Schémas de formules et de preuves en logique propositionnelle

Vincent Aravantinos

Université de Grenoble

Thèse

Pour obtenir le grade de

Docteur de l'Université de Grenoble
Spécialité : Informatique

Arrêté ministériel : 7 août 2006

Présentée et soutenue publiquement par

Vincent Aravantinos

le 23 septembre 2010

Formulas and proofs schemas in propositional logic

Thèse dirigée par Nicolas Peltier et codirigée par Ricardo Caferra

Jury

Maria Paola	Bonacina	Professeure à l'Université de Vérone	Rapporteure
Dominique	Duval	Professeure à l'Université Joseph Fourier	Examinatrice
Miki	Hermann	Chargé de recherche au CNRS	Rapporteur
Michel	Parigot	Chargé de recherche au CNRS	Examinateur
Nicolas	Peltier	Chargé de recherche au CNRS	Directeur
Brigitte	Plateau	Professeure à l'Ensimag	Présidente

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

Laboratoire d'Informatique de Grenoble, UMR 5217
Équipe CAPP
Bâtiment IMAG C - 220, rue de la Chimie
38400 Saint Martin d'Hères
FRANCE

Abstract

This thesis lies in the field of automated deduction, i.e. the development of algorithms aiming at proving automatically some mathematical conjectures. In this thesis, the conjectures that we want to prove belong to an extension of propositional logic called "formula schemas". Those objects allow to represent infinitely many propositional formulae in a finite way (similarly to the way that regular languages finitely represent infinitely many words). Proving a formula schema amounts to prove (at once) all the formulae that it represents.

We show that the problem of proving formula schemas is undecidable in general. The remaining part of the thesis presents algorithms that still try to prove schemas (event though, of course, they do not terminate in general). Those algorithms allow to identify decidable classes of schemas, i.e. classes for which there exists an algorithm that always terminates for any entry by answering if the schema is valid or not. One of those algorithms have been implemented.

Proof methods that are presented here mix classical procedures for propositional logic (Dpll or semantic tableaux) and inductive reasoning. Inductive reasoning is achieved by the use of "cyclic proofs", i.e. infinite proofs in which cycles are automatically detected. In such a case, we can turn those infinite proofs into finite objects, which we can call "proofs schemas".

Keywords : automated deduction, schematisations, cyclic proofs, fixed point logics, extension of propositional logic

Thanks

First of all, I thank my advisors Niko and Ricardo: they knew both how to help me and how to give me all the freedom I needed. Thanks to Ricardo whose courses at Ensimag made me passionate for logic, and thanks for his permanent encouragements. I still remember when I was a Master student that what convinced me to work with him was the sureness that he would not be my "chief", as many of my friends call their advisors, but a human and a friend. Niko has been the perfect PhD advisor: present, efficient, involved, friendly. And I can compare with the advisors of my friends... He knew how to bring me back on earth when I was lost in some theoretical dreams and I like to believe that he transmitted me the will to always do better, even when we already reached our objectives.

I am honoured that Miki Hermann and Maria Paola Bonacina accepted to review my thesis, I hope my work is good enough w.r.t. this effort. I thank Michel Parigot to be in my jury, I admire his integrity, even though (*even more*) it can generate administrative obstacles. I am very happy that Dominique be in my jury, we saw each other regularly during those three years and it has always been a pleasure to talk with her. Finally, as Brigitte Plateau was my teacher at Ensimag, I am particularly glad that she presides this jury.

I thank the CAPP team, particularly Thierry, Rachid, Pablo, Mnacho and Renan. And more generally the LIG: Rémi Tournaire, Marc Tchiboukdjian, Marin Bougeret, David Rouquet, Lydie du Bousquet and Béatrice Buccio. Totally haphazardly, the non-scientific thanks: Anne-Lucile, Mom, Dad, Matthieu, David, Damien, Luc, Rémi, Paul-Armand, Julien, Nicolas, ... And finally those that I do not thank, but that I thank not to thank (this sentence is clear to me): Max, Quentin, Florian.

Note. This thesis has been partly funded by the ANR project ASAP ("About Schemata And Proofs") involving both the CAPP team of the LIG and the team Theoretische Informatik und Logik (Theory of Informatic and Logic) of Alex Leitsch in Vienna (ANR-09-BLAN-0407-01) (`http://membres-lig.imag.fr/peltier/ASAP`).

The writing of this thesis was done with Melt (`http://melt.forge.ocamlcore.org/`), MLpost (`http://mlpost.lri.fr/`), Ocaml (`http://caml.inria.fr/`) and LaTeX.

Be pleased to only approach genius if it might get you away from Humanism! – D. A.

Tinos, your pedagogy only equals you lack of good behaviour. – D. C.

To me, maths are the same as kick-punch boxing. – M. A.

Contents

List of Figures

CHAPTER 1

Introduction

The word "schema" generally has two meanings: either it is a *graphic* schema, or it is a more abstract schema whose sense is quite well understood in the expression "follow a schema"[1]. This second meaning is the one we target in the following.

1.1 The notion of schema in logic

1.1.1 An object with "holes"

The notion of schema is ubiquitous in logic. The most classical examples are axiom schemas and inference rules, for instance:

- Modus ponens :
$$\frac{A \qquad A \Rightarrow B}{B}$$
where A and B are *meta-variables* standing for any formulas.

- In Hilbert-style proof systems, the modus ponens comes with a certain number of axiom schemas, e.g.:
$$A \Rightarrow (B \Rightarrow A)$$
$$(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B)) \Rightarrow (A \Rightarrow C)$$
$$(\neg B \Rightarrow \neg A) \Rightarrow ((\neg B \Rightarrow A) \Rightarrow B)$$
where, once more, A, B and C stand for any formulas.

- Induction schema in Peano arithmetic:
$$\forall x_1, \ldots, x_n (A(0, x_1, \ldots, x_n) \wedge \forall x (A(x, x_1, \ldots, x_n) \Rightarrow A(s(x), x_1, \ldots, x_n))) \Rightarrow \forall x (A(x, x_1, \ldots, x_n))$$
where A stands for any first-order formula whose free variables are $x, x_1, \ldots, x_n$ and $A(t, t_1, \ldots, t_n)$ denotes A where the variables $x, x_1, \ldots, x_n$ are replaced by $t, t_1, \ldots, t_n$.

In [Cor06], one can find a historic study of the notion of "schema" in logic. From this study, we can conclude that a schema can be defined in an abstract way as a triple made with:

[1] Obviously, we can find more or less obvious links between those two meanings; this is not the subject of this thesis.

- a *language*,
- a *pattern*, i.e. an element of the language with "holes",
- a *condition* to satisfy in order to fill in the holes.

An *instance* of a schema is an element of the language obtained by filling the holes of a pattern in a way that satisfies the condition.

For instance, for the axiom schema $A \Rightarrow (B \Rightarrow A)$, the language is the set of propositional formulas, the pattern is - $\Rightarrow$ (- $\Rightarrow$ -) where "-" denotes a hole and the condition is that the first and third holes be filled in with the same formula. The formula $(X \wedge Y) \Rightarrow (Z \Rightarrow (X \wedge Y))$ is an instance of the schema.

The first explicit use of the term "schema" seems to date back to Tarski [Cor08], but the notion itself can be considered as old as Aristotle with his *syllogistic figures*, e.g. the "Barbara" mode : *every M is P, on the other hand every S is M, thus every S is P.*

An important point is that a schema is no more than a mere syntactic object, without any *a priori* semantics: a schema is not *true* nor *false*, even though every of its instances might be (if the semantics associated with the language allow it). It is an important difference with *quantification*. Indeed, we could be tempted to consider that meta-variables are just variables and the schema as a formula with an implicit (universal) quantification on those variables: in the case of the axiom schemas of propositional logic, we obtain this way *quantified boolean formulas* [Wikb]; in the case of the schema of induction, we obtain the full inductive axiom, in higher order logic. But this would not have the same meaning as quantifiers do have semantics: the bound variable can take any value of the domain, whereas, for schemas, variables cannot be replaced with elements of the (generally recursively enumerable) language. Note, by the way, the difference of expressive power between the *schema* of induction in Peano arithmetic and the *axiom* of induction in higher order logic [BdlT07, Sha91]. However, even though their power of expression is lower, schemas offer more possibilities to control the set of instances. Indeed we can express more general conditions (thanks to the meta-language). With quantifications, the only expressible conditions are equalities or formulas of the language.

Actually a schema is just a *simple* mean to represent in a finite way an infinite set of objects.

1.1.2 Iterated schemas

The schemas that we use in this thesis are a bit more complex than "objects with holes": they are *iterated schemas*. Those schemas are less used in logic than the aforementioned ones. However, they are frequently used in common mathematical reasoning. We give as an example the following assertion:

$$\forall n \geq 1, \forall x_1, \ldots, x_n \in \mathbb{R}^+ : \left(\sum_{i=1}^{n} x_i^2\right) \leq \left(\sum_{i=1}^{n} x_i\right)^2$$

where $\mathbb{R}^+$ stands for the set of positive real numbers. This is *not* a first-order formula, due to the "...". This expression actually represents infinitely many first-order formulas:

$$\forall x_1 \in \mathbb{R}^+ : x_1^2 \leq x_1^2$$
$$\forall x_1, x_2 \in \mathbb{R}^+ : x_1^2 + x_2^2 \leq (x_1 + x_2)^2$$
$$\forall x_1, x_2, x_3 \in \mathbb{R}^+ : x_1^2 + x_2^2 + x_3^2 \leq (x_1 + x_2 + x_3)^2$$
$$\ldots$$

This way, this object allows to represent in a finite way an infinite set of formulas, simply by giving a value to n. It satisfies the aforementioned notion of schema but in a slightly more complex way: n is the "hole" and $n \geq 1$ is the filling condition. Thus, when we give a value to the hole, we obtain a first-order formula.

Remark however that there is an important difference with the previous schemas: here, it is not sufficient to replace n by an integer in order to obtain an object of the language; we first have to make a *computation* in order to obtain a normal form which indeed is an element of the language.

Sentences of the form $\forall x_1, \ldots, x_n.\phi$ (where "..." is part of the language) are *very frequent* in common mathematics (one can be convinced of this simply by going through this thesis). We give other examples of iterated schemas, more oriented towards logic:

- Iterated schemas of propositional formulas: the pigeonhole principle.

  Iterated schemas of propositional formulas are the main object of study of this thesis.

  We give as an example of such schemas the formalisation of the pigeonhole principle in propositional logic:

$$\bigwedge_{i=1}^{n+1}\left(\bigvee_{j=1}^{n} P_{i,j}\right)$$
$$\wedge$$
$$\bigwedge_{j=1}^{n}\bigwedge_{i_1=1}^{n+1}\left(P_{i_1,j} \Rightarrow \bigwedge_{\substack{i_2=1\\ i_2\neq i_1}}^{n+1} \neg P_{i_2,j}\right)$$

  where $n \geq 0$ is the number of pigeonholes, $n+1$ the number of pigeons, and $P_{i,j}$ means that the i^{th} pigeon goes into the j^{th} pigeonhole. The first formula expresses the fact that each pigeon goes into a pigeonhole, the second one the fact that each pigeonhole contains only one pigeon.

  This object is not a propositional formula, but when we give a value to n, we obtain a propositional formula. For instance for $n = 2$:

$$\begin{array}{c}(P_{1,1} \vee P_{1,2}) \wedge (P_{2,1} \vee P_{2,2})\\ \wedge\\ (P_{1,1} \Rightarrow \neg P_{2,1} \wedge \neg P_{3,1}) \wedge (P_{2,1} \Rightarrow \neg P_{1,1} \wedge \neg P_{3,1})\\ \wedge(P_{3,1} \Rightarrow \neg P_{1,1} \wedge \neg P_{2,1}) \wedge (P_{1,2} \Rightarrow \neg P_{2,2} \wedge \neg P_{3,2})\\ \wedge(P_{2,2} \Rightarrow \neg P_{1,2} \wedge \neg P_{3,2}) \wedge (P_{3,2} \Rightarrow \neg P_{1,2} \wedge \neg P_{2,2})\end{array}$$

  or $n = 3$:

$$\begin{array}{c}(P_{1,1} \vee P_{1,2} \vee P_{1,3}) \wedge (P_{2,1} \vee P_{2,2} \vee P_{2,3}) \wedge (P_{3,1} \vee P_{3,2} \vee P_{3,3})\\ \wedge\\ (P_{1,1} \Rightarrow \neg P_{2,1} \wedge \neg P_{3,1} \wedge \neg P_{4,1}) \wedge (P_{2,1} \Rightarrow \neg P_{1,1} \wedge \neg P_{3,1} \wedge \neg P_{4,1})\\ \wedge(P_{3,1} \Rightarrow \neg P_{1,1} \wedge \neg P_{2,1} \wedge \neg P_{4,1}) \wedge (P_{4,1} \Rightarrow \neg P_{1,1} \wedge \neg P_{2,1} \wedge \neg P_{3,1})\\ \wedge(P_{1,2} \Rightarrow \neg P_{2,2} \wedge \neg P_{3,2} \wedge \neg P_{4,2}) \wedge (P_{2,2} \Rightarrow \neg P_{1,2} \wedge \neg P_{3,2} \wedge \neg P_{4,2})\\ \wedge(P_{3,2} \Rightarrow \neg P_{1,2} \wedge \neg P_{2,2} \wedge \neg P_{4,2}) \wedge (P_{4,2} \Rightarrow \neg P_{1,2} \wedge \neg P_{2,2} \wedge \neg P_{3,2})\\ \wedge(P_{1,3} \Rightarrow \neg P_{2,3} \wedge \neg P_{3,3} \wedge \neg P_{4,3}) \wedge (P_{2,3} \Rightarrow \neg P_{1,3} \wedge \neg P_{3,3} \wedge \neg P_{4,3})\\ \wedge(P_{3,3} \Rightarrow \neg P_{1,3} \wedge \neg P_{2,3} \wedge \neg P_{4,3}) \wedge (P_{4,3} \Rightarrow \neg P_{1,3} \wedge \neg P_{2,3} \wedge \neg P_{3,3})\end{array}$$

  or, simply, for $n = 1$:

$$\begin{array}{c}(P_{1,1} \wedge P_{2,1})\\ \wedge\\ ((P_{1,1} \Rightarrow \neg P_{2,1}) \wedge (P_{2,1} \Rightarrow \neg P_{1,1}))\end{array}$$

  Remark that this kind of schema appears very naturally when we express problems in propositional logic. The same expression (i.e. the schematic one) of the pigeonhole principle appears e.g. in [Bus87, BP97].

- The 3-colouring of a graph : consider a graph whose nodes are numbered from 1 to n, then, whatever is the number of nodes in the graph, we can express the fact that it is 3-colorable in propositional logic:

$$\bigwedge_{i=1}^{n}(R_i \wedge \neg V_i \wedge \neg B_i) \vee (\neg R_i \wedge V_i \wedge \neg B_i) \vee (\neg R_i \wedge \neg V_i \wedge B_i)$$
$$\wedge$$
$$\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n} EDGE_{i,j} \Rightarrow [(R_i \wedge \neg R_j) \vee (V_i \wedge \neg V_j) \vee (B_i \wedge \neg B_j)]$$

where R_i, V_i and B_i express the fact that the i^{th} node is red, green or blue, respectively, and $EDGE_{i,j}$ expresses the fact that there exists an edge from the i^{th} to the j^{th} node in the graph.

Remark that we could also express the k-colouring, for every $k \geq 1$, by using a proposition $COUL_{i,j}$ meaning that the i^{th} node has the colour numbered j.

- Iterated schemas of terms. Schemas of first-order terms are another example of iterated schemas in logic. For instance, the expression $\text{succ}^n(\text{a})$ is an iterated schema of terms. Every instance of this schema is a first-order term:

$$\begin{array}{ll} n = 0: & \text{a} \\ n = 1: & \text{succ(a)} \\ n = 2: & \text{succ(succ(a))} \\ \ldots & \ldots \end{array}$$

Thus, if "succ" stands for the successor function and "a" for the integer 0, then every term representing an integer follows the schema $\text{succ}^n(\text{a})$.

Iterated schemas of terms have been studied formally to try to prevent divergence in Knuth-Bendic completion algorithm [KB70]. Many formalisms have been introduced with the goal of having the biggest power of expression while preserving the decidability of unification: recurrent terms [CHK90], termes with integer exponents [Com95], R-terms [Sal92], primal grammars [HG97]. A detailed presentation of terms schemas and of the motivations of their study is presented in [Her94]. We also cite P-grammars [ACP08, ACP10f] that we have introduced during the thesis, but whose contents are not presented here due to the lack of direct link with formula schemas.

Iterated schemas are thus more complex than just "objects with holes". Indeed, we can adapt the previous definition of schemas simply by stating that a pattern is no more an "element of the language with holes" but a *computable function whose codomain is the language*. Every parameter of this function is then a "hole". For instance, we can define the schema $p_1 \wedge (\bigwedge_{i=1}^{n} p_i \Rightarrow p_{i+1}) \wedge \neg p_{n+1}$ as the computable function $g(n) \stackrel{\text{def}}{=} p_1 \wedge f(n) \wedge \neg p_{n+1}$, where f is defined as follows:

$$f(n) \stackrel{\text{def}}{=} \begin{cases} \top & \text{si } n = 0 \\ (p_n \Rightarrow p_{n+1}) \wedge f(n-1) & \text{sinon} \end{cases}$$

A philosopher would probably say here that the *ontological commitment* is much more important with iterated schemas that with "simple" schemas: indeed a formalisation of the notion of algorithme is needed to express iterated schemas. Furthermore n must be interpreted as an *integer*, which obviously, cannot be specified in first-order logic. However the fact that a schema is still a purely syntactic object whose goal is to finitely represent an infinite set still remains.

1.2 Motivations

Il est clair que l'utilisation de schémas permet un gain en abstraction et en expressivité : le schéma d'axiome de récurrence permet d'exprimer des théories que nous ne pourrions pas exprimer avec de simples axiomes; le schéma du principe des pigeonniers permet d'exprimer ce problème pour toute valeur de n, ce que la logique propositionnelle seule est incapable de faire; les schémas de termes permettent de raisonner simultanément sur une infinité de termes. De plus, l'utilisation de schéma peut permettre de diminuer considérablement la *taille* des preuves [BZ94], ce qui va aussi dans le sens d'un gain en abstraction.

Cependant, le bénéfice n'est, a priori, pas aussi important que celui que peuvent apporter, p.ex. les logiques d'ordre supérieur. En tout cas, il est clair que tous les exemples fournis jusqu'ici sont exprimables dans une logique d'ordre supérieur. Par conséquent il est naturel de se demander *pourquoi s'intéresser aux schémas plutôt qu'à des logiques déjà existantes et notoirement plus expressives ?*

Note. Une réponse plus complète sera apportée dans le chapitre 10 sur les travaux connexes.

1.2.1 Formalisation et analyse des preuves

Une des motivations principales de ces recherches provient des travaux du groupe dirigé par Alexander Leitsch qui, avec le système CERES [DLL+10], élimine les coupures dans des preuves véritablement issues

de développements mathématiques, voir p.ex. [BHL+08]. La méthode CERES n'utilise pas l'algorithme classique d'élimination des coupures [Gen35] mais, à partir d'une preuve formelle, génère un ensemble de clauses qui sera fourni à un démonstrateur automatique basé sur la résolution [Rob65]. À partir de la preuve obtenue, nous pouvons extraire une preuve sans coupure du théorème original. Dans ce contexte, il est fréquent de rencontrer des schémas de preuve. Le raisonnement sur ces schémas est actuellement effectué manuellement, ce qui est évidemment fastidieux. Il serait donc particulièrement utile de pouvoir *automatiser* le raisonnement sur ces schémas.

Comme nous l'avons dit, la méthode CERES est basée sur la résolution, l'idée étant de pouvoir utiliser pour éliminer les coupures des procédures de preuve notoirement efficaces. Cependant ces dernières impliquent l'utilisation de la logique du premier ordre. Dans ce contexte les schémas sont un moyen simple de pouvoir gagner en abstraction tout en espérant pouvoir réutiliser les procédures existantes.

Note. Le sujet d'étude de cette thèse fait partie intégrante du projet ASAP ("About Schemata And Proofs") de l'ANR impliquant l'équipe CAPP du LIG et l'équipe Theoretische Informatik und Logik (Theory and Logic Group) d'Alex Leitsch à Vienne (ANR-09-BLAN-0407-01) (`http://membres-lig.imag.fr/peltier/ASAP`).

1.2.2 Structure et lisibilité des preuves formelles

Nous prétendons que, dans de nombreux cas, les schémas sont un gain en abstraction qui peut être plus *lisible* que son pendant à l'ordre supérieur. Nous pensons que ce gain en lisibilité provient notamment du fait que les schémas préservent la *structure* de ses instances.

Reprenons l'exemple :

$$\forall n \geq 1, \forall x_1, \ldots, x_n \in \mathbb{R}^+ : \left(\sum_{i=1}^{n} x_i^2\right) \leq \left(\sum_{i=1}^{n} x_i\right)^2 \tag{1.1}$$

Pourquoi le mathématicien écrit-il cette formule plutôt que la suivante, plus "rigoureuse" :

$$\forall n \geq 1, \forall f : \mathbb{N} \mapsto \mathbb{R}^+ : g(n) \leq h(n)^2 \tag{1.2}$$

où :

$$g(n) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{si } n = 0 \\ f(n)^2 + g(n-1) & \text{sinon} \end{cases}$$

$$h(n) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{si } n = 0 \\ f(n) + h(n-1) & \text{sinon} \end{cases}$$

Selon nous, la raison en est que (1.1) est *plus lisible*, *plus naturelle* que (1.2). En effet, pour cet exemple, n'importe quel lecteur ayant un minimum de connaissance en mathématiques connaîtra le résultat dans le cas $n = 2$, c.-à-d. :

$$\forall n \geq 1, \forall x_1, x_2 \in \mathbb{R}^+ : x_1^2 + x_2^2 \leq (x_1 + x_2)^2 \tag{1.3}$$

Or (1.1) apparaît de façon évidente comme étant une généralisation de ce résultat connu. En revanche, il faut plus de réflexion pour comprendre qu'il est équivalent d'écrire (1.2). Le point important que nous voulons mettre en valeur est que *la structure de (1.3) est préservée dans (1.1)*, mais pas dans (1.2). Nous pensons que ceci est la raison qui rend la formule (1.1) plus lisible. En tout cas, l'omniprésence dans le raisonnement mathématique d'expressions de la forme $\forall x_1, \ldots, x_n \in E : \phi$ (où "$\ldots$" fait partie du langage) là où nous pourrions avoir de façon plus rigoureuse $\forall f : \mathbb{N} \mapsto E : \phi$, nous semble justifier le fait que l'écriture schématique est souvent plus lisible que l'écriture "rigoureuse".

Nous pensons donc que, dans un contexte de formalisation des mathématiques, l'utilisation de schémas peut permettre d'obtenir une formalisation plus proche d'un raisonnement humain (c.-à-d. plus lisible, plus structurée) que ne le permet l'ordre supérieur. Évidemment, ceci n'est pas une vérité générale car beaucoup de concepts mathématiques s'expriment bien plus naturellement à l'ordre supérieur. Nous affirmons seulement qu'un raisonnement formel est beaucoup plus complexe que ne l'est son pendant naturel, et nous pensons que les schémas peuvent dans de nombreux cas atténuer cette complexité en apportant plus de structure et de lisibilité.

1.2.3 Automatisation

Évidemment, les logiques d'ordre supérieur pêchent clairement par leur manque d'automatisation : nous cherchons à avoir des propriétés au mieux de décidabilité et au moins de complétude (en fait nous aimerions préserver autant que possible les propriétés du langage d'origine). Dans ce contexte il est naturel de considérer des extensions de la logique du premier ordre présentant un compromis entre expressivité et automatisation. Ceci justifie, une fois encore, que nous nous intéressions à une logique "bridée" comme celle des schémas.

1.2.4 Une forme d'abstraction différente

Enfin nous pensons que les schémas itérés peuvent avoir un intérêt par eux-mêmes : la façon habituelle de gagner en abstraction à partir d'une logique est de passer à l'ordre supérieur. Nous pensons que les schémas itérés présentent une façon d'abstraire différente. Là où l'ordre supérieur permet un gain en abstraction via la *quantification*, les schémas permettent ce gain via le *calcul*. En effet, comme nous l'avons vu, les schémas itérés peuvent être vus comme de simples fonctions calculables dont le codomaine serait l'ensemble des formules propositionnelles. Au lieu de considérer des fonctions calculables nous pourrions en fait considérer n'importe quelle modèle Turing équivalent. Les schémas que nous considérons ici sont alors un cas très particulier d'abstraction de ce type puisque nous considérons des fonctions calculables très peu expressives. Nous pensons que ce type d'abstraction présente une alternative intéressante à l'habituel ordre supérieur.

1.3 But de la thèse

Nos objectifs sont :

- définir de façon formelle les schémas itérés de formules propositionnelles : leur syntaxe, leur sémantique;
- étudier les possibilités de raisonnement automatique sur ces schémas : définir des procédures permettant ce raisonnement et identifier des classes décidables.

Vues les motivations précédentes, le lecteur peut se demander : *pourquoi des schémas en logique propositionnelle plutôt qu'en logique du premier ordre ?* La réponse est simplement que l'automatisation du raisonnement est déjà difficile sur les schémas propositionnels (voir le théorème d'incomplétude 3.13), par conséquent il est plus réaliste de commencer par explorer les schémas propositionnels avant de passer aux schémas du premier ordre.

De plus cela permet de se focaliser uniquement sur la structure *logique* des formules sans se soucier des termes. Une extension des schémas au premier ordre nécessiterait l'intégration de schémas de termes. Notons que, dans le cadre du projet ASAP, Hicham Bensaid étudie déjà dans sa thèse l'intégration de schémas de termes aux démonstrateurs automatiques actuels, mais sans schémas de formules [BCP07, BCP09, BCP10a, BCP10b].

