système Presburger, 13
système Presburger étendu, 20

terme, 6
terme primal, 11
terme rédex, 11
terme régulier, 11
terme usuel, 11
terminalité, 15
type, 5

variable, 6
variable compteur, 11
variable usuelle, 11
variables libres, 7