1.4 Applications

Notons qu'il existe de nombreux schémas propositionnels intéressants. Nous avons déjà vu le principe des pigeonniers et la k-coloriabilité d'un graphe. Nous verrons dans la suite un exemple plus concret : la formalisation de circuits indépendamment du nombre de bits de leur entrée. Considérons par exemple un circuit additionneur à propagation de retenue qui calcule la somme S de deux vecteurs X et Y. Un tel circuit peut être spécifié de la façon suivante :

$$\text{additionneur} \stackrel{\text{def}}{=} \neg\text{retenue}_1 \wedge \bigwedge_{i=1}^{n} \text{additionneur\_complet}_i$$

où :

- $\text{additionneur\_complet}_i \stackrel{\text{def}}{=} (\text{somme}_i \Leftrightarrow S_i) \wedge (\text{retenue}_i \Leftrightarrow R_{i+1})$

- $\text{somme}_i \stackrel{\text{def}}{=} (X_i \oplus Y_i) \oplus R_i$
- $\text{retenue}_i \stackrel{\text{def}}{=} (X_i \wedge Y_i) \vee (Y_i \wedge R_i) \vee (X_i \wedge R_i)$

somme_i calcule le bit de sortie de la première cellule; retenue_i calcule la retenue R_{i+1}, propagée à la cellule suivante; n désigne le nombre de bits de X et Y. L'avantage d'utiliser les schémas ici plutôt que la logique propositionnelle est que nous pouvons raisonner sur ce circuit *quel que soit le nombre de bits de ses entrées*.

Note. De plus, une fois encore, la structure des formules propositionnelles sous-jacentes est préservée. Comparativement, le même circuit est modélisé dans [BK95] en logique monadique du second ordre faible [Bü59]. Ainsi, en considérant les propositions X_i, Y_i, R_i et S_i comme des variables de prédicat monadiques $X(i)$, $Y(i)$, $R(i)$ et $C(i)$ plutôt que comme des variables propositionnelles, nous obtenons la formule suivante :

$$\neg\text{retenue}_1 \wedge \forall i \leq n(\text{additionneur\_complet}_i)$$

(la formule n'est pas exactement la même, mais nous adaptons les notations pour des questions de lisibilité). Cette formule ne préserve pas la structure des formules propositionnelles sous-jacentes (ce qui ne l'empêche pas d'être très lisible).

1.5 Plan de la thèse

Le mémoire s'organise de la façon suivante :

- Le chapitre 2 présente les notions de base nécessaires pour la compréhension de la thèse : logique propositionnelle, arithmétique linéaire, méthode des tableaux, etc.
- Le chapitre 3 définit formellement les schémas itérés de formules propositionnelles. La sémantique des schémas est définie très simplement : un schéma est insatisfaisable ssi toutes ses instances sont insatisfaisables. Nous démontrons l'incomplétude de cette logique : il ne peut pas exister de procédure complète (pour la réfutation) pour les schémas. Par conséquent le problème de la satisfaisabilité pour les schémas est indécidable. Différents outils utiles pour la suite sont aussi introduits.
- Le chapitre 4 présente la première procédure de preuve définie sur les schémas : Stab. Cette procédure étend les tableaux propositionnels aux schémas de formules propositionnelles. Nous démontrons la correction et la complétude *pour la satisfaisabilité* de la procédure : en effet bien qu'il soit impossible d'obtenir une procédure complète pour la réfutation, il est possible d'obtenir la complétude pour la satisfaisabilité, c.-à-d. que nous pouvons toujours trouver un modèle si la conjecture en entrée de la procédure est satisfaisable.
- Dû à l'incomplétude, Stab ne termine pas en général : la procédure diverge. Cependant, dans de nombreux cas, nous pouvons prévenir la divergence en ajoutant une procédure de *détection de cycle*, ce qui correspond, mathématiquement, à une preuve par récurrence. Le chapitre 5 définit cette notion de façon générale (et indépendante de la procédure de preuve utilisée) puis introduit des raffinements dont nous démontrons la décidabilité. Nous démontrons aussi des résultats importants pour les démonstrations ultérieures.
- Le chapitre 6 présente une classe décidable (et donc complète pour la réfutation) de schémas propositionnels : les *schémas réguliers*. Nous démontrons la terminaison de Stab pour cette classe, puis nous définissons une autre procédure dédiée aux schémas réguliers : SchAut. Le raisonnement est plus simple sur cette procédure que sur Stab ce qui nous permet d'obtenir un résultat de complexité pour les schémas réguliers.
- Le chapitre 7 présente RegStab, une implémentation de Stab dédiée aux schémas réguliers. Le fonctionnement du logiciel, des exemples, et quelques détails d'implémentation sont présentés.
- Un inconvénient de Stab est son inaptitude à travailler avec des schémas *imbriqués*. Pour résoudre ce problème nous définissons au chapitre 8 la procédure Dpll* qui étend la procédure de Davis-Putnam-Logemann-Loveland aux schémas. Nous démontrons qu'elle est correcte et complète pour la satisfaisabilité.

- Le chapitre 9 met à profit la définition de DPLL* pour démontrer la complétude d'une nouvelle classe de schémas : les *schémas réguliers imbriqués*. Cette preuve constitue l'essentiel du chapitre.
- Le chapitre 10 passe en revue les travaux similaires, l'ensemble des alternatives possibles aux outils présentés dans cette thèse, leurs avantages et leurs inconvénients par rapport à notre approche. Notons que les schémas itérés de formules n'ont jamais été étudiés à notre connaissance, par conséquent il est nécessaire d'effectuer des traductions vers d'autres formalismes pour pouvoir faire la comparaison avec les travaux voisins. Évidemment, pour faire ces traductions, le formalisme des schémas itérés doit être connu, c'est pourquoi ce chapitre est présenté à la fin du mémoire. De plus certains travaux similaires ne sont comparables qu'à des sous-classes de schémas qui ne seront présentées que tardivement dans le mémoire.
- Enfin le chapitre 11 dresse le bilan des travaux effectués et esquisse les pistes de recherche envisageables à partir de ces travaux.

Tous les travaux présentés ont été publiés. La version que nous présentons dans ce mémoire est plus détaillée, de nombreuses erreurs sont corrigées, et les résultats sont plus généraux, ce qui sera mentionné le cas échéant. À titre indicatif, nous donnons la correspondance approximative entre les publications et le contenu de la thèse :

- [ACP09b] : chapitres 3, 4 et partiellement 6;
- [ACP09a] : chapitre 8;
- [ACP10c] : chapitre 6;
- [ACP10e] : chapitre 7;
- [ACP10a, ACP10b] : chapitres 5, 8 et 9;
- [ACP10d] : esquissé dans les sections 3.B et 9.A;
- [ACP08, ACP10f] ne sont pas présentés dans ce mémoire par souci de cohérence car ils traitent des schémas de termes et pas des schémas de formules.

CHAPTER 2

Preliminaries

Ce chapitre rappelle rapidement des notions habituelles et fixe les notations.

2.1 Propositional Logic

Le travail présenté dans ce manuscrit se base essentiellement sur la logique propositionnelle. Pour une présentation détaillée voir p. ex. [KK67, Caf10, End72, Eps90].

Definition 2.1 (Syntaxe)**.** *Soit $\mathcal{X}$ un ensemble infini dénombrable de* variables propositionnelles*. L'ensemble des* formules propositionnelles *en forme normale négative (f.n.n.) est le plus petit ensemble E tel que :*

- $\top, \bot \in E$*;*
- *pour toute variable propositionnelle x : $x, \neg x \in E$;*
- *si $\phi_1, \phi_2 \in E$ alors $\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2 \in E$.*

Note. Nous nous restreignons *tout au long de la thèse* à des formules propositionnelles en f.n.n.

Les variables ainsi que $\top$ et $\bot$ sont appelés *atomes*. Nous appelons *littéral* tout atome ou négation d'un atome. Un littéral est *positif* si c'est un atome, *négatif* si c'est la négation d'un atome. Le *complémentaire* d'un littéral l, noté l^{c}, est défini par :

$$\bot^{\mathrm{c}} \stackrel{\text{def}}{=} \top \quad \top^{\mathrm{c}} \stackrel{\text{def}}{=} \bot \quad (\neg x)^{\mathrm{c}} \stackrel{\text{def}}{=} x \quad x^{\mathrm{c}} \stackrel{\text{def}}{=} \neg x$$

Deux littéraux sont *opposés* si l'un est le complémentaire de l'autre.

Nous utiliserons les abréviations suivantes :

$$\begin{aligned} \phi_1 \Rightarrow \phi_2 &\stackrel{\text{def}}{=} \neg\phi_1 \vee \phi_2 \\ \phi_1 \Leftrightarrow \phi_2 &\stackrel{\text{def}}{=} (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1) \\ \phi_1 \oplus \phi_2 &\stackrel{\text{def}}{=} \neg(\phi_1 \Leftrightarrow \phi_2) \end{aligned}$$

Note. L'élimination de ces abréviations est exponentielle donc en pratique nous utiliserons des méthodes plus fines, voir p.ex. [PG86, BdlT92].

Une occurrence d'un littéral est *positive* dans une formule propositionnelle ssi cette occurrence n'apparaît pas sous une négation. Un littéral apparaît *positivement* dans une formule ssi il a une occurrence positive. Une *interprétation* est une fonction des variables propositionnelles dans l'ensemble des *valeurs de vérité* $\{\text{false}, \text{true}\}$.

Definition 2.2 (Sémantique)**.** *La relation de satisfaction $\models$ est définie inductivement :*

- $\mathfrak{I} \models x$ *ssi* $\mathfrak{I}(x) = \text{true}$*, où* x *est une variable;*
- $\mathfrak{I} \models \top$ *ssi* $\mathfrak{I} \not\models \bot$*;*
- $\mathfrak{I} \models \neg x$ *ssi* $\mathfrak{I} \not\models x$*;*
- $\mathfrak{I} \models \phi_1 \wedge \phi_2$ *ssi* $\mathfrak{I} \models \phi_1$ *et* $\mathfrak{I} \models \phi_2$*;*
- $\mathfrak{I} \models \phi_1 \vee \phi_2$ *ssi* $\mathfrak{I} \models \phi_1$ *ou* $\mathfrak{I} \models \phi_2$*.*

Ces notations seront utilisées pour toutes les logiques rencontrées par la suite : si ϕ est une formule et $\mathfrak{I}$ une interprétation, $\mathfrak{I} \models \phi$ signifie que ϕ est vraie dans $\mathfrak{I}$. Les notions de *modèle*, *contre-modèle*, *validité*, *satisfaisabilité* sont les mêmes que d'habitude :

- une interprétation $\mathfrak{I}$ telle que $\mathfrak{I} \models \phi$ est un *modèle* de ϕ;
- une interprétation $\mathfrak{I}$ telle que $\mathfrak{I} \not\models \phi$ est un *contre-modèle* de ϕ;
- si pour toute interprétation $\mathfrak{I}$ nous avons $\mathfrak{I} \models \phi$, alors ϕ est dite *valide*;
- si pour toute interprétation $\mathfrak{I}$ nous avons $\mathfrak{I} \not\models \phi$, alors ϕ est dite *insatisfaisable*;
- s'il existe une interprétation $\mathfrak{I}$ telle que $\mathfrak{I} \models \phi$, alors ϕ est dite *satisfaisable.*

L'équivalence de deux formules ϕ_1 et ϕ_2 sera notée $\phi_1 \equiv \phi_2$.

Definition 2.3 (Forme normale disjonctive)**.** *Une formule* ϕ *est en* forme normale disjonctive *ssi c'est une disjonction de conjonctions de littéraux.*
Dans ce cas, une conjonction de littéraux est appelée une clause conjonctive*.*

Le résultat suivant est classique :

Proposition 2.4. *Pour toute formule propositionnelle* ϕ*, il existe une formule* ϕ' *en forme normale disjonctive telle que* $\phi \equiv \phi'$*.* ϕ' *est calculable.*

Enfin nous rappelons la notion de *littéral pur*, fréquemment utilisée en déduction automatique.

Definition 2.5 (Littéral pur)**.** *Un littéral est* pur *dans une formule ssi sa négation n'apparaît pas positivement dans cette formule.*

On a alors le résultat suivant :

Theorem 2.6. *Si un littéral est pur dans une formule, alors la satisfaisabilité de la formule est préservée lorsque l'on évalue ce littéral à* true*.*

2.2 Linear Arithmetic

Nous rappelons maintenant l'arithmétique linéaire, i.e. l'arithmétique privée de la multiplication [1], et les propriétés qui font son intérêt. Pour une présentation approfondie voir p. ex. [Coo72].

Definition 2.7 (Expressions arithmétiques linéaires)**.** *Étant donné un ensemble infini dénombrable de* variables entières*, les* expressions arithmétiques linéaires *(ou, par la suite, simplement* expressions*) sont les éléments du plus petit ensemble* $\mathrm{E}_{\mathrm{lin}}$ *tel que :*

- *toute variable entière appartient à* $\mathrm{E}_{\mathrm{lin}}$*;*
- $\mathbb{Z} \subseteq \mathrm{E}_{\mathrm{lin}}$*;*
- *si* $e \in \mathrm{E}_{\mathrm{lin}}$ *alors* $-e \in \mathrm{E}_{\mathrm{lin}}$*;*
- *si* $e_1 \in \mathrm{E}_{\mathrm{lin}}$ *et* $e_2 \in \mathrm{E}_{\mathrm{lin}}$ *alors* $e_1 + e_2 \in \mathrm{E}_{\mathrm{lin}}$*,* $e_1 - e_2 \in \mathrm{E}_{\mathrm{lin}}$*.*

[1] Nous employons le terme "arithmétique linéaire" et pas "arithmétique de Presburger" parce que les entiers considérés peuvent être négatifs.

Soient deux expressions entières de la forme $x + q_1$ et $x + q_2$ où $q_1, q_2 \in \mathbb{Z}$. On définit l'ordre partiel $\preceq$ sur les expressions arithmétiques par $x + q_1 \preceq x + q_2$ ssi $q_1 \leq q_2$.

Un *environnement* est une fonction qui associe un entier relatif à chaque variable entière. La valeur d'une expression e dans un environnement ρ est définie de la façon suivante :

- $\rho(x) \stackrel{\text{def}}{=} \rho(x)$ si x est une variable entière
- $\rho(n) \stackrel{\text{def}}{=} n$ si $n \in \mathbb{Z}$
- $\rho(-e) \stackrel{\text{def}}{=} -\rho(e)$
- $\rho(e_1 + e_2) \stackrel{\text{def}}{=} \rho(e_1) + \rho(e_2), \rho(e_1 - e_2) \stackrel{\text{def}}{=} \rho(e_1) - \rho(e_2)$

Deux termes sont équivalents ssi ils ont la même valeur dans tout environnement (p. ex. $3 + 2 \equiv 5$, $x + 2 \equiv 2 + x$, $(x + y) - y \equiv x + (y - y) \equiv -y + x + y \equiv x$). Cette relation est connue pour être décidable [Coo72].

Pour tout environnement ρ, nous définissons une forme particulière de composition, notée $\circledcirc$, de la façon suivante : $\rho \circledcirc \{x \mapsto n\}(x) \stackrel{\text{def}}{=} n$ et $\rho \circledcirc \{x \mapsto n\}(y) \stackrel{\text{def}}{=} \rho(y)$ si $y \neq x$ (nous n'utilisons pas la composition habituelle simplement parce que le domaine de départ d'un environnement est différent de son domaine d'arrivée).

Definition 2.8 (Formule)**.** *Les formules de l'arithmétique linéaire sont les éléments du plus petit ensemble* Φ_{lin} *tel que :*

- *si* e_1 *et* e_2 *sont des expressions arithmétiques linéaires, alors* $e_1 = e_2 \in \Phi_{\text{lin}}$, $e_1 \neq e_2 \in \Phi_{\text{lin}}$, $e_1 < e_2 \in \Phi_{\text{lin}}$*;*
- *si* $k \in \mathbb{N}$ *et* e *est une expression arithmétique linéaire, alors* $k|e \in \Phi_{\text{lin}}$ *et* $k\not|e \in \Phi_{\text{lin}}$*;*
- Φ_{lin} *est clos par conjonction, disjonction et négation;*
- *si* x *est une variable entière et* $\phi \in \Phi_{\text{lin}}$ *alors* $\forall x\phi \in \Phi_{\text{lin}}$ *et* $\exists x\phi \in \Phi_{\text{lin}}$.

Note. Évidemment $\neq$ n'est pas nécessaire car définissable. Nous utiliserons l'abréviation $e_1 \leq e_2 \stackrel{\text{def}}{=} e_1 < e_2 + 1$. De même $k|e$ ("k divise e") est définissable par $k|e \equiv \exists f(e = kf)$. Ce prédicat n'est véritablement utile que pour assurer l'élimination des quantificateurs. Enfin $k\not|e$ est aussi définissable : $k\not|e \equiv 1|e \vee 2|e \vee \ldots \vee k - 1|e$.

Un environnement fait office d'interprétation pour les formules arithmétiques. Nous utilisons donc toujours la notation $\models$.

Definition 2.9 (Sémantique)**.** *La relation de satisfaction d'une formule* ϕ *dans un environnement* ρ *est définie par induction:*

- $\rho \models e_1 \bullet e_2$ *ssi* $\rho(e_1) \bullet \rho(e_2)$ *où* $\bullet \in \{=, \neq, <\}$
- $\rho \models k|e$ *ssi* k *divise* e *et* $\rho \models k\not|e$ *ssi* k *ne divise pas* e*.*
- *la sémantique de* $\neg$, $\vee$ *et* $\wedge$ *est la même qu'en logique propositionnelle (Définition 2.2);*
- $\rho \models \forall x\phi$ *ssi pour tout* $n \in \mathbb{Z}$, $\rho \circledcirc \{x \mapsto n\} \models \phi$*;*
- $\rho \models \exists x\phi$ *ssi il existe* $n \in \mathbb{Z}$, $\rho \circledcirc \{x \mapsto n\} \models \phi$.

L'*élimination des quantificateurs* est une propriété classique de l'arithmétique linéaire [KK67, Coo72], et particulièrement utile par la suite :

Theorem 2.10 (Élimination des quantificateurs)**.** *Soit une formule arithmétique* ϕ*. Il existe une formule* ϕ' *telle que* $\phi' \equiv \phi$ *et* ϕ' *ne contient ni* $\forall$ *ni* $\exists$ *(et une telle formule est calculable à partir de* ϕ*).*

Le théorème suivant est conséquence de l'élimination des quantificateurs :

Theorem 2.11. *Soient une formule arithmétique* ϕ *et un environnement* ρ*. Le problème de dire si* $\rho \models \phi$ *est décidable.*

Par la suite (section 3.C), il sera utile de considérer que toute formule arithmétique ϕ a les propriétés suivantes :

- ϕ ne contient pas de quantificateur;
- ϕ est sous forme normale disjonctive;
- ϕ ne contient pas de négation.

Alors ϕ est dite sous *forme normale arithmétique* (*f.n.a.*). Pour toute formule ϕ on obtient une formule équivalente ayant ces propriétés en lui appliquant les opérations suivantes dans cet ordre :

- élimination des quantificateurs (théorème 2.10);
- mise sous forme normale disjonctive (proposition 2.4);
- mise sous forme normale négative (application des règles de De Morgan) et élimination des négations :

$$\begin{array}{cc} \neg e_1 = e_2 \to e_1 \neq e_2 & \neg e_1 \neq e_2 \to e_1 = e_2 \\ \neg e_1 < e_2 \to e_2 \leq e_1 & \neg e_1 \leq e_2 \to e_2 < e_1 \\ \neg k|e \to k\not|e & \neg k\not|e \to k|e \end{array}$$

Il est clair que ces opérations terminent et que le résultat est une formule sous forme normale arithmétique équivalente à la formule d'origine :

Proposition 2.12. *Soit une formule arithmétique ϕ. Il existe une formule ϕ' telle que $\phi' \equiv \phi$ et ϕ' est sous forme normale arithmétique (et une telle formule est calculable à partir de ϕ).*

2.3 Tableaux sémantiques

Les règles des deux procédures décrites dans ce mémoire ("Stab", chapitre 4, et "Dpll*", chapitre 8), sont présentées dans le style de celles des tableaux sémantiques [Fit90, Smu68]. Nous rappelons donc cette notion ici.

Les tableaux sont souvent utilisés en déduction automatique, par ailleurs ils sont facilement traduisibles dans un système comme le calcul des séquents de Gentzen (voir p.ex. [Hod83]).

Definition 2.13 (Règle d'extension)**.** *Un* tableau *est un arbre étiqueté (nous laissons libre le type des étiquettes qui pourront aussi bien être des formules que des ensembles ou des paires de formules).*

Une règle d'extension *(ou simplement* règle*) est un tuple $(p, c_1, \ldots, c_n)$ où p, c_1, …, c_n sont des étiquettes avec des méta-variables. p est la* prémisse*, c_1, …, c_n les* conclusions*. Nous notons :*

$$\frac{p}{c_1 \mid \ldots \mid c_n}$$

à interpréter comme "si p est vrai alors l'un des c_1, …, c_n est vrai". Une étiquette est une instance *de la prémisse ssi elle peut être obtenue à partir de p par substitution des méta-variables.*

Supposons que l'étiquette d'une feuille ν d'un tableau t soit une instance de p. Dans ce cas t peut être étendu *en ajoutant n fils à ν, étiquetés par $c_1\sigma$, …, $c_n\sigma$ respectivement, où σ est la substitution des méta-variables faisant de l'étiquette de ν une instance de p.*

Une règle de clôture *indique quand une feuille est* close *(ou* fermée*), dans ce cas aucune règle ne peut s'appliquer sur cette feuille. Si toutes les feuilles d'un tableau sont closes on dit qu'il est* clos*.*

Quand aucune règle ne peut être appliquée sur une feuille, cette dernière est dite irréductible*.*

Une *dérivation* est une suite de tableaux $(t_i)_{i\in I}$ telle que I est soit $\mathbb{N}$, soit un intervalle $[0; n]$ où $n \in \mathbb{N}$, et pour tout $i \in I \setminus \{0\}$, t_i est obtenu par extension de t_{i-1}[2]. Les méthodes de preuve à définir comporteront plusieurs règles, aucun ordre n'étant imposé a priori sur la façon dont les règles doivent être appliquées (que ce soit dans le choix des règles ou dans le choix des feuilles qu'elles étendent). Selon

[2]Cette notion de dérivation est très basique mais suffisante pour ce que nous voulons en faire dans ce mémoire. On trouvera dans [Bon05] une formalisation beaucoup plus complète de cette notion pour les tableaux.

les cas nous pourrons donner une *stratégie* fixant cet ordre. En pratique, les règles seront accompagnées de *conditions d'application* précisant dans quel contexte elles peuvent être utilisées.

Intuitivement, la méthode des tableaux est appliquée de la façon suivante : soit une formule dont nous voulons savoir si elle est satisfaisable ou non; nous construisons le tableau ne contenant qu'un nœud étiqueté par cette formule; nous construisons une dérivation à partir de ce tableau; si nous obtenons un tableau clos alors la formule initiale est insatisfaisable; si nous obtenons un tableau dont l'une des feuilles est irréductible et non close alors elle est satisfaisable.

Definition 2.14 (Sémantique)**.** *Supposons qu'à chaque étiquette on puisse associer une formule*[3]*. Soit* $\mathfrak{I}$ *une interprétation, alors :*

- $\mathfrak{I}$ *satisfait un nœud ssi il satisfait la formule associée à son étiquette;*
- $\mathfrak{I}$ *satisfait un tableau ssi il satisfait une feuille de ce tableau.*

Classiquement, une règle d'extension est dite *correcte* ssi pour toute interprétation $\mathfrak{I}$: $\mathfrak{I}$ est modèle de la prémisse ssi il existe une conclusion ayant $\mathfrak{I}$ pour modèle. Une règle de clôture est *correcte* ssi toute instance de la prémisse est insatisfaisable. Enfin la notion de correction est étendue à un ensemble de règles de la façon suivante :

Definition 2.15 (Correction)**.** *Un ensemble de règles est* correct *ssi il a les propriétés suivantes :*

1. *toutes ses règles sont correctes;*
2. *toute feuille irréductible non close est satisfaisable.*

Cette notion est plus forte que la notion de correction utilisée habituellement (qui correspond au point 1 uniquement). En effet, nous aurons besoin du point 2 au chapitre 5. L'interprétation de ce second point est la suivante : le principe de nombreuses procédures de preuve consiste à "simplifier" successivement une formule de sorte à obtenir une formule dont la satisfaisabilité sera triviale. Il est donc essentiel que lorsqu'une feuille est irréductible, nous sachions de façon simple si elle est satisfaisable ou non. Dans le cas où une telle feuille est close, il est clair qu'elle est insatisfaisable. Mais si ce n'est pas le cas, alors il n'y pas de raison que la feuille soit satisfaisable. Le point 2 permet d'assurer que nous avons cette propriété.

Nous avons ensuite les deux résultats suivants :

Theorem 2.16. *Soit une formule* ϕ*, un tableau* t *ne contenant qu'un nœud étiqueté par* ϕ*, et une dérivation* d *à partir de* t*. Alors :*

- *si* d *contient un tableau clos alors* ϕ *est insatisfaisable;*
- *si* d *contient un tableau dont une feuille est irréductible et non close, alors* ϕ *est satisfaisable.*

Enfin nous rappelons la notion de complétude :

Definition 2.17 (Complétude pour la réfutation)**.** *Soit un ensemble de règles* R*. Nous disons que* R *est* complet pour la réfutation *ssi pour tout tableau restreint à un nœud dont l'étiquette est insatisfaisable, toute dérivation de* R *à partir de ce tableau contient un tableau clos.*

Note. Il s'agit d'une complétude "forte" dans le sens où on réclame que *toute dérivation* (et non pas qu'*il existe une* dérivation) contienne un tableau clos.

Comme nous le verrons, il est impossible d'avoir pour les schémas une procédure complète pour la réfutation. En revanche il est possible d'avoir la complétude pour la satisfaisabilité : nous pouvons garantir l'existence d'une branche irréductible non close lorsque l'étiquette de la racine est satisfaisable. Pour cela, nous devons définir une notion de dérivation équitable : une dérivation est *équitable* ssi soit il existe un i tel que t_i contient une feuille irréductible non close, soit pour tout i et toute feuille ν de t_i il existe $j \geq i$ tels qu'une règle s'applique en ν à t_j[4].

Definition 2.18 (Complétude pour la satisfaisabilité)**.** *Soit un ensemble de règles* R*. Nous disons que* R *est* complet pour la satisfaisabilité *ssi pour tout tableau restreint à un nœud dont l'étiquette est satisfaisable, toute dérivation équitable par* R *à partir de ce tableau contient un tableau ayant une feuille irréductible non close.*

[3] Des exemples de telles associations seront donnés ci-après.

[4] c.-à-d. qu'aucun choix n'est indéfiniment reporté.

2.3.1 Tableaux sémantiques propositionnels

À titre indicatif et pour faciliter la compréhension de STAB (Chapitre 4), nous présentons la méthode des tableaux dans le contexte de la logique propositionnelle (voir p.ex. [Smu68, Hod83]).

Definition 2.19 (Tableaux propositionnels)**.** *Un* tableau propositionnel *est un tableau étiqueté par des ensembles finis de formules propositionnelles.*

Les règles d'extension sont les suivantes :

$$\frac{\Phi \cup \{\phi_1 \wedge \phi_2\}}{\Phi \cup \{\phi_1\} \cup \{\phi_2\}} \qquad \frac{\Phi \cup \{\phi_1 \vee \phi_2\}}{\Phi \cup \{\phi_1\} \mid \Phi \cup \{\phi_2\}}$$

Une feuille est close quand l'ensemble qui l'étiquète contient deux littéraux opposés ou $\bot$*.*

Pour la sémantique (définition 2.14), la formule associée à chaque étiquette est la conjonction de ses éléments (p.ex. si un nœud est étiqueté par $\{A, B \vee \neg A, \neg C\}$, alors on lui associe la formule $A \wedge (B \vee \neg A) \wedge \neg C$).

Proposition 2.20. *La méthode des tableaux propositionnels termine quelle que soit la formule en entrée.*

On a alors le résultat suivant :

Theorem 2.21. *La méthode des tableaux propositionnels est correcte et complète pour la réfutation.*

2.3.2 DPLL

Nous rappelons DPLL (Davis-Putnam-Logemann-Loveland, [DLL62]). La présentation qui en est faite ici combine tableaux et réécriture.

Definition 2.22 (DPLL)**.** *Les tableaux sont étiquetés par des paires* (ϕ, L) *où* ϕ *est une formule propositionnelle et* L *un ensemble fini de littéraux.*

Les règles d'extension sont les suivantes :

$$\frac{(\phi, L)}{(\phi, L \cup \{\neg a\}) \mid (\phi, L \cup \{a\})}$$

si a *est un atome tel que* a *apparaît dans* ϕ*,* $a \notin L$ *et* $\neg a \notin L$

$$\frac{(\phi, L)}{(\phi', L)}$$

où ϕ' *est la forme normale de* ϕ *par le système suivant :*

$$\begin{cases} \neg\bot \to \top \quad \phi \wedge \top \to \phi \quad \phi \wedge \bot \to \bot \quad a \to \top \text{ si } a \in L \\ \neg\top \to \bot \quad \phi \vee \top \to \top \quad \phi \vee \bot \to \phi \quad a \to \bot \text{ si } \neg a \in L \end{cases}$$

Une feuille est close quand $\phi = \bot$*.*

Note. Il s'agit là d'une version basique de DPLL : il n'y a pas de propagation des clauses unitaires ni d'élimination des littéraux purs.

Le système de réécriture utilisé termine en utilisant la mesure qui associe à chaque formule le nombre de ses symboles. De plus il est confluent car orthogonal.

Pour la sémantique, on associe à chaque étiquette (ϕ, L) la conjonction de ϕ et de tous les éléments de L (p.ex. si un nœud est étiqueté par $(\neg A \vee B, \{C, \neg D, \neg E\})$ alors la formule associée est $(\neg A \vee B) \wedge C \wedge \neg D \wedge \neg E$).

Proposition 2.23. DPLL *termine quelle que soit la formule en entrée.*

on a alors le résultat suivant :

Theorem 2.24. DPLL *est correct et complet pour la réfutation.*

2.4 Ordres de terminaison

Afin de démontrer la terminaison des procédures définies tout au long de cette thèse, nous utiliserons des ordres bien fondés. Nous rappelons ici les notions relatives.

Definition 2.25. *Une relation d'ordre* $\lhd$ *est* bien fondée *ssi il n'existe pas de suite infinie* $x_1 \lhd x_2 \lhd \dots$

L'ordre naturel sur $\mathbb{N}$ est bien fondé (en prenant $>$ pour $\lhd$).

Definition 2.26. *Étant donné un ordre* $\lhd_1$ *sur un ensemble* E_1 *et un ordre* $\lhd_2$ *sur un ensemble* E_2*, on définit l'*extension lexicographique $\lhd_{\text{lex}}$ *de* $\lhd_1$ *et* $\lhd_2$ *sur* $E_1 \times E_2$ *de la façon suivante :*

$$\forall x_1, y_1 \in E_1, x_2, y_2 \in E_2 (x_1, x_2) \lhd_{\text{lex}} (y_1, y_2) \text{ ssi } x_1 \lhd_1 y_1 \text{ ou } x_1 = y_1 \text{ et } x_2 \lhd_2 y_2$$

Theorem 2.27. *L'extension lexicographique d'ordres bien fondés est bien fondée.*

Comme d'habitude on appelle multiensemble sur un ensemble E toute fonction de E dans $\mathbb{N}$. Nous ne considérerons que des multiensembles *finis*, c.-à-d. des fonctions nulles partout sauf sur un ensemble fini de points. Un multiensemble fini M est noté :

$$\{\underbrace{x_1, \dots, x_1}_{\times M(x_1)}, \dots, \underbrace{x_n, \dots, x_n}_{\times M(x_n)}\}$$

pour tous $x_i \in E$, $i \in [1; n]$, tels que $M(x_i) \neq 0$. Nous définissons alors l'ordre multiensemble :

Definition 2.28. *Étant donné un ordre* $\lhd$ *sur un ensemble* E*, l'*extension multiensemble $\lhd_{\text{mul}}$ *de* $\lhd$ *sur les multiensembles de* E *est définie de la façon suivante :*

$$\forall x, y_1 \dots y_n \in E, M \cup \{y_1, \dots, y_n\} \lhd_{\text{mul}} M \cup \{x\} \text{ ssi } y_i < x \text{ pour tout } i \in [1; n]$$

Theorem 2.29. *L'extension multiensemble d'ordres bien fondés est bien fondée.*

2.5 Langages et automates

Definition 2.30. *Soit un ensemble fini* A *appelé* alphabet*. L'ensemble des* mots *sur* A*, noté* A^**, est l'ensemble des suites finies d'éléments de* A*. On note* ϵ *le mot vide et* $m_1 \cdot m_2$ *la concaténation entre deux mots. L'*ordre préfixe $<$ *entre mots est défini de la façon suivante :* $m_1 < m_2$ *ssi il existe* m_3 *t.q.* $m_2 = m_1 \cdot m_3$*. Étant donné un mot* m_1 *et un préfixe* m_2 *de* m_1*, nous notons* $m_2 \setminus m_1$ *le mot tel que* $m_2 \cdot (m_2 \setminus m_1) = m_1$*.*

Nous nous servirons d'automates dans les exemples et pour définir SchAut (chapitre 6), une procédure de preuve à base d'automates.

Definition 2.31 (Automate)**.** *Soit un alphabet fini* L*. Un* automate fini non déterministe *est un tuple* $\langle Q, q_0, F, T \rangle$ *où* Q *est un ensemble fini d'*états*,* q_0 *est l'*état initial*,* F *est l'ensemble des* états finals *et* $T \subseteq Q \times L \cup \{\epsilon\} \times Q$ *est l'ensemble des* transitions*.*

Étant donné un mot m sur L, la *course* d'un automate a à partir d'un état q_1 est une suite finie d'états $(q_1, \dots, q_n)$ telle que pour tout $i \in [1; n-1]$, il existe $l \in L \cup \{\epsilon\}$ t.q. (q_i, l_i, q_{i+1}) est une transition de a et $m = l_1 \cdot \dots \cdot l_{n-1}$.

La course *accepte* m ssi q_n est un état final. Le mot m est accepté par a ssi il existe une course dont le premier état est l'état initial de a et acceptant m.

Enfin une transition dont l'étiquette est ϵ est appelée ϵ-transition.

CHAPTER 3

Formulae Schemata

Nous avons vu en introduction que notre but était d'exprimer des objets de la forme $\bigwedge_{i=1}^{n} \dots$ ou $\bigvee_{i=1}^{n} \dots$ (appelés "itérations"). Donc une première[1] possibilité est d'introduire des connecteurs $\bigwedge$ et $\bigvee$ ayant trois paramètres : une borne inférieure, une borne supérieure et une formule. Cette formalisation est utilisée dans [ACP09b].

Nous pouvons aussi voir $\bigwedge_{i=1}^{n}$ comme $\bigwedge_{i|1\leq i\wedge i\leq n}$ où $i|1 \leq i \wedge i \leq n$ se lit "i tel que $1 \leq i \wedge i \leq n$", ce qui se généralise en $\bigwedge_{i|\phi}$ où ϕ est une formule de l'arithmétique linéaire. Nous pouvons alors écrire un schéma comme, p.ex., $\bigwedge_{i|\forall j \exists k(k\leq j \vee i\geq k)}$. Les connecteurs admettent donc maintenant deux paramètres : une formule arithmétique et un schéma. Nous verrons en annexe 3.A.1 que cela n'augmente pas le pouvoir d'expression (mais nous pouvons gagner considérablement en concision), en fait cette syntaxe ne sert qu'à simplifier l'expression des procédures travaillant sur les schémas. Cette formalisation a été considérée dans [ACP09a].

En pratique il est très utile de pouvoir restreindre les valeurs que peut prendre le paramètre, p.ex. nous pouvons vouloir considérer $\bigwedge_{i=1}^{n} \dots$ avec $n \geq 0$ uniquement. Ce sera particulièrement utile dans les procédures où nous pourrons raisonner par cas selon que le paramètre est inférieur ou supérieur à une certaine valeur. Un troisième raffinement consiste donc à accompagner chaque schéma d'une contrainte sur le(s) paramètre(s). Dans ce cas la formule en elle-même est appelée un "motif" et nous appelons "schéma" la paire d'un motif et d'une contrainte. Cette présentation a été développée dans [ACP10a, ACP10b] et c'est celle que nous adopterons ici. Notons qu'elle correspond à la définition générale de schéma esquissée dans [Cor06] où un schéma est décrit comme la paire d'un motif, c.-à-d. une construction syntaxique contenant des "trous", et d'une contrainte spécifiant la façon dont les trous doivent être remplis. Enfin, notons que l'introduction de contraintes n'augmente pas non plus le pouvoir d'expression car elles peuvent être simulées par des schémas comme nous le verrons en annexe 3.A.1.

3.1 Le langage

3.1.1 Syntax

On dit qu'une formule arithmétique ϕ *encadre* une variable entière i ssi il existe des expressions arithmétiques linéaires e et f ne contenant pas i telles que $\phi \models e \leq i \wedge i \leq f$. Par exemple, $n \leq i \wedge i \leq 3n+1$, $1 \leq i \wedge i \leq n \wedge i \neq j$ et $(1 \leq i \vee 3 \leq i) \wedge i \leq n$ encadrent i, mais pas $i \leq n$, $i \geq 3$ ou $i \geq 1 \vee i \leq n$.

[1] Nos véritables premiers essais [Ara07] utilisaient des schémas de termes [Her94] pour représenter des schémas de formules en représentant des formules comme des termes, les connecteurs logiques étant des symboles de fonction. Nous n'avons pas retenu cette démarche dans ce mémoire car, d'une part, la représentation des formules par des termes s'avère difficile à manipuler et, d'autre part, l'unification (dont la décidabilité est le principal apport des grammaires primales dans le cadre de nos travaux) ne joue pas un rôle essentiel, tant que nous restons au niveau de la logique propositionnelle (évidemment tout ceci serait à reconsidérer si nous travaillions avec des schémas de formules du premier ordre). Cependant ces essais ont donné lieu à des travaux sur les grammaires primales : [ACP08, ACP10f].

Definition 3.1 (Motif)**.** *Étant donné un ensemble de* symboles propositionnels*, l'ensemble des* motifs *(en forme normale négative) est défini par induction :*

- *si p est un symbole propositionnel et $e_1, \ldots, e_n$ des expressions arithmétiques linéaires alors $p_{e_1,\ldots,e_n}$ et $\neg p_{e_1,\ldots,e_n}$ sont des motifs;*
- *les conjonction et disjonction de motifs sont des motifs;*
- *si m est un motif, i une variable entière et ϕ une formule arithmétique qui encadre*[2] *i alors $\bigwedge_{i|\phi} m$ et $\bigvee_{i|\phi} m$ sont des motifs.*

Example 3.2.

$$\bigwedge_{i|1\leq i\wedge i\leq n+1}\left(\bigvee_{j|1\leq j\wedge j\leq n} p_{i,j}\right)$$
$$\wedge$$
$$\bigwedge_{j|1\leq j\wedge j\leq n}\ \bigwedge_{i_1|1\leq i_1\wedge i_1\leq n+1}\left(p_{i_1,j}\Rightarrow \bigwedge_{i_2|1\leq i_2\wedge i_2\leq n+1\wedge i_2\neq i_1}\neg p_{i_2,j}\right)$$

est un motif (exprimant le principe des pigeonniers, voir p.3).

Un motif de la forme $\bigwedge_{i|\phi} m$ (resp. $\bigvee_{i|\phi} m$) est appelée une *conjonction itérée* (resp. *disjonction itérée*). On appellera *itération* l'une ou l'autre des deux formes, m est appellé le *corps* de l'itération, ϕ son *domaine* et i son *itérateur*. Si ϕ est de la forme $e \leq i \wedge i \leq f$, alors nous écrirons $\bigwedge_{i=e}^{f} m$ et $\bigvee_{i=e}^{f} m$, auquel cas l'expression $f - e + 1$ est appelée la *longueur* de l'itération.

Example 3.3. *Le schéma de l'exemple précédent peut ainsi être écrit de façon plus concise et naturelle :*

$$\bigwedge_{i=1}^{n+1}\left(\bigvee_{j=1}^{n} p_{i,j}\right)$$
$$\wedge$$
$$\bigwedge_{j=1}^{n}\bigwedge_{i_1=1}^{n+1}\left(p_{i_1,j}\Rightarrow \bigwedge_{i_2|1\leq i_2\wedge i_2\leq n+1\wedge i_2\neq i_1}\neg p_{i_2,j}\right)$$

Dans ce motif, $\bigvee_{j=1}^{n} p_{i,j}$ est une itération, plus précisément une disjonction itérée. Son corps est $p_{i,j}$, son domaine est $1 \leq j \wedge j \leq n$, son itérateur est la variable j et sa longueur vaut n (intuitivement, nous pouvons dire que l'itération a n rangs, d'où le terme de "longueur").

Le motif $\bigwedge_{i=1}^{n+1}\left(\bigvee_{j=1}^{n} p_{i,j}\right)$ est aussi une itération, c'est une conjonction itérée. Son corps contient lui-même l'itération $\bigvee_{j=1}^{n} p_{i,j}$, son domaine est $1 \leq i \wedge i \leq n+1$, son itérateur est i et sa longueur vaut $n + 1$.

Remarquons l'itération $\bigwedge_{i_2|1\leq i_2\wedge i_2\leq n+1\wedge i_2\neq i_1}\neg p_{i_2,j}$ qui met à profit la possibilité d'utiliser n'importe quel formule arithmétique dans le domaine (et pas seulement une formule de la forme $e \leq i \wedge i \leq f$). Par conséquent cette itération n'a pas de longueur.

Un motif de la forme $p_{e_1,\ldots,e_n}$ est appelé une *proposition indexée* ou *atome*. Un atome ou la négation d'un atome est appelé un *littéral*. Le nombre d'indices d'une proposition indexée est appelé son *arité*. Une proposition indexée est *monadique* ssi elle n'a qu'un indice. Toutes les classes décidables que nous identifierons par la suite imposeront que les propositions indexées soient monadiques.

Example 3.4. *Soit le motif $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$ (rappelons que $s_1 \Rightarrow s_2 \stackrel{def}{=} \neg s_1 \vee s_2$ donc l'implication est autorisée dans un motif) : p_0, p_i, p_{i+1} et p_n sont des propositions indexées (et des littéraux), $\neg p_i$ et $\neg p_n$ sont des littéraux. Toutes les propositions indexées sont monadiques.*

[2] Cette contrainte assure que chaque itération est *finie*. Si ça n'était pas le cas les instances ne seraient pas nécessairement des formules propositionnelles (donc finies), voir Définition 3.8.

Les itérations lient leurs itérateurs, les définitions de variables libres et liées s'en suivent naturellement. Les variables libres d'un schéma sont appelées *paramètres* du schéma. La substitution est définie comme d'habitude, en évitant les captures pour les variables libres, et en ne substituant pas les variables liées.

Example 3.5. *Soit m le motif $p_i \wedge \bigwedge_{i=1}^{i} \neg q_i$: i est à la fois libre (dans p_i) et liée (par l'itération). C'est donc un paramètre du schéma. Nous notons $[3/i]$ la substitution qui associe 3 à i et $m[3/i]$ l'application de cette substitution à m : $m[3/i] = p_3 \wedge \bigwedge_{i=1}^{3} \neg q_i$.*

Lorsque nous définirons la sémantique des schémas, il sera clair que ce motif est en fait équivalent à $p_n \wedge \bigwedge_{i=1}^{n} \neg q_i$, ce qui est bien plus clair.

Definition 3.6 (Schéma)**.** *Un* schéma *est une expression de la forme $m \,\&\, c$ où m est un motif et c est une formule arithmétique appelée* contrainte*.*

Étant donné un schéma $s = m \,\&\, c$, m est le motif de s*, noté m_s, et c est la* contrainte de s*, notée c_s.*

Typiquement, une contrainte aura pour variables libres un ou des paramètre(s) du motif. Les *paramètres* du schéma sont ceux de son motif. Pour un schéma s nous abuserons souvent des notations de la façon suivante : si m' est un motif, alors $s \wedge m'$ désigne $(m_s \wedge m') \,\&\, c_s$, si c' est une formule arithmétique, alors $s \,\&\, c'$ désigne $m_s \,\&\, (c_s \wedge c')$, et si s' est un autre schéma alors $s \wedge s'$ désigne $(m_s \wedge m_{s'}) \,\&\, (c_s \wedge c_{s'})$. Quand la contrainte n'est pas précisée c'est qu'il s'agit de $\top$.

Example 3.7. $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$ *est un schéma. Son seul paramètre est n, son motif est $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$, sa contrainte est $n \geq 0$.*

3.1.2 Semantics

La sémantique des schéma peut être décomposée en deux parties : nous donnons d'abord une valeur à chaque paramètre, en accord avec la contrainte, ce qui permet d'obtenir une formule propositionnelle. Puis nous donnons une interprétation propositionnelle de cette formule. Par conséquent, une *interprétation de schéma* est la paire d'un environnement arithmétique et d'une interprétation propositionnelle. Étant donné une interprétation de schéma $\mathfrak{I}$, nous notons $\mathfrak{I}_{\text{arith}}$ son environnement et $\mathfrak{I}_{\text{prop}}$ son interprétation propositionnelle.

En utilisant l'environnement nous produisons une formule propositionnelle. Une telle formule est appelée une *instance* du schéma :

Definition 3.8 (Instance)**.** *Étant donnés un schéma s et un environnement ρ, l'*instance *de s par rapport à ρ, notée $\langle s \rangle_\rho$, est une formule propositionnelle définie de la façon suivante :*

- *si $\rho \not\models c_s$ alors $\langle s \rangle_\rho \stackrel{def}{=} \bot$*
- *sinon $\langle s \rangle_\rho \stackrel{def}{=} \langle m_s \rangle_\rho$, où $\langle m_s \rangle_\rho$ est défini récursivement :*

$$
\begin{aligned}
\langle p_{e_1,\ldots,e_n} \rangle_\rho &\stackrel{def}{=} p_{\rho(e_1),\ldots,\rho(e_n)} \\
\langle \neg m \rangle_\rho &\stackrel{def}{=} \neg \langle m \rangle_\rho \\
\langle m_1 \wedge m_2 \rangle_\rho &\stackrel{def}{=} \langle m_1 \rangle_\rho \wedge \langle m_2 \rangle_\rho \\
\langle m_1 \vee m_2 \rangle_\rho &\stackrel{def}{=} \langle m_1 \rangle_\rho \vee \langle m_2 \rangle_\rho \\
\langle \bigwedge_{i|\delta} m \rangle_\rho &\stackrel{def}{=} \bigwedge \{ \langle m \rangle_{\rho \circledcirc \{i \mapsto I\}} \mid I \in \mathbb{Z} \text{ tel que } \rho \circledcirc \{i \mapsto I\} \models \delta \} \\
\langle \bigvee_{i|\delta} m \rangle_\rho &\stackrel{def}{=} \bigvee \{ \langle m \rangle_{\rho \circledcirc \{i \mapsto I\}} \mid I \in \mathbb{Z} \text{ tel que } \rho \circledcirc \{i \mapsto I\} \models \delta \}
\end{aligned}
$$

Note. Les symboles $\bigwedge$ et $\bigvee$ en membre droit ne désignent pas des connecteurs formels mais les opérations permettant d'obtenir les formules propositionnelles $\ldots \wedge \ldots \wedge \ldots$ et $\ldots \vee \ldots \vee \ldots$ ou $\top$ et $\bot$ quand les ensembles sont vides (d'où la distinction typographique). De telles itérations sont qualifiées de *bornées*, par opposition aux itérations du langage des schémas qui sont, elles, non bornées.

Comme le domaine de chaque itération encadre son itérateur, il est facile de voir que le résultat est une formule propositionnelle (donc finie) dont les atomes sont des propositions indexées par des nombres entiers relatifs.

Example 3.9. *Soit* $s = p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$, $\rho_{-1} = \{n \mapsto -1\}$, $\rho_0 = \{n \mapsto 0\}$, $\rho_1 = \{n \mapsto 1\}$. $\rho_2 = \{n \mapsto 2\}$. *Alors :*

$$\begin{aligned}\langle s\rangle_{\rho_{-1}} &= \bot \text{ (la contrainte n'est pas satisfaite)}\\ \langle s\rangle_{\rho_0} &= p_0 \wedge \top \wedge \neg p_0\\ \langle s\rangle_{\rho_1} &= p_0 \wedge (p_0 \Rightarrow p_1) \wedge \neg p_1\\ \langle s\rangle_{\rho_2} &= p_0 \wedge (p_0 \Rightarrow p_1) \wedge (p_1 \Rightarrow p_2) \wedge \neg p_2\end{aligned}$$

Definition 3.10 (Sémantique des schémas)**.** *Étant donnés un schéma s et une interprétation $\mathfrak{I} = (\mathfrak{I}_{\text{arith}}, \mathfrak{I}_{\text{prop}})$, $\mathfrak{I} \models s$ ssi $\mathfrak{I}_{\text{prop}} \models \langle s\rangle_{\mathfrak{I}_{\text{arith}}}$. Nous utiliserons toujours les notations -_{prop} et -_{arith} pour désigner les deux composantes d'une interprétation.*

Par conséquent *un schéma est insatisfaisable ssi toutes ses instances sont (propositionnellement) insatisfaisables*, a contrario il est satisfaisable ssi une seulement de ses instances l'est.

Example 3.11 (suite de l'exemple 3.9)**.** *$\langle s\rangle_{\rho_0}$, $\langle s\rangle_{\rho_1}$, $\langle s\rangle_{\rho_2}$ et $\langle s\rangle_{\rho_3}$ sont clairement insatisfaisables, et en fait il est facile de voir que c'est le cas pour n'importe quelle instance. Donc s est insatisfaisable.*

En revanche, si nous prenons $s = p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$ alors, comme nous n'avons plus la contrainte, nous avons $\langle s\rangle_{\rho_{-1}} = p_1 \wedge \top \wedge \neg p_0$, qui est une formule satisfaisable. Donc s est satisfaisable.

3.2 Résultats de complétude

3.2.1 Incompleteness

Par la suite notre but va être d'automatiser le raisonnement sur les schémas. Plus précisément nous aimerions disposer d'une procédure prenant en entrée un schéma et indiquant si ce schéma est valide ou non. En pratique nous nous focaliserons plutôt sur l'insatisfaisabilité, sachant que la validité peut se ramener à cette dernière, comme d'habitude en déduction automatique.

Avant d'aller plus loin nous montrons d'ores et déjà qu'il est impossible d'avoir une telle procédure, c.-à-d., en termes logiques, que la logique des schémas n'est pas complète (pour la réfutation, et *a fortiori* pour la validité).

Definition 3.12 (Complétude pour la réfutation)**.** *Une logique est* complète *ssi il existe une procédure de preuve telle que toute formule valide peut être démontrée. Elle est* complète pour la réfutation *ssi $\bot$ est déductible d'une formule insatisfaisable (ou, dans le cas de tableaux, si l'arbre vide est déductible).*

Par conséquent, si une logique est complète pour la réfutation, alors l'ensemble des formules insatisfaisables est récursivement énumérable. Nous montrons maintenant que l'ensemble des schémas insatisfaisables n'est pas récursivement énumérable et, par conséquent, que la logique des schémas est incomplète.

Theorem 3.13 (Incomplétude)**.** *Les schémas ne sont pas complets pour la réfutation.*

Concrètement cela signifie que :

- il n'y a pas d'espoir d'obtenir une procédure décidant la satisfaisabilité et terminant quelle que soit l'entrée.
- il n'y a même pas de notion universelle de *preuve* pour les schémas.

Preuve. Nous démontrons le théorème par réduction au problème de Post : soit $w_1,\ldots,w_k$ et $v_1,\ldots,v_k$ deux suites de mots sur un alphabet fini, une *solution* au problème de Post est une suite d'indices $s_1,\ldots,s_n$ telle que $w_{s_0} \cdot w_{s_1} \cdot \ldots = v_{s_0} \cdot v_{s_1} \cdot \ldots$ Nous appelons $w_{s_0} \cdot w_{s_1} \cdot \ldots$ (ou $v_{s_0} \cdot v_{s_1} \cdot \ldots$) le *témoin*. Il est connu que le problème de Post est indécidable [Pos46]. Nous construisons un schéma s de paramètre n tel que $\langle s\rangle_{\{n\mapsto N\}}$ est satisfaisable ssi le problème de Post admet une solution de longueur N (la solution constitue *grosso modo* le modèle du schéma). Pour cela nous encodons les suites $(w_i)_{i\in[1;k]}$, $(v_i)_{i\in[1;k]}$ et $(s_i)_{i\in[1;n]}$ puis nous définissons un schéma qui vérifie, caractère par caractère, que $(s_i)_{i\in[1;n]}$ est bien une solution. Comme les instances insolubles du problème de Post ne sont pas récursivement énumérables, les schémas insatisfaisables ne le sont pas non plus.

Nous encodons facilement les suites $(w_i)_{i\in[1;k]}$, $(v_i)_{i\in[1;k]}$ et $(s_i)_{i\in[1;n]}$ par des propositions indexées : nous avons la proposition indexée $w_{i,j,k}$ (resp. $v_{i,j,k}$) ssi le i^{th} caractère du j^{th} mot de w (resp. v) est la k^{th} lettre de l'alphabet. Par exemple si l'alphabet est $\{a, b, c\}$, $k = 2$, $w_1 = a$, $w_2 = bb$, $w_3 = ac$, alors w est encodée par le schéma $w_{1,1,1} \wedge w_{1,2,2} \wedge w_{2,2,2} \wedge w_{1,3,1} \wedge w_{2,3,3}$. Pour s, nous avons la proposition indexée $s_{i,j}$ ssi le i^{th} indice de s est j.

Ensuite nous voulons pouvoir vérifier que la solution en est bien une, c.-à-d. que nous avons bien : $w_{s_1} \cdot w_{s_2} \cdot \ldots = v_{s_1} \cdot v_{s_2} \cdot \ldots$. Pour cela nous introduisons une proposition indexée $\mathrm{Check}_{r_1,c_1,r_2,c_2,i}$ qui sera vraie ssi le i^{th} caractère du témoin est égal au ${c_1}^{th}$ caractère de $w_{s_{r_1}}$ et au ${c_2}^{th}$ caractère de $v_{s_{r_2}}$ ("r" pour rang). Ce comportement est spécifié par récurrence : si le premier caractère de w_{s_1} est égal au premier caractère de v_{s_1} alors nous avons $\mathrm{Check}_{1,1,1,1,1}$, puis si nous avons vérifié la séquence jusqu'au rang i (c.-à-d. que nous avons $\mathrm{Check}_{r_1,c_1,r_2,c_2,i}$) alors il faut vérifier l'égalité au rang suivant. Le rang suivant dépend des valeurs de r_1, c_1, r_2 et c_2 :

1. si $c_1 = |w_{s_{r_1}}|$ ($|x|$ désigne la longueur de x) et $c_2 < |v_{s_{r_2}}|$ alors nous passons à $\mathrm{Check}_{r_1+1,0,r_2,c_2+1,i+1}$;
2. si $c_2 = |w_{s_{r_2}}|$ et $c_1 < |w_{s_{r_1}}|$ alors nous passons à $\mathrm{Check}_{r_1,c_1+1,r_2+1,0,i+1}$;
3. si $c_1 = |w_{s_{r_1}}|$ et $c_2 = |v_{s_{r_2}}|$ alors nous passons à $\mathrm{Check}_{r_1,c_1+1,r_2,c_2+1,i+1}$;
4. si $c_1 < |w_{s_{r_1}}|$ et $c_2 < |v_{s_{r_2}}|$ alors nous passons à $\mathrm{Check}_{r_1+1,0,r_2+1,0,i+1}$.

Le cas de base et le cas inductif nécessitent de décider de l'égalité entre le ${c_1}^{th}$ caractère de $w_{s_{r_1}}$ et le ${c_2}^{th}$ caractère de $v_{s_{r_2}}$, ce sera formalisé par $\mathrm{Eq}_{r_1,c_1,r_2,c_2}$:

$$\mathrm{Eq}_{r_1,c_1,r_2,c_2} \Leftrightarrow \bigvee_{i_1=1}^{k} \bigvee_{i_2=1}^{k} (s_{r_1,i_1} \wedge s_{r_2,i_2} \Rightarrow w_{c_1,i_1,j} \wedge v_{c_2,i_2,j}) \tag{3.1}$$

La définition est complétée en parcourant tous les r_1, c_1, r_2, c_2 et j possibles :

$$\bigwedge_{r_1=1}^{n} \bigwedge_{c_1=1}^{m} \bigwedge_{r_2=1}^{n} \bigwedge_{c_2=1}^{m} \bigwedge_{j=1}^{q} (3.1)$$

où $m = \max_{i\in[1;k]}(|w_i|)$ (m est une constante pour une instance donnée du problème de Post) et q est la taille de l'alphabet.

Pour les cas inductifs, il faut aussi pouvoir détecter si le ${c_1}^{th}$ (resp. ${c_2}^{th}$) caractère du mot $w_{s_{r_1}}$ (resp. $v_{s_{r_2}}$) désigne le dernier caractère de ce mot, ce qui est formalisé par la proposition indexée LastW_{r_1,c_1} (resp. LastV_{r_2,c_2}) :

$$\mathrm{LastW}_{r_1,1} \Leftrightarrow \bigvee\{s_{r_1,i} \mid i \in [1;k], |w_i| = 1\}$$

$$\ldots$$

$$\mathrm{LastW}_{r_1,m} \Leftrightarrow \bigvee\{s_{r_1,i} \mid i \in [1;k], |w_i| = m\}$$

que nous devons ensuite encadrer par une itération $\bigwedge_{r_1=1}^{n}$.

On peut maintenant définir précisément les cas inductifs :

1. $\mathrm{Check}_{r_1,c_1,r_2,c_2,i} \wedge \mathrm{LastW}_{r_1,c_1} \wedge \neg\mathrm{LastV}_{r_2,c_2} \wedge \mathrm{Eq}_{r_1,c_1,r_2,c_2}$
   $\Rightarrow \mathrm{Check}_{r_1+1,0,r_2,c_2+1,i+1}$;
2. $\mathrm{Check}_{r_1,c_1,r_2,c_2,i} \wedge \neg\mathrm{LastW}_{r_1,c_1} \wedge \mathrm{LastV}_{r_2,c_2} \wedge \mathrm{Eq}_{r_1,c_1,r_2,c_2}$
   $\Rightarrow \mathrm{Check}_{r_1,c_1+1,r_2+1,0,i+1}$;
3. $\mathrm{Check}_{r_1,c_1,r_2,c_2,i} \wedge \mathrm{LastW}_{r_1,c_1} \wedge \mathrm{LastV}_{r_2,c_2} \wedge \mathrm{Eq}_{r_1,c_1,r_2,c_2}$
   $\Rightarrow \mathrm{Check}_{r_1,c_1+1,r_2,c_2+1,i+1}$;
4. $\mathrm{Check}_{r_1,c_1,r_2,c_2,i} \wedge \neg\mathrm{LastW}_{r_1,c_1} \wedge \neg\mathrm{LastV}_{r_2,c_2} \wedge \mathrm{Eq}_{r_1,c_1,r_2,c_2}$
   $\Rightarrow \mathrm{Check}_{r_1+1,0,r_2+1,0,i+1}$.

chacun étant ensuite encadré par les itérations $\bigwedge_{r_1=1}^{n}\bigwedge_{c_1=1}^{m}\bigwedge_{r_2=1}^{n}\bigwedge_{c_2=1}^{m}\bigwedge_{i=1}^{n}$.

On conclut en exprimant le fait qu'il existe un rang tel que tout a été vérifié et où les mots ont été entièrement "consommés" des deux côtés :

$$\bigvee_{i=1}^{n}\bigvee_{r=1}^{n}\bigwedge_{c_1=1}^{m}\bigwedge_{c_2=1}^{m}(\mathrm{Check}_{r,c_1,r,c_2,i}\wedge\mathrm{LastW}_{r,c_1}\wedge\mathrm{LastV}_{r,c_2})$$

□

Par conséquent, la suite de cette thèse se consacre essentiellement à l'obtention de sous-classes des schémas qui soient complètes pour la réfutation (et donc, comme nous allons le voir dans la section suivante, décidables).

3.2.2 Completeness w.r.t. Satisfiability

Comme nous venons de le voir, il est vain de chercher une procédure de preuve complète pour la réfutation. En revanche il est une propriété intéressante qui est facile à obtenir pour les schémas : la "complétude pour la satisfaisabilité" qui correspond à une "co-"complétude (pour la réfutation) : les formules satisfaisables (au lieu de insatisfaisables) sont récursivement énumérables. Les schémas vérifient cette notion de complétude (ce qui n'est pas le cas, p.ex., de la logique du premier ordre[3]).

Definition 3.14. *Une logique est* complète pour la satisfaisabilité *ssi l'ensemble des formules satisfaisables est récursivement énumérable.*

Il est équivalent de dire qu'il existe une procédure prenant en entrée une formule et terminant quand la formule est satisfaisable.

Theorem 3.15. *Les schémas sont complets pour la satisfaisabilité.*

Preuve. L'ensemble des instances d'un schéma est récursivement énumérable (l'ensemble des paramètres d'un schéma est fini et les entiers relatifs sont récursivement énumérables). Étant donné un environnement et un schéma, l'instance correspondante est calculable. Enfin la satisfaisabilité propositionnelle est décidable. Nous avons donc trivialement une procédure qui termine *quand le schéma est satisfaisable.* □

Nous savons donc d'ores et déjà que la satisfaisabilité des schémas est semi-décidable, et nous avons une première procédure, naïve, qui termine quand le schéma en entrée est satisfaisable. Par conséquent toute classe complète pour la réfutation est décidable : il suffit de lancer, en parallèle, la procédure complète pour la réfutation et la procédure complète pour la satisfaisabilité. La première procédure qui termine permet de conclure.

3.3 Various Notions of Occurrence

Pour définir les systèmes de preuves et la notion de littéral pur, ou simplement pour raisonner sur les schémas, nous avons fréquemment besoin de parler d'*occurrences* d'un littéral dans un schéma. Cependant cette notion n'est pas triviale : il est clair que p_0 et p_n apparaissent dans $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$, en revanche pouvons-nous dire que p_{n-1} apparaît dans ce schéma ? En un sens non puisque, simplement, nous ne "lisons" pas ce littéral lorsque nous lisons le schéma. Mais si nous déroulons le schéma, alors p_{n-1} apparaît bien : $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge (p_{n-1} \Rightarrow p_n) \wedge \neg p_n$ Nous pouvons donc considérer que p_{n-1} apparaît mais de façon implicite. Cependant il est correct de dérouler une itération uniquement si nous savons que cette itération est non vide. Nous distinguons donc deux types d'occurrences implicites : celles qui *peuvent* apparaître parce qu'il existe des valeurs des paramètres pour lesquelles nous pouvons suffisamment dérouler l'itération de sorte à ce qu'elles apparaissent; et celles qui apparaissent *toujours* parce que la contrainte assure que, de toute manière, nous pouvons dérouler l'itération.

Nous obtenons les définitions suivantes :

[3]En revanche, comme nous le verrons au chapitre 10, la logique du premier ordre restreinte aux modèles finis, a cette propriété.

Definition 3.16. *Soit un littéral l et un schéma s.*

- l apparaît positivement *dans s, ssi s contient une occurrence de l qui n'est pas sous une négation.*
- l apparaît librement *dans s ssi il apparaît positivement dans s et aucun de ses indices ne contient de variable liée dans s.*
- l peut apparaître *dans s, noté $l \sqsubset_\diamond s$, ssi tout paramètre de l est un paramètre de s et il existe un environnement tel que $\langle l \rangle_\rho$ apparaît positivement dans $\langle s \rangle_\rho$.*
- l apparaît toujours *dans s, noté $l \sqsubset_\Box s$, ssi tout paramètre de l est un paramètre de s et pour tout environnement ρ satisfaisant c_s, $\langle l \rangle_\rho$ apparaît positivement dans $\langle s \rangle_\rho$.*
- l peut apparaître implicitement *dans s ssi l n'apparaît pas librement dans s mais l peut apparaître dans s.*
- l apparaît toujours implicitement *dans s ssi l n'apparaît pas explicitement dans s mais l apparaît toujours dans s.*

Soit un littéral l' de s tel qu'il existe un environnement ρ pour lequel $\langle l \rangle_\rho = \langle l' \rangle_\rho$ mais $l \neq l'$. Nous disons que l' est une occurrence implicite *de l.*

Example 3.17. *Soit $s \stackrel{def}{=} p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$.*

p_0 et $\neg p_n$ sont les seuls littéraux apparaissant librement dans s. Par conséquent ils peuvent apparaître et apparaissent toujours dans s.

p_1, p_2, etc. peuvent apparaître dans s, tout comme p_{n-1}, p_{n-2}, etc. Comme aucun de ces littéraux n'apparaît explicitement dans s, ils peuvent apparaître implicitement dans s.

p_1 n'apparait pas toujours car, vue la contrainte $n \geq 0$ nous pouvons avoir l'environnement $\{n \mapsto 0\}$. En revanche si nous modifions la contrainte en posant $s' \stackrel{def}{=} p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 1$, alors p_1 apparaît toujours dans s (de même que p_{n-1}). Comme il n'apparaît pas explicitement, il apparaît toujours implicitement.

Il est évident qu'il est décidable de savoir si un littéral apparaît librement dans un schéma. Nous montrons maintenant que $\sqsubset_\diamond$ et $\sqsubset_\Box$ sont des relations décidables. Soit un littéral l de la forme $p_{e_1,\ldots,e_n}$ (resp. $\neg p_{e_1,\ldots,e_n}$) dont nous voulons savoir s'il peut apparaître ou apparaît toujours dans un schéma s.

La formule arithmétique $l \sqsubset m_s$ est définie récursivement par :

$$l \sqsubset l' \stackrel{\text{def}}{=} \begin{cases} e_1 = f_1 \wedge \ldots \wedge e_n = f_n & \text{si } l' = p_{f_1,\ldots,f_n} \text{ (resp. } \neg p_{f_1,\ldots,f_n}) \\ \bot & \text{sinon} \end{cases}$$

$$l \sqsubset m_1 \wedge m_2 \stackrel{\text{def}}{=} l \sqsubset m_1 \vee m_2 \stackrel{\text{def}}{=} (l \sqsubset m_1) \vee (l \sqsubset m_2)$$

$$l \sqsubset \bigwedge_{i|\delta} m \stackrel{\text{def}}{=} l \sqsubset \bigvee_{i|\delta} m \stackrel{\text{def}}{=} \exists i(\delta \wedge l \sqsubset m)$$

Par exemple, $p_n \sqsubset p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$ est la formule :

$$n = 0 \vee \exists i(0 \leq i \wedge i \leq n-1 \wedge (n = i \vee n = i+1)) \vee \bot$$

et $\neg p_n \sqsubset p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$ est la formule :

$$\bot \vee \exists i(0 \leq i \wedge i \leq n-1 \wedge (\bot \vee \bot)) \vee n = n$$

Proposition 3.18. *Soient un littéral l et un schéma s. On a :*

- *$l \sqsubset_\Box s$ ssi $\forall n_1, \ldots, n_n(c_s \Rightarrow l \sqsubset m_s)$ est valide*
- *$l \sqsubset_\diamond s$ ssi $\exists n_1, \ldots, n_n(c_s \wedge l \sqsubset m_s)$ est valide.*

Preuve. Soit un environnement ρ satisfaisant c_s. Supposons que $\langle l \rangle_\rho$ apparaît positivement dans $\langle s \rangle_\rho$. D'après la définition 3.8, c'est équivalent à dire qu'il existe $I_1, \ldots, I_n$ et un littéral l' t.q.

$$\langle l \rangle_\rho = \langle l' \rangle_{\rho \oslash \{i_1 \mapsto I_1\} \oslash \ldots \oslash \{i_n \mapsto I_n\}}$$

où les $i_1, \ldots, i_n$ sont les variables liées par les itérations contenant l' et les $I_1, \ldots, I_n$ satisfont les domaines correspondants. Il est facile de constater que cette dernière assertion correspond précisément à la définition de $l \sqsubset m_s$. □

Corollary 3.19. *Les relations $\sqsubset_\Box$ et $\sqsubset_\Diamond$ sont décidables.*

Enfin nous avons aussi le résultat suivant : l'ensemble $\{l \mid l \sqsubset_\Box s\}$ est *fini* pour un schéma s donné. Ainsi, non seulement $\sqsubset_\Box$ est décidable, mais en plus l'énumération de tous les littéraux $l \sqsubset_\Box s$ termine.

Proposition 3.20. *Soit un schéma s. L'ensemble $\{l \mid l \sqsubset_\Box s\}$ est fini.*

Preuve. Soit un environnement ρ quelconque. Pour tout littéral l, si $l \sqsubset_\Box s$, alors $\langle l \rangle_\rho$ apparaît positivement dans $\langle s \rangle_\rho$. Donc $\{l \mid l \sqsubset_\Box s\}$ est inclus dans $\{l \mid \langle l \rangle_\rho$ apparaît positivement dans $\langle s \rangle_\rho\}$ et il est clair que cet ensemble est fini car $\langle s \rangle_\rho$ est une formule propositionnelle. Par conséquent elle contient une quantité finie de littéraux. □

Example 3.21. *L'ensemble $\{l \mid l \sqsubset_\Box p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0\}$ est égal à $\{p_0, \neg p_n\}$. L'ensemble $\{l \mid l \sqsubset_\Box p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 1\}$ est égal à $\{p_0, p_1, \neg p_n\}$.*

3.4 Domain Normal Form

Initialement (dans [ACP09b]), les schémas n'avaient pour domaines que des formules de la forme $e \leq i \wedge i \leq f$, ce qui avait pour avantage d'être intuitif, de correspondre à la plupart des schémas rencontrés en pratique, et d'être suffisamment simple pour assurer un traitement théorique aisé. Cependant, pour définir Dpll* (chapitre 8) nous avons eu besoin de généraliser ces domaines à n'importe quelle formule arithmétique. Cette généralisation a un prix : elle complique énormément les systèmes de preuves. Par conséquent nous définissons des restrictions et des formes normales sur les domaines afin de faciliter leur traitement par ces systèmes. Nous prouvons, en annexe 3.C que la plupart des schémas peuvent être ramenés à des schémas équivalents dont les domaines suivent les restrictions présentées ici (en tout cas, tous les schémas rencontrés en pratique le peuvent).

Note. Ces résultats sont mis en annexe pour des raisons de lisibilité et parce qu'ils présentent un intérêt mineur par rapport à notre problématique générale. Cependant ils seront *nécessaires* à l'analyse de Dpll*, quoique cette analyse n'utilise ces résultats que dans des cas très particuliers, comme nous le verrons alors.

Par définition, toutes les itérations d'un schéma ont des domaines encadrant leurs itérateurs. Cependant l'encadrement est sémantique, mais n'apparaît pas forcément de façon syntaxique dans le domaine lui-même. Le but principal des formes normales suivantes est d'imposer que cet encadrement soit explicite, c.-à-d. qu'une formule $e \leq i \wedge i \leq f$ apparaisse dans le domaine (au lieu d'être seulement une conséquence logique de celui-ci). De plus nous imposons qu'une seule formule de cette forme puisse apparaître.

Definition 3.22. *Une formule* encadre fortement *i ssi elle a la forme :*

$$e \leq i \wedge i \leq f$$

où e et f sont des expressions arithmétiques ne contenant pas i.

Une itération est *fortement encadrée* ssi son domaine encadre fortement son itérateur. Un motif est *fortement encadré* ssi toutes ses itérations le sont. Un schéma est *fortement encadré* ssi son motif l'est. Les schémas fortement encadrés correspondent précisément aux schémas présentés dans [ACP09b]. La procédure Stab (chapitre 4) ne prendra en entrée que des schémas fortement encadrés.

En revanche, pour Dpll*, nous devons relaxer un peu cette définition, ce qui nous amène à introduire les schémas syntaxiquement encadrés :

Definition 3.23. *Une formule* encadre syntaxiquement *i ssi elle a la forme :*

$$e \leq i \wedge i \leq f \wedge \phi$$

où e et f sont des expressions arithmétiques ne contenant pas i et ϕ est une formule sans disjonction ni quantificateur et ne contenant pas i.

Une itération est *syntaxiquement encadrée* ssi son domaine encadre syntaxiquement son itérateur. Un motif est *syntaxiquement encadré* ssi toutes ses itérations le sont. Un schéma est *syntaxiquement encadré* ssi son motif l'est. Tout schéma fortement encadré est syntaxiquement encadré.

La propriété suivante est la propriété principale des schémas syntaxiquement encadrés et celle qui fait tout leur intérêt :

Proposition 3.24. *Si une formule arithmétique ϕ est syntaxiquement encadrée alors il existe une expression e tel que $(\forall \vec{n} \exists i \phi) \Rightarrow (\forall \vec{n} \phi[e/i])$, où $\vec{n}$ est l'ensemble des variables libres de ϕ, différentes de i.*

Preuve. Par définition, ϕ a la forme $e \leq i \wedge i \leq f \wedge \psi$ où ψ *ne contient pas* i. Donc si $\forall \vec{n} \exists i \phi$ est valide, il est clair que nous avons $\phi[e/i]$ et aussi $\phi[f/i]$, d'où le résultat. □

C'est là tout l'intérêt de ne considérer que des itérations syntaxiquement encadrées. En effet, si ce n'était pas le cas une itération pourrait avoir pour domaine $(n_1 \leq i \wedge i \leq n_2) \vee (n_3 \leq i \wedge i \leq n_4)$[4]. Dans ce cas, savoir que l'itération est non vide ne permet pas pour autant de dire quel rang rend l'itération non vide : ce peut être aussi bien n_1 que n_2, n_3, ou n_4 en fonction des interprétations. Ceci pose un problème lorsque nous devons "déplier" une itération, c.-à-d. sortir un rang de l'itération : il n'est pas correct de sortir un rang si nous ne pouvons pas garantir qu'il apparaît effectivement dans l'itération.

Notons aussi qu'un domaine syntaxiquement ou fortement encadré ne peut pas, par exemple, être de la forme $1 \leq k.i \wedge k.i \leq n$. Ceci est justifié par le même argument : si nous savons que $n \geq k$, alors nous savons que le domaine est non vide, mais comment savoir quel rang existe ? Intuitivement nous voudrions prendre $\lceil n/k \rceil$, mais nous n'avons pas de symbole de division. Il est possible d'introduire un symbole de division mais cela pose d'autres problèmes. Il est plus simple et peu restrictif de simplement interdire ce type de domaine.

Annexe

3.A Variations sur le langage

Nous étudions ici diverses simplifications ou extensions qui peuvent être apportées au langage.

3.A.1 Suppression de la contrainte

Par exemple, nous pouvons nous débarrasser de la contrainte :

Proposition 3.25. *Pour tout motif m et toute contrainte c :*

$$m \,\&\, c \equiv \left(\bigvee_{i | c \wedge i = 0} m \right) \,\&\, \top$$

où i est une nouvelle variable (c.-à-d. n'apparaissant pas dans s).

Ainsi pour tout schéma il existe un schéma équivalent dont la contrainte est $\top$. Nous pouvons donc supprimer la contrainte de la définition de schéma, c.-à-d. restreindre la notion de schéma à celle de motif.

[4] En fait il est impossible d'avoir un domaine de cette forme car il n'est pas encadré : en effet nous ne pouvons pas trouver d'expressions e et f telles que $\delta \models e \leq i \wedge i \leq f$ (en notant δ le domaine défini ci-dessus). Mais nous pouvons avoir, p.ex., $\delta \wedge n_5 \leq n_1 \wedge n_5 \leq n_3 \wedge n_2 \leq n_6 \wedge n_4 \leq n_6$ qui est bien encadré mais présente les mêmes problèmes que δ.

Preuve. Pour tout environnement ρ :

$$\begin{aligned}
\langle \bigvee_{i|c_s \wedge i=0} m_s \rangle_\rho &= \bigvee \{ \langle m_s \rangle_{\rho \circledcirc \{i \mapsto I\}} \mid I \in \mathbb{Z} \text{ tel que } \rho \circledcirc \{i \mapsto I\} \models c_s \wedge i = 0 \} \\
&= \bigvee \{ \langle m_s \rangle_{\rho \circledcirc \{i \mapsto 0\}} \mid \rho \circledcirc \{i \mapsto 0\} \models c_s \} \\
&= \bigvee \{ \langle m_s \rangle_{\rho \circledcirc \{i \mapsto 0\}} \mid \rho \models c_s \} \qquad (i \text{ n'apparaît pas dans } c_s) \\
&= \bigvee \{ \langle m_s \rangle_\rho \mid \rho \models c_s \} \qquad (i \text{ n'apparaît pas dans } m_s) \\
&= \begin{cases} \langle m_s \rangle_\rho & \text{si } \rho \models c_s \\ \bot & \text{sinon} \end{cases} \\
&= \langle m_s \rangle_\rho
\end{aligned}$$

□

Ainsi la contrainte peut en fait être codée dans le motif. Nous pouvons obtenir le même résultat grâce à la proposition suivante ce qui a l'avantage de ne pas augmenter le niveau d'imbrication des itérations :

Proposition 3.26. *Pour tout motif m et toute contrainte c :*

$$m \,\&\, c \equiv (m \wedge \bigvee_{i|c \wedge i=0} \top) \,\&\, \top$$

où i est une nouvelle variable (c.-à-d. n'apparaissant pas dans s).

Pour les contraintes de la forme $e \leq f$, une autre traduction est possible :

Proposition 3.27. *Pour tout motif m et toute contrainte c :*

$$m \,\&\, (c \wedge e \leq f) \equiv \left(m \wedge \bigvee_{i=e}^{f} \top \right) \,\&\, c$$

Ce qui donne donc une autre façon d'éliminer la contrainte si cette dernière est une conjonction d'inégalités.

3.A.2 Itérations généralisées

Par ailleurs, il est envisageable d'étendre le langage de sorte à pouvoir itérer sur d'autres fonctions booléennes que $\wedge$ ou $\vee$. Par exemple en considérant un ensemble F de *symboles de fonctions itérées* tel que tout symbole $f \in F$ est muni d'une définition (inductive) de la forme suivante :

$$f(i) \stackrel{\text{def}}{=} \begin{cases} \phi_0 & \text{si } i = 0 \\ \phi_i[f(i-1)] & \text{sinon} \end{cases}$$

où ϕ_0 et ϕ_i sont des formules propositionnelles dont les atomes sont des propositions indexées pouvant dépendre de i (nous ne considérons ici que le cas simple où les fonctions ne peuvent pas faire appel à d'autres fonctions). Nous pouvons ensuite ajouter au langage des atomes de la forme f_e pour $f \in F$ et une expression arithmétique e. Puis pour tout environnement ρ :

$$\langle f_e \rangle_\rho \stackrel{\text{def}}{=} f(\rho(e))$$

Example 3.28. *Nous pouvons représenter une "équivalence itérée" de la façon suivante :*

$$f(i) \stackrel{def}{=} \begin{cases} p_0 & si\ i = 0 \\ p_i \Leftrightarrow f(i-1) & sinon \end{cases}$$

Alors f_n représente, de façon informelle, le schéma $\Leftrightarrow_{i=0}^{n} p_i$.

Il est facile de voir qu'une conjonction itérée peut être spécifiée par de telles définitions :

Proposition 3.29. *Soit m un motif ne contenant pas d'itération et n un paramètre de m, alors :*

$$\bigwedge_{i=1}^{n} m \equiv f_n$$

où f est définie de la façon suivante :

$$f(i) \stackrel{def}{=} \begin{cases} \top & si\ i = 0 \\ m \wedge f(i-1) & sinon \end{cases}$$

Preuve. Pour tout environnement ρ : $\langle \bigwedge_{i=1}^{n} m \rangle_\rho = \langle f_n \rangle_\rho$, par récurrence sur $\rho(n)$. □

En revanche, il est plus surprenant d'avoir le résultat suivant :

Proposition 3.30. *Soit f une fonction et ϕ_0, ϕ_i tels que :*

$$f(i) \stackrel{def}{=} \begin{cases} \phi_0 & si\ i = 0 \\ \phi_i[f(i-1)] & sinon \end{cases}$$

Alors f_n est satisfaisable ssi $p_n \wedge (p_0 \Leftrightarrow \phi_0) \wedge \bigwedge_{i=1}^{n} p_i \Leftrightarrow \phi_i[p_{i-1}]$ l'est, où p est un symbole n'apparaissant pas dans ϕ_0 et ϕ_i.

Preuve. Soit un modèle $\mathfrak{M}$ de f_n. Soit l'interprétation $\mathfrak{N}$ définie par :

$$\mathfrak{N}_{\text{arith}} \stackrel{\text{def}}{=} \mathfrak{M}_{\text{arith}}$$

$$\mathfrak{N}_{\text{prop}}(\langle q_e \rangle_{\mathfrak{M}_{\text{arith}}}) \stackrel{\text{def}}{=} \begin{cases} \mathfrak{M}_{\text{prop}}(\langle q_e \rangle_{\mathfrak{M}_{\text{arith}}}) & \text{si } q \neq p \\ \mathfrak{M}_{\text{prop}}(\langle f_e \rangle_{\mathfrak{M}_{\text{arith}}}) & \text{si } q = p \end{cases}$$

Comme ni ϕ_0 ni ϕ_i ne contiennent p, il est facile de montrer que $\mathfrak{N}$ est un modèle de $p_n \wedge (p_0 \Leftrightarrow \phi_0) \wedge \bigwedge_{i=1}^{n} p_i \Leftrightarrow \phi_i[p_{i-1}]$ par récurrence sur $\mathfrak{N}_{\text{arith}}(n)$. □

Autrement dit, il est possible d'exprimer toutes les fonctions itérées avec uniquement la conjonction itérée. Conséquemment, nous pouvons même nous dispenser de la disjonction itérée.

Ce résultat justifie le fait que nous nous restreignons dans cette thèse à des conjonctions et disjonctions itérées, plus simples à manipuler et à écrire et suffisamment générales.

3.B Autres résultats d'incomplétude

Dans [ACP10d], nous donnons deux autres preuves d'incomplétude pour des *sous-classes strictes* des schémas : la classe *homothétique* et la classe *à translation non bornée*.

Definition 3.31 (Classe homothétique)**.** *Un schéma s appartient à la* classe homothétique *ssi il vérifie les propriétés suivantes :*

- *s a au plus un paramètre n*
- *Chaque itération de s est de la forme $\bigwedge_{i=1}^{n} s'$ ou $\bigvee_{i=1}^{n} s'$ où s' ne contient pas d'itération et chacun de ses atomes est de la forme $p_{q_1 i+q_2}$ avec $q_1 \in \{1, 2\}$ et $q_2 \in \{-1, 0, 1\}$.*
- *Les atomes apparaissant en dehors des itérations sont de la forme p_0 ou p_n.*

Definition 3.32 (Classe à translation non bornée)**.** *Un schéma s appartient à la* classe à translation non bornée *ssi il vérifie les propriétés suivantes :*

- *s a au plus deux paramètres n et l*
- *Chaque itération de s est de la forme $\bigwedge_{i=1}^{n} s'$ ou $\bigvee_{i=1}^{n} s'$ où s' ne contient pas d'itération et chacun de ses atomes est de la forme $p_{q_1 i+q_2+q_3 l}$ avec $q_1, q_3 \in \{0, 1\}$ et $q_2 \in \{-1, 0, 1\}$.*

- *Les atomes apparaissant en dehors des itérations sont de la forme p_0 ou p_n.*

Dans les deux cas, les schémas sont très restreints : seules des propositions monadiques sont autorisées, tous les schémas doivent avoir les mêmes bornes pour les itérations et il n'est pas possible d'imbriquer les itérations. Dans le premier cas, il est possible de multiplier la variable liée par 2. Dans le second cas, il est possible d'ajouter un paramètre à une variable liée.

Nous avons alors le théorème suivant :

Theorem 3.33. *(i) L'ensemble des schémas insatisfaisables de la classe homothétique n'est pas récursivement énumérable.*

(ii) L'ensemble des schémas insatisfaisables de la classe à translation bornée n'est pas récursivement énumérable.

Preuve. Voir [ACP10d]. La preuve est encore par réduction au problème de Post, mais le codage utilisé est beaucoup plus complexe que dans le cas des schémas non monadiques (surtout pour le point (i)). □

Dans le premier cas, le point important est la possibilité de multiplier une variable liée par 2. Dans le second, c'est la possibilité d'ajouter un paramètre, c.-à-d. une valeur non bornée. Ainsi nous constatons que même en restreignant beaucoup le langage, l'indécidabilité persiste.

Ces résultats montrent que les classes décidables de schémas (schémas réguliers et réguliers imbriqués, voir chapitres 6 et 9) que nous considérons dans cette thèse sont extrêmement difficiles à étendre et peuvent donc être considérés comme "canoniques" (du point de vue du compromis entre le pouvoir d'expression et la simplicité des définitions).

3.C Domain Normal Form

Nous détaillons ici la section 3.4 : nous montrons que la plupart des schémas peuvent être ramenés à des schémas équivalents syntaxiquement ou fortement encadrés.

Nous définissons les notions de schéma *quasi-syntaxiquement encadré* et *quasi-fortement encadré*. Puis nous donnons un algorithme permettant de réduire tout schéma à un schéma équivalent quasi-syntaxiquement encadré. De même pour les schémas quasi-fortement encadrés. Ainsi si un système requiert en entrée des schémas sous ces formes normales, il suffit d'appliquer l'algorithme à ces schémas. Les systèmes de preuve pourront ainsi être définis *modulo une forme normale donnée*.

Finalement, nous donnons un critère syntaxique simple permettant d'assurer qu'un schéma quasi-syntaxiquement (resp. fortement) encadré est bien syntaxiquement (resp. fortement) encadré. Ceci prouve qu'il n'est pas si restrictif de ne considérer que des schémas syntaxiquement ou fortement encadrés.

3.C.1 Domaines quasi-syntaxiquement encadrés

Definition 3.34. *Une formule arithmétique* encadre quasi-syntaxiquement *une variable i ssi elle est de la forme :*

$$e \leq k.i \wedge l.i \leq f \wedge \phi \wedge \psi$$

où :

- $k, l \in \mathbb{N}$;
- e *et* f *sont des expressions arithmétiques ne contenant pas* i;
- ϕ *est une conjonction de formules de la forme* $k|e'$ *ou* $k\not|e'$, *où* e' *contient* i;
- ψ *est une formule sans disjonction ni quantificateur ne contenant pas* i.

Example 3.35. *Les formules suivantes encadrent quasi-syntaxiquement i :*

- $1 \leq i \wedge i \leq n$
- $1 \leq 2i \wedge 3i \leq n$
- $1 \leq i \wedge i \leq n \wedge 2|i$

- $1 \leq i \wedge i \leq n \wedge 1 \leq n$

En revanche, ce n'est pas le cas de :

- $(1 \leq i \vee 2 \leq i) \wedge i \leq n$ *(cette contrainte encadre bien i, mais pas quasi-syntaxiquement) car cette formule contient une disjonction;*
- $1 \leq i \wedge i \leq n_1 \wedge i \leq n_2 \wedge n_1 \leq n \wedge n_2 \leq n$ *car i apparaît dans deux atomes de la forme $i \leq \ldots$;*
- *et* $1 \leq i \wedge i \leq n \wedge (i \lhd j \vee i > j)$.

Une itération est *quasi-syntaxiquement encadrée* ssi son domaine encadre quasi-syntaxiquement son itérateur. Un motif est *quasi-syntaxiquement encadré* ssi toutes ses itérations le sont. Un schéma est *quasi-syntaxiquement encadré* ssi son motif l'est. Nous montrons maintenant comment ramener tout schéma a un schéma quasi-syntaxiquement encadré équivalent.

Definition 3.36 ($\mathrm{R}_{\mathrm{syn}}$)**.** *Soit* $\mathrm{R}_{\mathrm{syn}}$ *le système de réécriture de motifs suivant :*

- Forme normale arithmétique *(appliquée uniquement si δ n'est pas en f.n.a.) :*

$$\Pi_{i|\delta}\, m \to \Pi_{i|\delta'}\, m$$

  où δ' est en forme normale arithmétique (définition p.12) et $\delta' \equiv \delta$

- Élimination des disjonctions *:*

$$\Pi_{i|\delta_1 \vee \delta_2}\, m \to (\Pi_{i|\delta_1}\, m)\, \amalg\, (\Pi_{i|\delta_2}\, m)$$

  où $(\Pi, \amalg) \in \{(\bigwedge, \wedge), (\bigvee, \vee)\}$

- Linéarisation *:*

$$\Pi_{i|k.i \leq e \wedge l.i \leq f \wedge \phi'}\, m \to (\Pi_{i|k.i \leq e \wedge l.e \leq k.f \wedge \phi'}\, m)\, \amalg\, (\Pi_{i|l.i \leq f \wedge k.f < l.e \wedge \phi'}\, m)$$

  où $k, l \in \mathbb{N}$, e et f sont des expressions ne contenant pas i.

Nous appliquons les règles dans l'ordre où nous les avons données.

Example 3.37. *Le schéma :*

$$\bigwedge_{i|\exists j(1 \leq j \wedge j \leq i \wedge i \leq n)} p_i$$

n'est pas quasi-syntaxiquement encadré. Après application de la règle Forme normale arithmétique*, nous obtenons :*

$$\bigwedge_{i|1 \leq i \wedge i \leq n} p_i$$

qui est bien quasi-syntaxiquement encadré (et même syntaxiquement encadré).

Le schéma :

$$\bigwedge_{i|(1 \leq i \vee 2 \leq i) \wedge i \leq n} p_i$$

n'est pas quasi-syntaxiquement encadré (à cause de la disjonction). Après application de la règle Élimination des disjonctions*, nous obtenons :*

$$\bigwedge_{i|1 \leq i \wedge i \leq n} p_i \wedge \bigwedge_{i|2 \leq i \wedge i \leq n} p_i$$

qui est bien quasi-syntaxiquement encadré (et même, encore une fois, syntaxiquement encadré).

Le schéma :

$$\bigwedge_{i|1 \leq i \wedge i \leq n_1 \wedge i \leq n_2} p_i$$

n'est pas quasi-syntaxiquement encadré. Mais après application de la règle Linéarisation, *nous obtenons le schéma syntaxiquement encadré :*

$$\bigwedge_{i|1\leq i\wedge i\leq n_2\wedge n_2\leq n_1} p_i \wedge \bigwedge_{i|1\leq i\wedge i\leq n_1\wedge n_1\leq n_2} p_i$$

Le schéma :

$$\bigwedge_{i|1\leq i\wedge 2i\leq n_1\wedge i\leq n_2} p_i$$

n'est pas quasi-syntaxiquement encadré. Mais après application de la règle Linéarisation, *nous obtenons le schéma :*

$$\bigwedge_{i|1\leq i\wedge 2i\leq n_2\wedge n_2\leq 2n_1} p_i \wedge \bigwedge_{i|1\leq i\wedge i\leq n_1\wedge 2n_1\lhd n_2} p_i$$

qui est quasi-syntaxiquement encadré (mais pas syntaxiquement encadré).

Nous pouvons démontrer que l'ensemble de règles ci-dessus termine, que toute forme normale obtenue est équivalente au motif original et est bien quasi-syntaxiquement encadrée. Cependant, il manque à ce système une propriété utile par la suite : lorsqu'une itération est "divisée" en deux par une règle (comme c'est le cas de Élimination des disjonctions et Linéarisation), nous aimerions que les deux nouvelles itérations aient des domaines disjoints :

Definition 3.38. *Deux domaines δ_1 et δ_2 sont* disjoints *ssi $\delta_1 \wedge \delta_2$ est insatisfaisable.*

Il est facile de voir que Linéarisation a cette propriété car si nous avons $l.e \leq k.f$ alors nous n'avons pas $k.f < l.e$. En revanche Élimination des disjonctions n'a pas cette propriété : il suffit de considérer, p.ex., $\bigwedge_{i|\delta\vee\delta} p_i$ qui est réécrit en $(\bigwedge_{i|\delta} p_i) \wedge (\bigwedge_{i|\delta} p_i)$. Par conséquent nous remplaçons la règle Élimination des disjonctions par la règle suivante :

Definition 3.39 (Élimination des disjonctions bis)**.**

$$\begin{aligned}\Pi_{i|(l_1\wedge\ldots\wedge l_k)\vee\delta\vee\phi}\, m \rightarrow (\Pi_{i|l_1\wedge\ldots\wedge l_k}\, m)\ &\Pi\, (\Pi_{i|(\neg l_1\wedge\delta)\vee\phi}\, m)\\ &\Pi\, (\Pi_{i|(l_1\wedge\neg l_2\wedge\delta)\vee\phi}\, m)\\ &\Pi\, (\Pi_{i|(l_1\wedge l_2\wedge\neg l_3\wedge\delta)\vee\phi}\, m)\\ &\ldots\\ &\Pi\, (\Pi_{i|(l_1\wedge\ldots\wedge l_{k-1}\wedge\neg l_k\wedge\delta)\vee\phi}\, m)\end{aligned}$$

où $l_1, \ldots, l_k$ sont des littéraux, $k \geq 1$, δ est une clause conjonctive différente de $\bot$ et ϕ est une formule quelconque (possiblement $\top$).

Nous supposons aussi que les négations sont éliminées dans la foulée en suivant les règles données en 2.2. Cette règle a bien la propriété que les domaines de ses itérations sont deux à deux disjoints. À partir de maintenant, c'est cette règle que nous appellerons Élimination des disjonctions, et $\mathrm{R}_{\mathrm{syn}}$ dénotera l'ensemble de règles précédent avec la nouvelle Élimination des disjonctions. Comme nous allons le voir, ce système a bien toutes les propriétés voulues : il transforme tout schéma en un schéma quasi-syntaxiquement encadré équivalent.

Proposition 3.40. $\mathrm{R}_{\mathrm{syn}}$ *termine.*

Preuve. Forme normale arithmétique termine car le nombre d'itérations dont le domaine n'est pas en f.n.a. décroît strictement. Élimination des disjonctions diminue strictement le nombre de disjonctions dans le domaine d'une itération, donc elle termine. De plus elle préserve la f.n.a. donc elle ne suscite pas de nouvelle application de Forme normale arithmétique. Pour Linéarisation, chaque itération $\Pi_{i|\delta}\, m$ est remplacée par deux itérations ayant chacune un nombre inférieur d'occurrences de i. De plus Linéarisation n'ajoute pas de disjonction et préserve la forme normale arithmétique, donc ne suscite pas d'application supplémentaire des règles précédentes. □

Il existe donc une forme normale (non nécessairement unique) par $\mathrm{R}_{\mathrm{syn}}$ pour chaque motif. Les trois propositions suivantes assurent que toute forme normale par $\mathrm{R}_{\mathrm{syn}}$ est bien un motif syntaxiquement encadré et équivalent au motif original.

Proposition 3.41. *Toute forme normale d'un motif par* $\mathrm{R}_{\mathrm{syn}}$ *est un motif.*

Preuve. Il suffit de vérifier que toutes les itérations sont encadrées. C'est évident pour Forme normale arithmétique car les domaines sont équivalents avant et après la réécriture. Pour Élimination des disjonctions, comme $(l_1 \wedge \ldots \wedge l_k) \vee \delta \vee \phi$ encadre i, alors δ et ϕ encadrent i donc c'est évident. Pour Linéarisation, comme l'itération est encadrée avant la réécriture, ϕ' entraîne $e \lhd i$ pour une certaine expression e. De plus les domaines des deux nouvelles itérations contiennent chacun ϕ' et une formule de la forme $k.i \leq e$ donc ils encadrent bien i. □

Proposition 3.42. *Toute forme normale par* $\mathrm{R}_{\mathrm{syn}}$ *est un motif quasi-syntaxiquement encadré.*

Preuve. Les seules disjonctions apparaissent à la racine par irréductibilité par rapport à Forme normale arithmétique. Par conséquent s'il existait une telle disjonction Élimination des disjonctions pourrait s'appliquer ce qui est impossible par irréductibilité. Donc tout domaine est une conjonction, conformément à la forme imposée par la définition d'encadrement quasi-syntaxique. Si un domaine contenait une formule de la forme $k.i \leq e \wedge l.i \leq f$ alors Linéarisation s'appliquerait, impossible par irréductibilité. Donc tout domaine a au plus une inégalité de la forme $k.i \leq e$ et une inégalité de la forme $e \leq k.i$. Finalement tout domaine a exactement une inégalité de chacune de ces formes car toute itération était encadrée avant la réécriture et la proposition précédente assure que c'est toujours le cas après. □

Proposition 3.43. *Soit un schéma* s*. Notons* m *une forme normale de* m_s *par* $\mathrm{R}_{\mathrm{syn}}$*. Alors* $m \,\&\, c_s$ *est équivalent à* s*.*

Preuve. Forme normale arithmétique préserve les modèles puisque les domaines sont équivalents. Pour Élimination des disjonctions et Linéarisation, il est facile de voir que les instances du membre gauche et du membre droit sont équivalentes dans tout environnement. □

Theorem 3.44. *Pour tout schéma* s *il existe un schéma quasi-syntaxiquement encadré* s' *tel que* $s' \equiv s$*. De plus* s' *est calculable.*

Preuve. Conséquence des propositions précédentes. □

3.C.2 Domaines quasi-fortement encadrés

Nous restreignons encore les schémas quasi-syntaxiquement encadrés de sorte à ce que, non seulement l'encadrement soit explicitement présent dans la formule, mais qu'en plus le domaine ne spécifie que cet encadrement, c.-à-d. qu'il n'exprime pas d'inéquations sur les variables autres que son itérateur.

Definition 3.45. *Une formule arithmétique* encadre quasi-fortement *une variable* i *ssi elle est de la forme :*

$$e \leq k.i \wedge l.i \leq f \wedge \phi$$

où :

- $k, l \in \mathbb{N}$*;*
- e *et* f *sont des expressions arithmétiques ne contenant pas* i*;*
- ϕ *est une conjonction de formules de la forme* $k|e'$ *ou* $k\not|e'$*.*

Une itération est quasi-fortement encadrée *ssi son domaine encadre quasi-fortement son itérateur. Un motif est* quasi-fortement encadré *ssi toutes ses itérations le sont. Un schéma est* quasi-fortement encadré *ssi son motif l'est.*

Nous montrons maintenant comment nous pouvons ramener tout schéma a un schéma quasi-fortement encadré équivalent. D'après les résultats précédents, il suffit de montrer ce résultat pour les schémas *quasi-syntaxiquement* encadrés uniquement.

Soit $\mathrm{R}_{\mathrm{fort}}$ la règle de réécriture suivante :

$$\Pi_{i|\delta \wedge e \leq f}\, m \to \left[\left(\bigvee_{i=e}^{f} \top\right) \Rightarrow \Pi_{i|\delta}\, m\right] \wedge \left[\left(\bigwedge_{i=e}^{f} \bot\right) \Rightarrow \epsilon\right]$$

où $(\Pi, \epsilon) \in \{(\bigwedge, \top), (\bigvee, \bot)\}$ et si e et f ne contiennent pas i. Nous appliquons $\mathrm{R}_{\mathrm{fort}}$ en commençant par les itérations les plus profondes.

Proposition 3.46. $\mathrm{R}_{\mathrm{fort}}$ *termine.*

Preuve. Le nombre de conjonctions apparaissant dans les domaines de toutes les itérations diminue strictement (vrai uniquement parce que nous appliquons $\mathrm{R}_{\mathrm{fort}}$ en commençant par les itérations les plus profondes). □

Proposition 3.47. *Toute forme normale par* $\mathrm{R}_{\mathrm{fort}}$ *d'un motif quasi-syntaxiquement encadré est quasi-fortement encadrée.*

Preuve. Tout d'abord, il est facile de voir que la règle préserve l'encadrement quasi-syntaxique. Ensuite si un motif est quasi-syntaxiquement encadré mais pas quasi-fortement encadré, la règle s'applique. Comme le motif est irréductible, il est donc quasi-fortement encadré. □

Proposition 3.48. *Soit un schéma* s*. Notons* m *une forme normale de* m_s *par* $\mathrm{R}_{\mathrm{fort}}$*. Alors* $m \,\&\, c_s$ *est équivalent à* s*.*

Preuve. Il suffit de constater que $\bigvee_{i=e}^{f} \top$ est vrai ssi $e \leq f$ et $\bigwedge_{i=e}^{f} \bot$ est vrai ssi $e > f$. Il est ensuite facile de voir que les instances du membre gauche et du membre droit sont équivalentes dans tout environnement. □

Theorem 3.49. *Pour tout schéma* s *il existe un schéma* s' *quasi-fortement encadré tel que* $s' \equiv s$*. De plus* s' *est calculable.*

Preuve. Conséquence du théorème 3.44 et des propositions précédentes. □

3.C.3 Domaines linéaires

Les deux formes normales précédentes n'induisent aucun restriction sur le schéma d'origine puisque nous avons vu que nous pouvions ramener tout schéma à de telles formes normales. Nous introduisons maintenant la notion de *domaine linéaire*. Combiné avec le quasi-encadrement syntaxique, nous retrouvons la notion d'encadrement syntaxique. De même pour le quasi-encadrement fort.

Definition 3.50 (Domaine linéaire)**.** *Une variable* i apparaît linéairement *dans une formule* ϕ *ssi* i *n'apparaît qu'une fois dans chaque atome.*

Une itération est linéairement encadrée *ssi son itérateur apparaît linéairement dans son domaine. Un schéma est* linéairement encadré *ssi toutes ses itérations le sont.*

Par exemple i apparaît linéairement dans $i \leq n$ et $1 \leq i \wedge i \leq n$ mais pas dans $2i \leq n$, $1 \leq i+i \wedge i \leq n$, $i \leq i$ ou $2|i$.

Proposition 3.51. *Tout schéma linéairement et quasi-syntaxiquement encadré est syntaxiquement encadré.*

Proposition 3.52. *Tout schéma linéairement et quasi-fortement encadré est fortement encadré.*

CHAPTER 4

STAB

Nous présentons une première procédure de preuve pour les schémas. D'après le théorème 3.2.1, nous savons qu'elle n'est pas complète pour la réfutation. En revanche, nous verrons qu'elle est complète pour la satisfaisabilité. Notre but est de disposer d'une procédure utilisable en pratique et qui permettra par la suite d'identifier des classes décidables de schémas. Appelée STAB pour "Schemata TABleaux", cette procédure est une extension naturelle des tableaux propositionnels (défintion 2.19) aux schémas de formules. STAB a été introduit dans [ACP09b].

4.1 System Presentation

Dans ce chapitre, nous supposons que toutes les itérations sont fortement encadrées (définition 3.22). Il est facile de voir que STAB préserve cette propriété.

Definition 4.1 (STAB)**.** *Les nœuds des tableaux de* STAB *sont étiquetés à la fois par des ensembles de schémas, de motifs et de contraintes. Nous notons $S(\nu)$ (resp. $M(\nu)$, resp. $C(\nu)$) l'ensemble des schémas (resp. motifs, resp. contraintes) étiquetant un nœud ν. Les règles sont les suivantes :*

- *Règles propositionnelles :*

$$\frac{\Gamma \cup \{m_1 \wedge m_2\}}{\Gamma \cup \{m_1\} \cup \{m_2\}} \wedge \qquad \frac{\Gamma \cup \{m_1 \vee m_2\}}{\Gamma \cup \{m_1\} \mid \Gamma \cup \{m_2\}} \vee$$

- *Règles de contraintes :*

$$\frac{\Gamma \cup \{m \,\&\, c\}}{\Gamma \cup \{m\} \cup \{c\}} \,\&$$

- *Règles d'itérations :*

$$\frac{\Gamma \cup \{ \bigwedge_{i=e}^{f} m\}}{\Gamma \cup \{m[f/i] \wedge \bigwedge_{i=e}^{f-1} m \,\&\, e \leq f\} \mid \Gamma \cup \{\top \,\&\, e > f\}} \text{ ITERATED } \wedge$$

$$\frac{\Gamma \cup \{ \bigvee_{i=e}^{f} m\}}{\Gamma \cup \{m[f/i] \vee \bigvee_{i=e}^{f-1} m \,\&\, e \leq f\} \mid \Gamma \cup \{\bot \,\&\, e > f\}} \text{ ITERATED } \vee$$

- *Règle de contradiction :*

$$\frac{\Gamma \cup \{p_{e_1,\ldots,e_n}\} \cup \{\neg p_{f_1,\ldots,f_n}\}}{\Gamma \cup \{p_{e_1,\ldots,e_n}\} \cup \{\neg p_{f_1,\ldots,f_n}\} \cup \{e_1 \neq f_1 \wedge \ldots \wedge e_n \neq f_n\}} \text{ CONTRADICTION}$$

Une feuille ν est close ssi $\bot \in M(\nu)$ ou $C(\nu)$ insatisfaisable (c.-à-d. la conjonction de ses éléments).

Pour la sémantique, nous associons à chaque nœud ν le schéma :

$$\left(\bigwedge_{m\in M(\nu)} m \wedge \bigwedge_{s\in S(\nu)} m_s\right) \,\&\, \left(\bigwedge_{s\in S(\nu)} c_s \wedge \bigwedge_{c\in C(\nu)} c\right)$$

Pour savoir si un schéma est satisfaisable ou non, nous appliquons les règles sur un tableau contenant un seul nœud étiqueté par le singleton contenant ce schéma.

Quelques remarques :

- les règles propositionnelles sont exactement les mêmes que celles des tableaux propositionnels;
- les règles d'itérations peuvent trivialement être simplifiées, au détriment de la symétrie, en :

$$\frac{\Gamma \cup \{\bigwedge_{i=e}^{f} m\}}{\Gamma \cup \{m[f/i] \wedge \bigwedge_{i=e}^{f-1} m \,\&\, e \leq f\} \mid \Gamma \cup \{e > f\}} \qquad \frac{\Gamma \cup \{\bigvee_{i=e}^{f} m\}}{\Gamma \cup \{m[f/i] \vee \bigvee_{i=e}^{f-1} m \,\&\, e \leq f\}}$$

STAB ne montrera sa véritable utilité qu'avec la détection de cycles, introduite au chapitre 5. Néanmoins, cette procédure est d'ores et déjà plus efficace que la procédure naïve esquissée dans la démonstration du théorème 3.15 (c.-à-d. l'énumération de toutes les instances). Tout d'abord, STAB peut terminer quand le schéma est insatisfaisable, ce qui n'est pas le cas de la procédure naïve (à moins que le schéma soit juste une formule propositionnelle insatisfaisable) : c'est le cas p.ex. pour $\bigwedge_{i=1}^{n} \bot \,\&\, n \geq 1$ (et plus généralement pour tout schéma de la forme $\bigwedge_{i=1}^{n} \phi \,\&\, n \geq 1$ où ϕ est une formule propositionnelle insatisfaisable). De plus STAB peut terminer beaucoup plus rapidement quand le schéma est satisfaisable : si nous prenons, p.ex., le schéma $(\bigwedge_{i=n}^{10000} p_1 \wedge \neg p_1)$ nous devrons attendre, avec la procédure naïve, d'avoir atteint $n = 10000$ alors que STAB trouvera immédiatement le modèle $n = 10000$, $p_1 = \text{false}$.

La figure 4.1 donne un exemple simple de dérivation de STAB.

4.2 Soundness

Les tableaux de STAB sont interprétés suivant la définition 2.14. Nous utiliserons la notation $\mathfrak{I} \models x$ que x désigne un nœud, son étiquette ou un tableau.

Proposition 4.2. *Soient une interprétation $\mathfrak{I}$ et une itération $\bigwedge_{i=e}^{f} m$. $\mathfrak{I} \models \bigwedge_{i=e}^{f} m$ ssi soit $\mathfrak{I}_{\text{arith}} \models e \leq f$ et $\langle \bigwedge_{i=e}^{f} m\rangle_{\mathfrak{I}_{\text{arith}}} = \langle m[f/i] \wedge \bigwedge_{i=e}^{f-1} m\rangle_{\mathfrak{I}_{\text{arith}}}$, soit $\mathfrak{I}_{\text{arith}} \models e > f$ et $\langle \bigwedge_{i=e}^{f} m\rangle_{\mathfrak{I}_{\text{arith}}} = \langle \top\rangle_{\mathfrak{I}_{\text{arith}}}$.*

Nous démontrons séparément les points 1 et 2 de la définition 2.15 :

Lemma 4.3. *Soient :*

- *une interprétation $\mathfrak{I}$,*
- *une feuille ν d'un tableau t,*
- *et un tableau t' obtenu par application d'une règle de* STAB *en ν.*

Alors $\mathfrak{I} \models \nu$ ssi il existe un fils ν' de ν dans t' t.q. $\mathfrak{I} \models \nu'$.

Notons bien qu'il s'agit là d'une *équivalence.*

Preuve. Par inspection des règles. Le résultat est classique pour les règles propositionnelles, et il est trivial pour la règle & (qui est très proche de $\wedge$). Pour ITERATED $\wedge$, c'est une conséquence directe de la proposition précédente. Le résultat correspondant à la disjonction s'obtient par dualité, d'où la conclusion pour ITERATED $\vee$. Enfin pour CONTRADICTION il est clair que si $\mathfrak{I} \models \nu$ nous avons nécessairement $e_1 \neq f_1 \wedge \ldots \wedge e_n \neq f_n$ (autrement ν est trivialement insatisfaisable car elle contient un atome et sa négation). La réciproque est triviale car la prémisse est incluse dans la conclusion. □

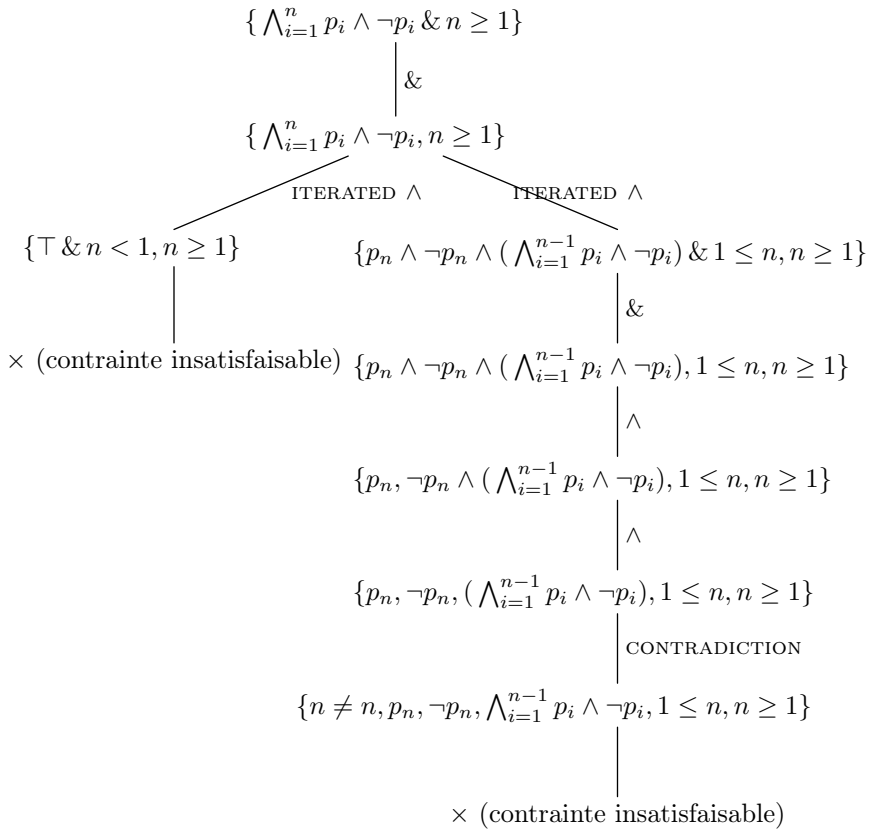


Figure 4.1: Application de STAB à $\bigwedge_{i=1}^{n} p_i \wedge \neg p_i \,\&\, n \geq 1$ ($\times$ désigne une branche fermée)

Nous démontrons maintenant le point 2 de la définition 2.15 :

Lemma 4.4. *Si une feuille d'un tableau t est irréductible et non close alors t est satisfaisable.*

Preuve. Soit ν cette feuille. Il est clair que l'ensemble étiquetant ν ne contient que des littéraux et des formules arithmétiques, par irréductibilité par rapport aux règles propositionnelles et des règles d'itérations. Soient deux littéraux $p_{e_1,\ldots,e_n}$ et $\neg p_{f_1,\ldots,f_n}$ contenus dans l'ensemble de ν. Par irréductibilité de CONTRADICTION, cet ensemble contient aussi $e_1 \neq f_1 \wedge \ldots \wedge e_n \neq f_n$. Comme ν est non close, $C(\nu)$ est satisfaisable, il existe donc un environnement ρ tel que $\rho \models e_1 \neq f_1 \wedge \ldots \wedge e_n \neq f_n$, donc $\langle p_{e_1,\ldots,e_n} \rangle_\rho \neq \langle \neg p_{f_1,\ldots,f_n} \rangle_\rho$. Il existe en fait un environnement ρ satisfaisant *toutes* les formules arithmétiques de ν, donc tel que pour *toute paire* de littéraux $p_{e_1,\ldots,e_n}$ et $\neg p_{f_1,\ldots,f_n}$ nous avons $\langle p_{e_1,\ldots,e_n} \rangle_\rho \neq \langle \neg p_{f_1,\ldots,f_n} \rangle_\rho$. Par conséquent l'ensemble $\{\langle l \rangle_\rho \mid l \text{ est un littéral de } \nu\}$ est un ensemble non contradictoire de littéraux propositionnels. Il existe donc une interprétation propositionnelle $\mathfrak{I}$ validant cet ensemble. Par définition, $(\rho, \mathfrak{I})$ est un modèle de ν et donc de t. □

Par conséquent :

Theorem 4.5 (Correction)**.** STAB *est correcte.*

Preuve. C'est une conséquence immédiate des lemmes 4.3 et 4.4, et de la définition 2.15. □

Et donc STAB vérifie le théorème 2.16.

4.3 Completeness for Satisfiability

Le but est donc de démontrer que si un tableau est satisfaisable nous atteindrons toujours par application des règles un tableau dont une branche au moins sera irréductible et non close. L'idée générale consiste à montrer que l'application de chaque règle nous rapproche d'un tel tableau. Cette notion de "rapprochement" est formalisée par une mesure.

Soit $\mathfrak{I}$ une interprétation, nous définissons la mesure $\mu^1_{\mathfrak{I}}(x)$ qui associe un entier à un schéma, un motif ou une contrainte :

- $\mu^1_{\mathfrak{I}}(c) \stackrel{\text{def}}{=} 0$ si c est une contrainte;
- $\mu^1_{\mathfrak{I}}(m \,\&\, c) \stackrel{\text{def}}{=} \mu^1_{\mathfrak{I}}(m)$;
- $\mu^1_{\mathfrak{I}}(m) \stackrel{\text{def}}{=} 1$ si m est $\top$, $\bot$ ou un litéral;
- $\mu^1_{\mathfrak{I}}(m_1 \star m_2) \stackrel{\text{def}}{=} \mu^1_{\mathfrak{I}}(m_1) + \mu^1_{\mathfrak{I}}(m_2)$ où $\star \in \{\wedge, \vee\}$;
- $\mu^1_{\mathfrak{I}}(\Pi^f_{i=e} m) \stackrel{\text{def}}{=} 2 + \#E + \sum_{g \in E} \mu^1_{\mathfrak{I}}(m[g/i])$ où $E \stackrel{\text{def}}{=} [\mathfrak{I}_{\text{arith}}(e); \mathfrak{I}_{\text{arith}}(f)]$, $\#E$ dénote le cardinal de E et $\Pi \in \{\bigwedge, \bigvee\}$.

et pour un ensemble F de schémas, motifs et contraintes, $\mu^1_{\mathfrak{I}}(F) \stackrel{\text{def}}{=} \{\mu^1_{\mathfrak{I}}(x) \mid x \in F\}$ ordonné par l'extension multi-ensemble de l'ordre usuel sur les entiers naturels.

Soit un nœud ν d'un tableau t, nous définissons $\mu^2(\nu, t)$ comme le nombre de paires $(p_{e_1,\ldots,e_n}, \neg p_{f_1,\ldots,f_n})$ de littéraux de ν tels qu'il existe ρ satisfaisant $(\bigwedge\{c \in C(\nu)\}) \wedge (\bigwedge\{c_s \mid s \in S(\nu)\}) \wedge e_1 = f_1 \wedge \ldots \wedge e_n = f_n$. Et finalement : $\mu_{\mathfrak{I}}(\nu, t) \stackrel{\text{def}}{=} (\mu^1_{\mathfrak{I}}(M(\nu) \cup S(\nu) \cup C(\nu)), \mu^2(\nu))$, ordonné par l'extension lexicographique de ces deux mesures.

La propriété suivante est la raison d'être de cette mesure : toute règle de STAB diminue strictement la valeur de la mesure.

Lemma 4.6. *Soient:*

- *t un tableau satisfaisable,*
- *$\mathfrak{M}$ un modèle de t,*
- *ν une feuille d'un tableau t, telle que $\mathfrak{M} \models \nu$,*
- *et t' un tableau obtenu par application d'un règle de* STAB *à ν.*

Alors pour chaque fils ν' de ν dans t' tel que $\mathfrak{M} \models \nu'$, nous avons $\mu_{\mathfrak{M}}(\nu', t') < \mu_{\mathfrak{M}}(\nu, t)$.

Example 4.7. *Supposons que ν est étiquetée par $\bigwedge^n_{i=1} p_i$ et que ν' est obtenue par application de* ITERATED *$\wedge$, générant d'un côté $p_n \wedge \bigwedge^{n-1}_{i=1} p_i \,\&\, n \geq 1$ et de l'autre $\top \,\&\, n < 1$.*

Soit le modèle $\mathfrak{M}$ de ν défini par $\mathfrak{M}_{\text{arith}}(n) \stackrel{def}{=} 5$ (peu importe la valeur ailleurs) et $\mathfrak{M}_{\text{prop}}(p_i) \stackrel{def}{=} \text{true}$ pour tout $i \in [1; 5]$. Ce modèle est aussi un modèle de $p_n \wedge \bigwedge^{n-1}_{i=1} p_i \,\&\, n \geq 1$ (mais pas de $\top \,\&\, n < 1$ car $\mathfrak{M}_{\text{arith}}(n) \geq 1$). Pour simplifier, nous n'observons que le comportement de μ^1 et comme la mesure des contraintes est nulle, nous ne précisons pas ces dernières.

Alors, pour $\bigwedge^n_{i=1} p_i$, nous avons $E = \{1, 2, 3, 4, 5\}$ et :

$$\mu^1_{\mathfrak{M}}(\bigwedge_{i=1}^{n} p_i) = 2 + 5 + (\mu^1_{\mathfrak{M}}(p_1) + \mu^1_{\mathfrak{M}}(p_2) + \mu^1_{\mathfrak{M}}(p_3) + \mu^1_{\mathfrak{M}}(p_4) + \mu^1_{\mathfrak{M}}(p_5)) = 12$$

Pour $p_n \wedge \bigwedge^{n-1}_{i=1} p_i$, nous avons $E = \{1, 2, 3, 4\}$ et :

$$\mu^1_{\mathfrak{M}}(p_n \wedge \bigwedge_{i=1}^{n-1} p_i) = \mu^1_{\mathfrak{M}}(p_5) + 2 + 4 + (\mu^1_{\mathfrak{M}}(p_1) + \mu^1_{\mathfrak{M}}(p_2) + \mu^1_{\mathfrak{M}}(p_3) + \mu^1_{\mathfrak{M}}(p_4)) = 11$$

Soit le modèle $\mathfrak{N}$ de ν défini par $\mathfrak{N}_{\text{arith}}(n) \stackrel{def}{=} 0$ (peu importe la valeur ailleurs) et n'importe quelle interprétation propositionnelle. Ce modèle est aussi un modèle de $\top$ (mais pas de $\bigwedge^{n-1}_{i=1} p_i \,\&\, n \geq 1$ car $\mathfrak{N}_{\text{arith}}(n) < 1$).

Alors, pour $\bigwedge^n_{i=1} p_i$, nous avons $E = \emptyset$ et $\mu^1_{\mathfrak{N}}(\bigwedge^n_{i=1} p_i) = 2$. Pour $\top$ nous avons $\mu^1_{\mathfrak{N}}(\top) = 1$.

Dans tous les cas, la mesure a bien strictement diminué.

Preuve (du lemme 4.6). Par inspection des règles:

- $\vee$: nous remplaçons $m_1 \vee m_2$ par un motif strictement inférieur (en effet $\mu^1_{\mathfrak{M}}(m) \geq 1$ quel que soit m, donc $\mu^1_{\mathfrak{M}}(m_1 \vee m_2) > \mu^1_{\mathfrak{M}}(m_1)$ et $\mu^1_{\mathfrak{M}}(m_1 \vee m_2) > \mu_{\mathfrak{M}}(m_2)$).
- $\wedge$: cette fois nous remplaçons $m_1 \wedge m_2$ par deux motifs inférieurs, donc comme nous considèrons un ordre multi-ensemble, ν' est bien strictement inférieure.
- & : idem $\wedge$.
- ITERATED $\wedge$: nous avons $\mu^1_{\mathfrak{M}}(\bigwedge_{i=e}^{f} m) = 2 + \#E + \sum_{g \in E} \mu^1_{\mathfrak{M}}(m[g/i])$ où $E \stackrel{\text{def}}{=} [\mathfrak{M}_{\text{arith}}(e); \mathfrak{M}_{\text{arith}}(f)]$. Supposons que ν' corresponde à la branche droite de la règle : dans ce cas $\#E = 0$ et la somme est vide donc $\mu^1_{\mathfrak{M}}(\bigwedge_{i=e}^{f} m) = 2 \geq \mu^1_{\mathfrak{M}}(\top \,\&\, e > f) = 1$, et nous avons bien le résultat. Supposons maintenant que ν' corresponde à la branche gauche, dans ce cas :

$$\begin{array}{l} \mu^1_{\mathfrak{M}}(m[f/i] \wedge \bigwedge_{i=e}^{f-1} m \,\&\, e \leq f) \\ = \mu^1_{\mathfrak{M}}(m[f/i]) + 2 + \#(E \setminus \{\mathfrak{M}_{\text{arith}}(f)\}) + \sum_{g \in (E \setminus \{\mathfrak{M}_{\text{arith}}(f)\})} \mu^1_{\mathfrak{M}}(m[g/i]) \\ = \mu^1_{\mathfrak{M}}(m[f/i]) + 2 + \#E - 1 + \sum_{g \in (E \setminus \{\mathfrak{M}_{\text{arith}}(f)\})} \mu^1_{\mathfrak{M}}(m[g/i]) \end{array}$$

  De plus, il est facile de prouver que $\mu^1_{\mathfrak{M}}(m[f/i]) = \mu^1_{\mathfrak{M}}(m[\mathfrak{M}_{\text{arith}}(f)/i])$, donc :

$$\begin{array}{l} = \mu^1_{\mathfrak{M}}(m[\mathfrak{M}_{\text{arith}}(f)/i]) + 2 + \#E - 1 + \sum_{g \in (E \setminus \{\mathfrak{M}_{\text{arith}}(f)\})} \mu^1_{\mathfrak{M}}(m[g/i]) \\ = 2 + \#E - 1 + \sum_{g \in E} \mu^1_{\mathfrak{M}}(m[g/i]) \\ = (2 + \#E + \sum_{g \in E} \mu^1_{\mathfrak{M}}(m[g/i])) - 1 \\ = \mu^1_{\mathfrak{M}}(\bigwedge_{i=e}^{f} m) - 1 \end{array}$$

- ITERATED $\vee$: idem ITERATED $\wedge$.
- CONTRADICTION : il est clair que $\mu^2(\nu, t)$ diminue strictement alors que $\mu^1_{\mathfrak{M}}(M(\nu) \cup S(\nu) \cup C(\nu))$ est constant, donc $\mu_{\mathfrak{M}}(\nu, t)$ diminue strictement.

□

Proposition 4.8. *Si une feuille est satisfaisable et irréductible alors elle n'est pas close.*

Preuve. Si elle était close soit elle contiendrait $\bot$, soit l'ensemble de ses contraintes seraient insatisfaisable. Dans les deux cas la feuille elle-même serait insatisfaisable. □

Theorem 4.9 (Complétude pour la satisfaisabilité)**.** *Soient :*

- t_0 *un tableau satisfaisable,*
- $\mathfrak{M}$ *un modèle de* t_0*,*
- *et* $(t_i)_{i \in I}$ *une dérivation équitable.*

Il existe $k \in I$ *et une feuille* ν_k *de* t_k *telle que* ν_k *est irréductible et non close.*

Preuve. D'après le lemme 4.3, pour tout $i \in I$, t_i contient une feuille ν_i telle que $\mathfrak{M} \models \nu_i$. Soit k tel que $\mu_{\mathfrak{M}}(\nu_k, t_k)$ soit minimale (k existe car la mesure est bien fondée). Il est clair d'après le lemme précédent qu'aucune règle ne s'applique à ν_k dans la dérivation (autrement cela entrerait en contradiction avec l'hypothèse de minimalité). Comme la dérivation est équitable, soit ν_k est irréductible, soit il existe une autre feuille qui est irréductible et non close. Si ν_k est irréductible, alors elle ne peut être close d'après la proposition précédente. □

4.4 Example

4.4.1 Un exemple de schéma satisfaisable

Nous commençons par un exemple où le schéma est satisfaisable: $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$ (c.-à-d. l'exemple récurrent $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$ mais sans la contrainte $n \geq 0$ qui assure l'insatisfaisabilité de ce dernier). Le tableau obtenu est donné en figure 4.2.

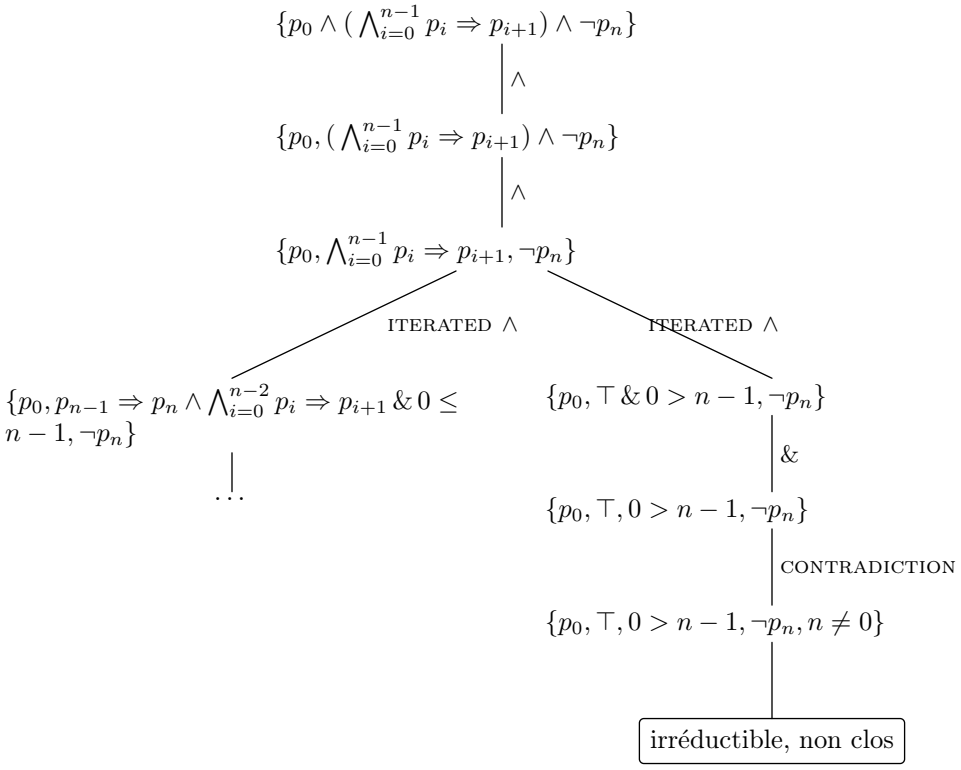


Figure 4.2: Application de STAB à $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$

Par correction, le schéma est donc satisfaisable. À la lueur de la feuille irréductible non close, nous voyons plus précisément qu'il existe un modèle pour tout $n < 0$. Notons que la partie gauche de l'arbre peut très bien être infinie (d'ailleurs elle l'est, comme nous le verrons avec l'exemple suivant) concrétisant ainsi la nécessité de l'équité dans les hypothèses du théorème 4.9. En particulier, une implémentation doit s'assurer d'appliquer les règles de façon équitable pour ne pas passer tout son temps dans la branche infinie (et donc ne jamais terminer).

4.4.2 Échec de terminaison

Nous ne présentons qu'un exemple jouet, c.-à-d. rapide et trivial, parce que nous allons voir que, sans détection de cycle, STAB ne va pas bien loin. Nous appliquons STAB à l'exemple récurrent $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$ (c.-à-d. l'exemple précédent mais avec la contrainte $n \geq 0$), que nous savons insatisfaisable (cf l'exemple 3.9). Le tableau obtenu est (partiellement) donné en figure 4.3.

Il est facile de voir que la branche marquée "..." sera étendue indéfiniment: les flèches en pointillés montrent que la preuve est en train de se "répéter", la procédure ne s'arrête donc jamais. Ces flèches présentent pour la première fois de façon informelle la notion de cycle. Cette notion est étudiée formellement dans le chapitre suivant. L'objectif est d'ajouter à STAB un mécanisme permettant de détecter ces cycles et de fermer les feuilles correspondantes.

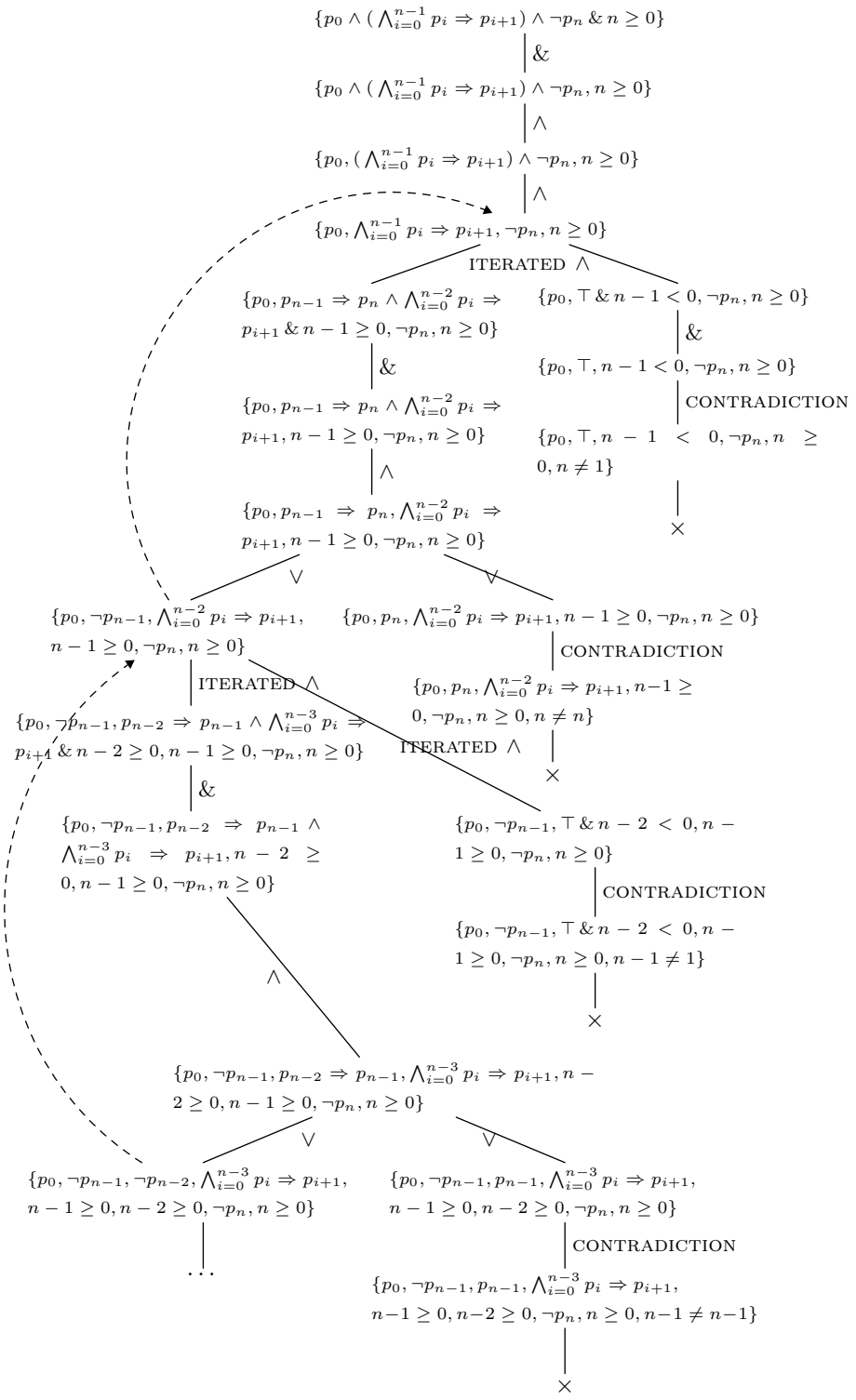


Figure 4.3: Application de STAB à $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$.

CHAPTER 5

Cycle Detection

D'après le théorème d'incomplétude, nous savons qu'aucune procédure ne peut décider la satisfaisabilité des schémas. Cependant, comme nous l'avons vu en section 4.4.2, l'intuition permet parfois de repérer des "cycles" dans la construction (divergente) d'une preuve. Dans ce chapitre, nous formalisons cette notion afin d'améliorer la terminaison en détectant ces cycles de manière automatique. Nous proposons une première définition générale mais indécidable (section 5.1). Puis nous prenons l'approche opposée avec un raffinement décidable, presque trivial, de la définition générale : l'"égalité à un décalage près" (section 5.2). Enfin en section 5.3 nous proposons des extensions qui, à partir d'un raffinement quelconque de la définition générale, construisent un raffinement plus puissant. Ces extensions appliquées à l'égalité à un décalage près seront suffisantes pour prouver tous les résultats de décidabilité de ce mémoire. Enfin nous revenons sur l'exemple qui divergeait avec Stab (section 4.4.2) pour constater que la détection de cycle évite la divergence (section 5.4).

Le contenu de ce chapitre a été développé tout au long de [ACP09b, ACP09a, ACP10d, ACP10a] et particulièrement dans [ACP10b].

5.1 General Definition

L'existence d'un cycle signifie qu'un nœud "boucle" sur un autre nœud. De façon informelle, nous voulons qu'un nœud boucle sur un autre si nous pouvons montrer que la procédure se comporte de la même façon sur les deux nœuds. Cependant, présentée de cette manière, la notion de bouclage est dépendante de la procédure. Nous préfèrons donc définir cette notion par rapport au problème lui-même, c.-à-d. en termes de satisfaisabilité[1]. Ainsi, nous pouvons introduire la notion de bouclage, en première approximation, de la façon suivante : si la satisfaisabilité d'un nœud ν_3 est équivalente à la satisfaisabilité d'un nœud déjà vu ν_2, alors il ne sert à rien de le développer. En fait il suffit même que la satisfaisabilité de ν_2 soit seulement conséquence de la satisfaisabilité de ν_3.

De même, si la satisfaisabilité de ν_2 implique la satisfaisabilité d'un autre nœud ν_1, alors il ne sert à rien de développer ν_2, et ainsi de suite. Nous voyons bien que si nous voulons conclure, il faut s'assurer qu'il y a bien une "progression" dans cette série de nœuds. Pour ce faire, il faut que les modèles du nœud sur lequel nous bouclons soient inférieurs aux modèles du nœud qui boucle, par rapport à un ordre quelconque sur les modèles. Et si nous voulons nous assurer que nous finirons bien par trouver un modèle, cet ordre doit être bien fondé.

Nous obtenons ainsi la définition suivante :

[1] Bien que ce ne soit pas le point de vue élaboré dans ce mémoire, il serait aussi intéressant de tenter de formaliser cette notion de "comportement identique" de la procédure : peut-être en s'orientant vers les notions de bisimulation, ou en s'intéressant aux travaux qui étudient les connections entre géométrie et théorie de la preuve. Il y a en effet fort à parier que le problème clé dans le cadre d'une telle étude serait la caractérisation d'une bonne notion de forme normale d'une preuve ("bonne" par rapport au problème qui nous intéresse ici).

Definition 5.1 (Bouclage)**.** *Soit $\prec$ un ordre strict et bien fondé sur les interprétations. Nous disons qu'un schéma s_1* boucle *sur un schéma s_2 par rapport à $\prec$ ssi pour tout modèle $\mathfrak{M}_1$ de s_1 il existe un modèle $\mathfrak{M}_2$ de s_2 t.q. $\mathfrak{M}_2 \prec \mathfrak{M}_1$. Nous appelons s_1 le* bourgeon*, s_2 le* compagnon[2] *et $\prec$ un* ordre de bouclage*.*

En fait, le lecteur l'aura peut-être reconnu, nous effectuons simplement une preuve par descente infinie [Wika] : si l'existence d'un contre-exemple à un certain rang d'un ordre bien fondé implique l'existence d'un contre-exemple à un rang inférieur, alors il ne peut pas y avoir de contre-exemple du tout, par bonne fondation. C'est une variante classique de la preuve par récurrence mise en évidence, semble-t-il, par Fermat.

Example 5.2. *Soit une variable n. Pour toutes interprétations $\mathfrak{I}_1$ et $\mathfrak{I}_2$ telles que $\mathfrak{I}_{1\text{arith}}(n) \geq 1$ et $\mathfrak{I}_{2\text{arith}}(n) \geq 1$, nous définissons $\mathfrak{I}_1 \prec \mathfrak{I}_2$ ssi $\mathfrak{I}_{1\text{arith}}(n) < \mathfrak{I}_{2\text{arith}}(n)$. Comme nous avons imposé $\mathfrak{I}_{1\text{arith}}(n) \geq 1$ et $\mathfrak{I}_{2\text{arith}}(n) \geq 1$, cet ordre est bien fondé (il est évidemment strict).*

Il est évident que tout modèle de $\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$ donne à n une valeur supérieure ou égale à 1, nous pouvons donc appliquer l'ordre $\prec$ sur les interprétations. Alors, $s_1 \stackrel{def}{=} \bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1$ boucle bien sur $s_2 \stackrel{def}{=} \bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$ par rapport à $\prec$ standard (par rapport à n) : soit un modèle $\mathfrak{M}_1$ de s_1, nous pouvons construire un modèle $\mathfrak{M}_2$ de s_2 par $\mathfrak{M}_{2\text{arith}}(n) \stackrel{def}{=} \mathfrak{M}_{1\text{arith}}(n) - 1$ et pour tout $i \in [1; \mathfrak{M}_{2\text{arith}}(n)]$: $\mathfrak{M}_{2\text{prop}}(p_i) \stackrel{def}{=} \mathfrak{M}_{1\text{prop}}(p_i)$. Il est clair que $\mathfrak{M}_2$ est bien un modèle de s_2 et que $\mathfrak{M}_2 \prec \mathfrak{M}_1$.

En pratique, lorsqu'un cycle est détecté, la branche du nœud qui boucle est close.

Definition 5.3 (Règle de bouclage)**.** *Nous appelons* règle de bouclage *la règle comme quoi si un nœud boucle sur un autre nœud de l'arbre, alors la branche est close.*

Nous emploierons plutôt le terme *bloquée* pour clairement distinguer ce cas de celui où une branche est close de façon classique. Dans toute la suite, nous supposons l'ordre de bouclage $\prec$ fixé.

5.1.1 Résultats génériques

Nous donnons deux résultats génériques : si une procédure est correcte (resp. complète), alors ajouter la règle de bouclage préserve la correction (resp. complétude).

Theorem 5.4 (Correction)**.** *Si une procédure à base de tableaux, sans règle de bouclage, est correcte alors elle l'est toujours quand nous ajoutons la règle de bouclage.*

Preuve. Comme toutes les règles autres que la règle de bouclage sont inchangées, la propriété 1 de la définition 2.15 est préservée. Considérons maintenant la propriété 2. Soit une feuille irréductible non close de la procédure avec bouclage. Appelons b la branche qui la contient. Comme la règle de bouclage ferme une branche, elle ne peut pas avoir été appliquée sur un quelconque nœud de b. Par conséquent toutes les règles utilisées dans la construction de b sont uniquement des règles de la procédure sans règle de bouclage. Comme nous savons que cette procédure est correcte, la feuille est bien satisfaisable. □

Theorem 5.5 (Complétude)**.** *Si une procédure à base de tableaux est correcte et complète pour la satisfaisabilité (c.-à-d. si le tableau initial est satisfaisable alors dans toute dérivation équitable il existe un tableau contenant une feuille irréductible non close), alors la même procédure avec règle de bouclage est complète pour la satisfaisabilité.*

Preuve. Soit un schéma satisfaisable s et t_0 l'arbre ne contenant qu'un nœud étiqueté par s. Soit une dérivation équitable $(t_i)_{i\in I}$ obtenue par la procédure *avec bouclage*. Nous construisons une dérivation $(t'_i)_{i\in I}$ de la procédure *sans bouclage*, à partir de $(t_i)_{i\in I}$, en développant (de façon équitable) les branches bloquées, c.-à-d. closes par bouclage (notons que cette dérivation n'est pas unique). Par complétude de la procédure sans bouclage, il existe une feuille irréductible non close dans un tableau de $(t'_i)_{i\in I}$. Nous montrons que l'existence d'une telle feuille ν dans un tableau de $(t'_i)_{i\in I}$ implique l'existence d'une feuille irréductible non close dans un tableau de $(t_i)_{i\in I}$. Cette preuve se fait par induction sur le plus petit modèle $\mathfrak{M}$ de ν (qui a bien un modèle car irréductible et non close; et qui a un plus petit modèle car $\prec$ est bien fondée).

[2] Nous suivons ici la terminologie des preuves cycliques : "bud" et "companion", voir chapitre 10.

Si $\mathfrak{M}$ est minimal parmi tous les modèles de s (car, par correction, $\mathfrak{M}$ est bien un modèle de s), alors il est impossible que ν boucle sur un autre nœud de $(t_i)_{i\in I}$. Donc ν appartient aussi à un tableau de $(t_i)_{i\in I}$. Et, par correction, $\mathfrak{M}$ est aussi un modèle de tout ancêtre de ν, donc aucun n'est bloqué dans $(t_i)_{i\in I}$. Par conséquent il existe bien une feuille irréductible non close dans $(t_i)_{i\in I}$.

S'il existe un modèle $\mathfrak{N}$ de s, tel que $\mathfrak{N} \prec \mathfrak{M}$ alors il se peut que ν ou l'un de ses ancêtres boucle dans $(t_i)_{i\in I}$ sur un nœud η. Comme ν est satisfaisable, ses ancêtres et η le sont aussi. Soit $\mathfrak{N}$ le plus petit modèle de η. Nous avons donc $\mathfrak{N} \prec \mathfrak{M}$. Par correction, il existe un fils de η dont $\mathfrak{N}$ est modèle. De même il existe un fils de ce fils dont $\mathfrak{N}$ est modèle, et ainsi de suite. Il existe donc une suite de descendants de η pour lesquels $\mathfrak{N}$ est modèle. Or η apparaît aussi dans $(t'_i)_{i\in I}$ (puisque $(t'_i)_{i\in I}$ est obtenu à partir de $(t_i)_{i\in I}$). Donc, par complétude de la procédure sans bouclage, la suite de descendants de η est nécessairement finie et se termine par une feuille η' irréductible et non close pour laquelle $\mathfrak{N}$ est modèle. Nous avons donc une feuille irréductible non close dans un tableau de $(t'_i)_{i\in I}$. Son plus petit modèle est strictement inférieur au plus petit modèle de ν. Nous pouvons donc conclure par induction. □

Corollary 5.6. STAB*, muni de la détection de cycle, est correct et complet.*

5.1.2 Ordre standard

Dans notre contexte, il est intuitif de raisonner par induction sur les paramètres des schémas. Nous l'avons vu notamment dans l'exemple de cycle donné en 4.4.2 : nous souhaitons, par exemple, que $\bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1$ boucle sur $\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$. Ceci doit être reflété par l'ordre $\prec$ utilisé dans la définition précédente. C'est le cas par exemple de l'*ordre standard* :

Definition 5.7. *Soient $\mathfrak{I}_1$ et $\mathfrak{I}_2$ deux interprétations. Soit un paramètre n. L'*ordre standard *$\prec_n$ par rapport à n est défini par : $\mathfrak{I}_1 \prec_n \mathfrak{I}_2$ ssi $\mathfrak{I}_{1\text{arith}}(n) < \mathfrak{I}_{2\text{arith}}(n)$.*

Nous notons $s_1 \prec_n s_2$ si s_2 boucle sur s_1 avec l'ordre standard.

Cependant cet ordre n'est pas bien fondé en général car $\mathfrak{I}_{1\text{arith}}(n) \in \mathbb{Z}$. Il faut donc se restreindre à des interprétations pour lesquels n a une valeur minimale. Nous avons alors :

Proposition 5.8. *Soit $k \in \mathbb{Z}$. $\prec_n$ est bien fondé sur l'ensemble des interprétations donnant à n une valeur supérieure ou égale à k.*

Ainsi il est possible d'utiliser cet ordre uniquement si nous pouvons assurer que tous les modèles d'un schéma donnent à n une valeur supérieure ou égale à k. De plus, cette propriété doit être vraie non seulement du schéma à la racine d'un tableau, mais aussi de tout autre nœud du tableau car l'ordre de bouclage ne peut pas changer au cours de la procédure. Cependant si un système de preuve est correct, alors tout modèle d'un quelconque nœud est aussi modèle de la racine. Par conséquent si tout modèle de la racine attribue à n une valeur supérieure à k, alors c'est aussi le cas de tout modèle de n'importe quel nœud apparaissant dans le tableau. Ainsi si un système de preuve est correct, alors il suffit, pour pouvoir utiliser la règle de bouclage avec l'ordre standard, d'imposer que tous les modèles du schéma initial donnent à n une valeur supérieure à k. Les deux systèmes étudiées dans cette thèse sont corrects.

En pratique, nous rencontrons très souvent des schémas dont tous les modèles ont cette propriété : c'est le cas de tout schéma s tel que $c_s \models n \geq k$ pour un certain $k \in \mathbb{Z}$. Comme, par la suite, le seul ordre de bouclage que nous utiliserons sera l'ordre standard, il sera sous-entendu que tous les schémas concernés auront cette propriété.

Notons enfin que nous pouvons facilement utiliser des ordres plus puissants p.ex. en considérant des tuples ou des ensembles finis de paramètres (au lieu d'un seul n) munis de l'extension lexicographique ou multiensemble. Mais après avoir développé la notion de bouclage de façon générale, nous nous intéresserons maintenant uniquement à sa mise en application pratique. Dans ce contexte, il est suffisant d'effectuer une induction sur un seul paramètre à la fois, nous nous contenterons donc de l'ordre standard par rapport à un paramètre.

Example 5.9. *Il est évident que tout modèle de $\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$ donne à n une valeur supérieure ou égale à 0, nous pouvons donc utiliser l'ordre standard pour la détection de cycle. D'autre part, $s_1 \stackrel{def}{=} \bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1$ boucle bien sur $s_2 \stackrel{def}{=} \bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$ par rapport à l'ordre standard (par rapport à n) : soit un modèle $\mathfrak{M}_1$ de s_1, nous pouvons construire un modèle $\mathfrak{M}_2$ de s_2 par $\mathfrak{M}_{2\text{arith}}(n) \stackrel{def}{=} \mathfrak{M}_{1\text{arith}}(n) - 1$*

et pour tout $i \in [1; \mathfrak{M}_{2\text{arith}}(n)]$ *:* $\mathfrak{M}_{2\text{prop}}(p_i) \stackrel{def}{=} \mathfrak{M}_{1\text{prop}}(p_i)$*. Il est clair que* $\mathfrak{M}_2$ *est bien un modèle de* s_2 *et que* $\mathfrak{M}_2 \prec_n \mathfrak{M}_1$*.*

De même $s_1 \stackrel{def}{=} \bigwedge_{i=2}^{n} p_i \,\&\, n \geq 2$ *boucle sur* $s_2 \stackrel{def}{=} \bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$ *par rapport à l'ordre standard : soit un modèle* $\mathfrak{M}_1$ *de* s_1*, nous pouvons construire un modèle* $\mathfrak{M}_2$ *de* s_2 *par* $\mathfrak{M}_{2\text{arith}}(n) \stackrel{def}{=} \mathfrak{M}_{1\text{arith}}(n) - 1$ *et pour tout* $i \in [2; \mathfrak{M}_{2\text{arith}}(n)]$ *:* $\mathfrak{M}_{2\text{prop}}(p_i) \stackrel{def}{=} \mathfrak{M}_{1\text{prop}}(p_{i-1})$*. Il est clair que* $\mathfrak{M}_2$ *est un modèle de* s_2 *et que* $\mathfrak{M}_2 \prec_n \mathfrak{M}_1$*.*

5.1.3 Raffinements de bouclage

Nous terminons cette section en introduisant des notions qui seront utiles par la suite. Vu le théorème 3.2.1 (incomplétude), notre but principal est désormais d'identifier des classes décidables de schémas. Pour ce faire, nous allons utiliser de façon cruciale la détection de cycle : en effet notre but sera de prouver que si un schéma vérifie certaines contraintes syntaxiques, alors chaque branche potentiellement infinie d'un tableau généré sera bloquée par une détection de cycle. Nous prouverons ceci en montrant tous les schémas potentiellement générés par la procédure appartiennent à un même ensemble, fini modulo une relation de bouclage. Par exemple, supposons qu'une procédure (sans détection de cycle) appliquée à $\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$ génère $\bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1$, puis $\bigwedge_{i=1}^{n-2} p_i \,\&\, n-2 \geq 1$, etc. Tous ces schémas appartiennent à l'ensemble infini :

$$\left\{ \bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1, \bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1, \bigwedge_{i=1}^{n-2} p_i \,\&\, n-2 \geq 1, \dots \right\}$$

En revanche il est clair que :

$$\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1 \prec_n \bigwedge_{i=1}^{n-3} p_i \,\&\, n-3 \geq 1$$
$$\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1 \prec_n \bigwedge_{i=1}^{n-2} p_i \,\&\, n-2 \geq 1$$
$$\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1 \prec_n \bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1$$

Donc si la procédure est munie de la détection de cycle avec l'ordre standard, alors la règle de bouclage fermera la branche contenant $\bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1$. Mathématiquement cela se traduit par le fait que bien que l'ensemble $\{ \bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1, \bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1, \bigwedge_{i=1}^{n-2} p_i \,\&\, n-2 \geq 1, \dots \}$ soit infini, il est fini "modulo le bouclage", ce que nous allons formaliser maintenant.

Definition 5.10. *Soit un ordre de bouclage sur les interprétations. Une relation binaire transitive* $\rhd$ *sur les schémas telle que pour tous* s_1 *et* s_2 *:* $s_1 \rhd s_2$ *implique que* s_1 *boucle sur* s_2 *par rapport à cet ordre, est appelée un raffinement de bouclage.*

Soit S *un ensemble de schémas et* $\rhd$ *un raffinement de bouclage.* S *est* $\rhd$-connexe *ssi pour tous* $s_1, s_2 \in S$ *nous avons soit* $s_1 \rhd s_2$*, soit* $s_2 \rhd s_1$*, soit* $s_1 = s_2$*. Nous appelons* composantes $\rhd$-connexes *d'un ensemble ses plus grands sous-ensembles* $\rhd$*-connexes. Un schéma* $s \in S$ *tel qu'il n'y a pas de* $s' \in S$ *vérifiant* $s \rhd s'$ *est appelé un* $\rhd$-minimum *de* S*.*

Si S *est* $\rhd$*-connexe, nous disons qu'il est* $\rhd$-bien fondé *ssi il contient un* $\rhd$*-minimum. Si* S *n'est pas* $\rhd$*-connexe, nous disons qu'il est* $\rhd$-bien fondé *ssi toutes ses composantes* $\rhd$*-connexes sont* $\rhd$*-bien fondés. Dans ce cas l'ensemble de tous les* $\rhd$*-minima de* S *est noté* $S/\rhd$*. Si cet ensemble est fini alors nous disons que* S *est fini modulo* $\rhd$*.*

Example 5.11. *Prenons pour* $\rhd$ *la relation de bouclage avec l'ordre standard. Alors l'ensemble* S_1 *est connexe :* $S_1 \stackrel{def}{=} \{p_n, p_{n-1}, p_{n-2}, \dots\}$*. En revanche* $S_2 \stackrel{def}{=} \{p_n, \neg p_n \wedge p_n\}$ *ne l'est pas (nous n'avons ni* $p_n \rhd \neg p_n \wedge p_n$*, ni* $\neg p_n \wedge p_n \rhd p_n$*).*

D'autre part, p_n *est un minimum de* S_1*, c'est aussi un minimum de* S_2*, tout comme* $\neg p_n \wedge p_n$*.*

S_1 *et* S_2 *sont bien fondés mais* $S_3 \stackrel{def}{=} \{\dots, p_{n+2}, p_{n+1}, p_n, p_{n-1}, p_{n-2}, \dots\}$ *ne l'est pas. Nous avons* $S_1/\rhd = \{p_n\}$ *et* $S_2/\rhd = \{p_n, \neg p_n \wedge p_n\}$*,* S_1 *et* S_2 *sont donc tous deux finis modulo* $\rhd$*.*

Note. La notion de bonne fondation rencontrée ici n'a pas la même raison d'être que celle rencontrée pour l'ordre entre interprétations (définition 5.1) : cette dernière sert à assurer la *complétude* de notre notion de bouclage alors qu'ici elle servira à démontrer la *terminaison* de procédures qui utilisent le bouclage.

5.2 Equality up to a Shift

Il est clair que la notion de bouclage n'est pas décidable en général, même en se restreignant à l'ordre standard : en effet elle requiert de décider l'existence d'un modèle, problème dont nous connaissons l'indécidabilité. En pratique, il nous faut donc considérer des cas particuliers décidables de bouclage.

Nous nous intéressons ici à un cas très simple de bouclage, que nous appelons l'"égalité à un décalage près". Dans toute la suite, c'est ce raffinement de bouclage que nous utiliserons[3].

5.2.1 Définition et propriétés élémentaires

Definition 5.12 (Égalité à un décalage près)**.** *Soient deux schémas s_1 et s_2. Nous disons que s_1 est* égal à s_2 à un décalage près de n *ssi $s_1 = s_2[n-k/n]$ où k est un entier* strictement *positif. Nous notons $s_1 \rightrightarrows_n^k s_2$ ou juste $s_1 \rightrightarrows_n s_2$ si k est clair dans le contexte ou sans importance pour le discours.*

Note. Notons que, contrairement à ce que le terme "égalité" peut laisser croire, l'égalité à un décalage près *n'est pas symétrique* (car nous devons avoir $k > 0$ dans la définition), c.-à-d. qu'en général nous n'avons pas nécessairement $s_2 \rightrightarrows_n s_1$ si $s_1 \rightrightarrows_n s_2$.

Example 5.13. $\bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1$ *est égal à* $\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$ *à un décalage près. Plus précisément :* $\bigwedge_{i=1}^{n-1} p_i \,\&\, n-1 \geq 1 \rightrightarrows_n^1 \bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$. *En revanche* $\bigwedge_{i=2}^{n} p_i \,\&\, n \geq 2 \not\rightrightarrows_n \bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1$.

Notons qu'il s'agit d'une égalité purement syntaxique, p.ex. en toute rigueur nous ne tenons même pas compte de la commutativité ou de l'associativité de $\vee$ ou $\wedge$. En pratique nous en tiendrons quand même compte pour simplifier les exemples. De même, rappelons que les expressions arithmétiques sont considérées à une équivalence près, p.ex. n et $n+1-1$ sont considérés comme désignant la même expression. Ainsi $\bigwedge_{i=1}^{n} p_i \,\&\, n \geq 1 \rightrightarrows_n \bigwedge_{i=1}^{n+1} p_i \,\&\, n+1 \geq 1$. Cette notion de bouclage est donc triviale, peu puissante, mais particulièrement facile à implémenter. Cette simplicité facilite aussi les raisonnements sur cette notion.

Proposition 5.14. *L'égalité à un décalage près est un raffinement de bouclage.*

Preuve. Supposons que $s_1 \rightrightarrows_n^k s_2$. Soit un modèle $\mathfrak{M}_1$ de s_1. Nous construisons un modèle $\mathfrak{M}_2$ de s_2 de la façon suivante : $\mathfrak{M}_{2\text{arith}}(n) \stackrel{\text{def}}{=} \mathfrak{M}_{1\text{arith}}(n) - k$, $\mathfrak{M}_{2\text{arith}}(p) \stackrel{\text{def}}{=} \mathfrak{M}_{1\text{arith}}(p)$ si $p \neq n$, et $\mathfrak{M}_{2\text{prop}} \stackrel{\text{def}}{=} \mathfrak{M}_{1\text{prop}}$. Pour prouver que $\mathfrak{M}_2$ est un modèle de s_2, il suffit de prouver que $\langle s_2 \rangle_{\mathfrak{M}_{2\text{arith}}} = \langle s_1 \rangle_{\mathfrak{M}_{1\text{arith}}}$ car $\mathfrak{M}_{2\text{prop}} = \mathfrak{M}_{1\text{prop}}$:

$$\begin{aligned}
\langle s_1 \rangle_{\mathfrak{M}_{1\text{arith}}} &= \langle s_2[n-k/n] \rangle_{\mathfrak{M}_{1\text{arith}}} && (\text{car } s_1 \rightrightarrows_n^k s_2)\\
&= \langle s_2[\mathfrak{M}_{1\text{arith}}(n-k)/n] \rangle_{\mathfrak{M}_{1\text{arith}}} &&\\
&= \langle s_2[\mathfrak{M}_{1\text{arith}}(n)-k/n] \rangle_{\mathfrak{M}_{1\text{arith}}} && (\text{car } k \in \mathbb{Z})\\
&= \langle s_2[\mathfrak{M}_{2\text{arith}}(n)/n] \rangle_{\mathfrak{M}_{1\text{arith}}} && (\text{définition de } \mathfrak{M}_{1\text{arith}})\\
&= \langle s_2[\mathfrak{M}_{2\text{arith}}(n)/n] \rangle_{\mathfrak{M}_{2\text{arith}}} && (\text{définition de } \mathfrak{M}_{2\text{arith}}(p) \text{ pour } p\text{n})\\
&= \langle s_2 \rangle_{\mathfrak{M}_{2\text{arith}}} &&
\end{aligned}$$

Par ailleurs, il est clair que $\mathfrak{M}_2 \prec_n \mathfrak{M}_1$. Finalement $\rightrightarrows_n$ est évidemment transitive. □

Les propriétés suivantes ne sont pas nécessairement utiles pour la suite. Elles sont mentionnées uniquement pour permettre de mieux appréhender la notion d'égalité à un décalage près.

[3] Dans [ACP09b], définition 10, nous avons introduit un autre raffinement de bouclage, plus puissant que l'égalité à un décalage près mais toujours décidable. Cependant en pratique l'égalité à un décalage près et ses extensions sont plus pratiques à manipuler formellement et plus efficaces à implémenter. De plus elle est suffisante pour les résultats théoriques présentés dans ce mémoire. Par conséquent nous ne présentons pas ici cet autre raffinement.

Proposition 5.15. *Soient n une variable entière, s_1, s_2 et s_3 des schémas. Les propriétés suivantes sont triviales :*

1. *si $s_1 \rightrightarrows_n s_2$ alors s_1 contient n ssi s_2 contient n;*
2. *si s_1 contient n et $s_1 \rightrightarrows_n s_2$ alors $s_2 \not\rightrightarrows_n s_1$ ($\rightrightarrows_n$ n'est donc pas symétrique) ;*
3. *si s_1 contient n alors $s_1 \not\rightrightarrows_n s_1$ ($\rightrightarrows_n$ n'est donc pas réflexive) ;*
4. *si s_1 contient n et s_2 ne contient pas n alors $s_1 \not\rightrightarrows_n s_2$ et $s_2 \not\rightrightarrows_n s_1$ ($\rightrightarrows_n$ n'est donc pas totale) ;*
5. *si s_1 ne contient pas n alors $s_1 \rightrightarrows_n s_2$ ssi $s_2 \rightrightarrows_n s_1$ ssi $s_1 = s_2$;*
6. *si $s_1 \rightrightarrows_n s_2$ et $s_3 \rightrightarrows_n s_2$ alors soit $s_1 \rightrightarrows_n s_3$, soit $s_3 \rightrightarrows_n s_1$, soit $s_1 = s_3$.*

5.2.2 Propriétés de clôture

Comme nous l'avons vu précédemment, notre but est d'identifier des classes décidables et, pour ce faire, de montrer que les ensembles de schémas potentiellement générés par une procédure sont finis modulo un raffinement de bouclage. Plus précisément, nous nous intéressons ici aux ensembles finis modulo l'égalité à un décalage près, que nous appellerons plus brièvement *ensembles finis à un décalage près* ou encore *ensembles $\rightrightarrows_n$-finis*. En particulier, nous aurons besoin de propriétés de *clôture* sur ces ensembles (au chapitre 6 et surtout au chapitre 9). En effet au chapitre 9 nous démontrerons la décidabilité d'une classe de schémas par induction sur la structure du schéma : sachant, par exemple, que le schéma s_1 génère l'ensemble de schémas S_1 et que s_2 génère S_2, alors $s_1 \wedge s_2$ génèrera l'ensemble $S = \{s'_1 \wedge s'_2 \mid s'_1 \in S_1, s'_2 \in S_2\}$. Nous aimerions alors avoir l'assurance que si S_1 et S_2 sont $\rightrightarrows_n$-finis alors S est aussi $\rightrightarrows_n$-fini. La suite de cette section est dédiée à la caractérisation d'une hypothèse permettant d'assurer ces propriétés de clôture.

Avant ceci, notons que nous pouvons aussi vouloir considérer des ensembles de motifs ou de formules arithmétiques finis à un décalage près, p.ex. pour pouvoir constituer l'ensemble $\{\bigwedge_{i|\delta} m \mid m \in M, \delta \in \Delta\}$ où M (resp. Δ) est un ensemble de motifs (resp. formules arithmétiques). De même, nous pouvons aussi vouloir considérer des ensembles d'expressions arithmétiques finis à un décalage près, p.ex. pour constituer l'ensemble de formules arithmétiques $\{e_1 = e_2 \mid e_1 \in E_1, e_2 \in E_2\}$ où E_1 et E_2 sont des ensembles d'expressions arithmétiques. Ainsi nous étendons d'abord la définition d'égalité à un décalage près à d'autres objets que les schémas.

Definition 5.16. *N'importe quel objet pouvant contenir une expression arithmétique linéaire est appelée un* shiftable. *Formellement : un shiftable est une expression arithmétique, un motif, une contrainte, un schéma ou un tuple de décalables.*

Soient deux shiftables d_1 et d_2, et n une variable entière. Nous disons que d_1 est égal à d_2 à un décalage près de n *ssi $d_1 = d_2[n - k/n]$ où $k > 0$. Nous notons $d_1 \rightrightarrows_n^k d_2$ ou juste $d_1 \rightrightarrows_n d_2$ si k est clair dans le contexte ou sans importance pour le discours.*

Example 5.17. *n, $n-2$ et $n-k$ sont des shiftables (où k est une variable entière). Nous avons $n - 2 \rightrightarrows_n^2 n$, mais $n - k \not\rightrightarrows_n n$. Notons que $n \rightrightarrows_n^1 n + 1$, c.-à-d. la substitution se fait modulo les opérations arithmétiques.*

De même $\forall i \cdot n \geq i$ et $\forall i \cdot n - 1 \geq i$ sont des shiftables, et : $\forall i \cdot n - 1 \geq i \rightrightarrows_n^1 \forall i \cdot n \geq i$. En revanche nous rappelons que la substitution tient compte des variables liées, donc $\forall n \cdot n - 1 \geq 1 \not\rightrightarrows_n \forall n \cdot n \geq 1$[4]

Comme nous considèrons aussi des tuples, la paire de schémas (p_n, p_{n-1}) est un shiftable. Nous avons $(p_n, p_{n-1}) \rightrightarrows_n^1 (p_{n+1}, p_n)$ mais $(p_n, p_{n-1}) \not\rightrightarrows_n (p_n, p_{n-1})$.

Les notions de $\rightrightarrows_n$-minimum, $\rightrightarrows_n$-connexité, $\rightrightarrows_n$-bien fondé et de finitude modulo $\rightrightarrows_n$ s'étendent de manière immédiate au shiftables.

Étudions maintenant les propriétés de clôture. Nous commençons par observer que si D_1 et D_2 sont $\rightrightarrows_n$-finis alors $D_1 \times D_2$ n'est pas nécessairement $\rightrightarrows_n$-fini. Par exemple, l'ensemble D_1 est bien $\rightrightarrows_n$-fini :

$$D_1 \stackrel{\text{def}}{=} \{p_n, p_{n-1}, p_{n-2}, \dots\}$$

[4]Nous mentionnons ce problème uniquement dans un souci de clarté : il ne devrait pas se poser en pratique car nous utiliserons en général des noms de variables différents pour les variables libres et liées.

ainsi que le singleton $D_2 \stackrel{\text{def}}{=} \{p_n\}$. En revanche $D_1 \times D_2$ ne l'est pas :

$$D_1 \times D_2 = \{(p_n, p_n), (p_{n-1}, p_n), (p_{n-2}, p_n), \dots\}$$

En effet, il est clair que, quels que soient k_1 et k_2 t.q. $k_1 \neq k_2$, $(p_{n-k_1}, p_n) \not\rightrightarrows_n (p_{n-k_2}, p_n)$, ainsi pour tout $k \in \mathbb{N}$, (p_{n-k}, p_n) est un minimum dans $D_1 \times D_2$, qui a donc une infinité de minima. Il est facile de voir qu'il en sera de même pour $\{m_1 \wedge m_2 \mid m_1 \in M_1, m_2 \in M_2\}$ ou $\{m_1 \vee m_2 \mid m_1 \in M_1, m_2 \in M_2\}$. Il nous faut donc restreindre les opérations de clôture.

Intuitivement, nous voyons que ce qui pose problème dans le contre-exemple précédent est le fait que nous pouvons prendre, dans l'ensemble produit, deux objets arbitrairement "éloignés". Nous allons donc essayer de nous restreindre à un éloignement borné, ce que nous formalisons maintenant :

Definition 5.18. *Un shiftable d est* à translation bornée *par rapport à une variable n ssi toutes les expressions arithmétiques apparaissant dans d et contenant n sont de la forme $n + k$ où $k \in \mathbb{Z}$.*

Le cas échéant, la déviation *de d par rapport à n, notée $\mathfrak{d}(d)$, est définie comme suit :*

$$\mathfrak{d}(d) \stackrel{def}{=} \max(\{|k_1 - k_2| \mid k_1, k_2 \in \mathbb{Z},\ n + k_1 \text{ et } n + k_2 \text{ apparaissent dans } d\})$$

si n apparaît dans d, et $\mathfrak{d}(d) \stackrel{def}{=} 0$ sinon. Nous notons $\mathfrak{B}_k$ l'ensemble $\{d \mid \mathfrak{d}(d) \leq k\}$.

Example 5.19. *$n + 1$, $\forall i \cdot i \leq n + 1$, p_n, $p_{n-1} \wedge \neg p_{n+2}$ et $\bigwedge_{i=1}^{n+2} p_i \wedge p_{n+1}$ sont à translation bornée par rapport à n. En revanche, $2n$, $n + i$, $\forall i \cdot 1 \leq n + i$, p_{2n}, p_{n+i}, $\bigvee_{i=1}^{2n} p_i$ et $(\bigvee_{i=1}^{n} p_i \wedge p_{n+i})$ ne le sont pas. Enfin $p_n \wedge p_{2i}$ est à translation bornée par rapport à n mais pas par rapport à i.*

Nous avons : $\mathfrak{d}(n + 1) = 0$, $\mathfrak{d}(p_{n-1} \wedge \neg p_{n+2}) = 3$, $\mathfrak{d}(\bigwedge_{i=1}^{n+2} p_i \wedge p_{n+1}) = 1$ et $\mathfrak{d}(1) = 0$.

Theorem 5.20. *Soient D_1 et D_2 deux ensembles de shiftables à translation bornée par rapport à une variable n. Si D_1 et D_2 sont $\rightrightarrows_n$-finis par rapport à n alors, pour tout $k \in \mathbb{N}$, $(D_1 \times D_2) \cap \mathfrak{B}_k$ est $\rightrightarrows_n$-fini.*

Example 5.21. *Dans le contre-exemple donné ci-dessus les déviations des schémas de $D_1 \times D_2$ étaient non bornées. C'est pourquoi le théorème ne s'appliquait pas. En revanche $(D_1 \times D_2) \cap \mathfrak{B}_2$ est égal à l'ensemble suivant :*

$$\{(p_n, p_n), (p_{n-1}, p_n), (p_{n-2}, p_n)\}$$

qui est fini, donc $\rightrightarrows_n$-fini.

De même, $D_1 \times D_1$ n'est pas fini à un décalage près :

$$D_1 \times D_1 = \left\{ \begin{array}{c} (p_n, p_n), (p_{n-1}, p_n), (p_{n-2}, p_n), \dots \\ (p_n, p_{n-1}), (p_{n-1}, p_{n-1}), (p_{n-2}, p_{n-1}), \dots \\ (p_{n-1}, p_{n-2}), (p_{n-2}, p_{n-2}), (p_{n-3}, p_{n-2}), \dots \\ \dots \end{array} \right\}$$

Mais :

$$(D_1 \times D_2) \cap \mathfrak{B}_1 = \left\{ \begin{array}{c} (p_n, p_n), (p_{n-1}, p_n) \\ (p_n, p_{n-1}), (p_{n-1}, p_{n-1}), (p_{n-2}, p_{n-1}) \\ (p_{n-1}, p_{n-2}), (p_{n-2}, p_{n-2}), (p_{n-3}, p_{n-2}) \\ \dots \end{array} \right\}$$

est bien fini à un décalage près (l'ensemble est présenté de sorte à ce que les trois "colonnes" correspondent aux composantes connexes).

Preuve (du théorème 5.20). Nous montrons que $(D_1 \times D_2) \cap \mathfrak{B}_k$ a un ensemble fini de composantes connexes, et que toutes sont bien fondées. Plus précisément, nous montrons que chaque composante connexe a l'une des formes suivantes, où $\hat{d_1} \in D_1/\rightrightarrows_n$, $\hat{d_2} \in D_2/\rightrightarrows_n$, $\delta \in [-k; k]$ et $\max(d) \stackrel{\text{def}}{=} \max(\{p \in \mathbb{Z} \mid n + p \text{ apparaît dans } d\})$:

$$\begin{cases} \{\hat{d_1}\} \times \{\hat{d_2}\} & \text{si ni } \hat{d_1}, \text{ ni } \hat{d_2} \text{ ne contiennent } n \\ \{\hat{d_1}\} \times D_2 & \text{si } \hat{d_1} \text{ ne contient pas } n \text{ et } \hat{d_2} \text{ contient } n \\ D_1 \times \{\hat{d_2}\} & \text{si } \hat{d_1} \text{ contient } n \text{ et } \hat{d_2} \text{ ne contient pas } n \\ \left\{(d_1, d_2) \in D_1 \times D_2 \middle| \begin{array}{c} d_1 \rightrightarrows_n \hat{d_1} \\ d_2 \rightrightarrows_n \hat{d_2} \\ \max(d_1) - \max(d_2) = \delta \end{array} \right\} & \text{si } \hat{d_1} \text{ et } \hat{d_2} \text{ contiennent tous deux } n \end{cases}$$

Ainsi comme il existe une quantité finie de $\hat{d_1} \in D_1/\rightrightarrows_n$, $\hat{d_2} \in D_2/\rightrightarrows_n$ et $\delta \in [-k;k]$, il en va de même des composantes connexes de $(D_1 \times D_2) \cap \mathfrak{B}_k$. Nous montrons les résultats suivants :

- tout ensemble ayant l'une des formes ci-dessus est connexe;
- un tel ensemble est maximal par rapport à la connexité, c'est donc bien une composante connexe;
- un tel ensemble est bien fondé;
- toute composante connexe a l'une des formes ci-dessus.

Nous aurons donc bien montré qu'il existe une quantité finie de composantes connexes bien fondées.

Il est trivial que $\{\hat{d_1}\} \times \{\hat{d_2}\}$ est connexe car il ne contient qu'un élément. D_1 et D_2 sont connexes donc il est facile de voir qu'il en va de même de $\{\hat{d_1}\} \times D_2$ et $D_1 \times \{\hat{d_2}\}$. Supposons à présent que $\hat{d_1}$ et $\hat{d_2}$ contiennent chacun n et montrons que $\left\{(d_1, d_2) \in D_1 \times D_2 \middle| \begin{smallmatrix} d_1 \rightrightarrows_n \hat{d_1} \\ d_2 \rightrightarrows_n \hat{d_2} \\ \max(d_1) - \max(d_2) = \delta \end{smallmatrix}\right\}$ est connexe. Nous montrons donc que pour tous $d_1, d_1' \in D_1$ et $d_2, d_2' \in D_2$ tels que :

$$\begin{array}{lll} d_1 \rightrightarrows_n \hat{d_1} & d_2 \rightrightarrows_n \hat{d_2} & \max(d_1) - \max(d_2) = \delta \\ d_1' \rightrightarrows_n \hat{d_1} & d_2' \rightrightarrows_n \hat{d_2} & \max(d_1') - \max(d_2') = \delta \end{array}$$

nous avons soit $(d_1, d_2) \rightrightarrows_n (d_1', d_2')$, soit $(d_1', d_2') \rightrightarrows_n (d_1, d_2)$, soit $(d_1', d_2') = (d_1, d_2)$. Par connexité de D_1, nous avons soit $d_1 \rightrightarrows_n^p d_1'$, soit $d_1' \rightrightarrows_n^p d_1$, soit $d_1' = d_1$ pour un certain $p > 0$. Supposons que nous ayons $d_1 \rightrightarrows_n^p d_1'$, alors $\max(d_1) - \max(d_1') = p$. De plus, comme $\max(d_1) - \max(d_2) = \delta$ et $\max(d_1') - \max(d_2') = \delta$, il est facile de déduire $\max(d_2) - \max(d_2') = p$. Or, par connexité de D_2, nous avons soit $d_2 \rightrightarrows_n d_2'$, soit $d_2' \rightrightarrows_n d_2$, soit $d_2' = d_2$. Donc, comme $\max(d_2) - \max(d_2') = p > 0$, nous ne pouvons qu'avoir $d_2 \rightrightarrows_n d_2'$ et, plus précisément, $d_2 \rightrightarrows_n^p d_2'$. Par conséquent nous avons bien $(d_1, d_2) \rightrightarrows_n (d_1', d_2')$. Enfin le cas $d_1' \rightrightarrows_n^p d_1$ est symétrique et le cas $d_1 = d_1'$ entraîne facilement $d_2 = d_2'$ en prenant $p = 0$ dans les équations précédentes.

Montrons maintenant que ces ensembles sont maximaux pour la connexité, c.-à-d. si un élément quelconque de $(D_1 \times D_2) \cap \mathfrak{B}_k$ est égal à un décalage près à un élément de l'un de ces ensembles, alors il appartient à cet ensemble. D'après la propriété 5.15 (5), si $(e_1, e_2) \rightrightarrows_n (\hat{d_1}, \hat{d_2})$ et ni $\hat{d_1}$ ni $\hat{d_2}$ ne contiennent n alors toute paire (e_1, e_2) est égale à $(\hat{d_1}, \hat{d_2})$. Il est donc trivial dans ce cas que $\{\hat{d_1}\} \times \{\hat{d_2}\}$ est maximal. De même si $\hat{d_1}$ ne contient pas n, alors pour toute paire $(e_1, e_2) \in D_1 \times D_2$ t.q. $(e_1, e_2) \rightrightarrows_n (\hat{d_1}, d_2)$ où $d_2 \in D_2$, nous avons $e_1 = \hat{d_1}$. Donc nous avons bien $e_1 \in \{\hat{d_1}\}$ (et $e_2 \in D_2$ par définition). La preuve est symétrique pour $D_1 \times \{\hat{d_2}\}$. Enfin soient $e_1 \in D_1$ et $e_2 \in D_2$ tels que $(e_1, e_2) \rightrightarrows_n (d_1, d_2)$ où $d_1 \rightrightarrows_n \hat{d_1}$, $d_2 \rightrightarrows_n \hat{d_2}$, $\max(d_1) - \max(d_2) = \delta$ et $\hat{d_1}$ et $\hat{d_2}$ contiennent n. On a $(e_1, e_2) \rightrightarrows_n (\hat{d_1}, \hat{d_2})$ par transitivité de $\rightrightarrows_n$. Donc nous avons bien $e_1 \rightrightarrows_n \hat{d_1}$ et $e_2 \rightrightarrows_n \hat{d_2}$. Il reste à prouver que $\max(e_1) - \max(e_2) = \delta$. Soit $p \in \mathbb{N}$ tel que $(e_1, e_2) \rightrightarrows_n^p (d_1, d_2)$. On a alors $e_1 \rightrightarrows_n^p d_1$ et $e_2 \rightrightarrows_n^p d_2$. Donc à chaque expression de la forme $n + q$ dans e_1 correspond une expression $n + q + p$ dans d_1 donc $\max(e_1) = \max(d_1) + p$. De même $\max(e_2) = \max(d_2) + p$. Il est donc clair que $\max(e_1) - \max(e_2) = \max(d_1) - \max(d_2) = \delta$.

Nous montrons maintenant que ces ensembles sont bien fondés, c.-à-d. qu'il contiennent un minimum. Dans le cas où ni $\hat{d_1}$ ni $\hat{d_2}$ ne contiennent n c'est trivial car l'ensemble est un singleton. Dans le cas où seul d_1 (resp. d_2) contient n, c'est une conséquence triviale de la bonne fondation de D_1 (resp. D_2). Dans le dernier cas, supposons que D ne contienne pas de minimum, c.-à-d. : pour tout $(d_1, d_2) \in D$, il existe $(d_1', d_2') \in D$ t.q. $(d_1, d_2) \rightrightarrows_n (d_1', d_2')$. Dans ce cas, par la remarque précédente : pour tout $d_1 \in D_1$, il existe $d_1' \in D$ t.q. $d_1 \rightrightarrows_n d_1'$, ce qui contredit la bonne fondation de D_1.

Vues les propriétés précédentes tous ces ensembles sont bien des composantes connexes, donc montrer que toute composante connexe est bien l'un de ces ensembles revient à montrer que tout élément de $(D_1 \times D_2) \cap \mathfrak{B}_k$ appartient à l'un d'eux. Soit donc $(d_1, d_2) \in (D_1 \times D_2) \cap \mathfrak{B}_k$. Si ni d_1 ni d_2 ne contiennent n alors $(d_1, d_2) \in \{\hat{d_1}\} \times \{\hat{d_2}\}$ où $\hat{d_1} = d_1$ et $\hat{d_2} = d_2$. Si d_2 contient n mais pas d_1 alors $(d_1, d_2) \in \{\hat{d_1}\} \times D_2$ où $\hat{d_1} = d_1$. De même si d_1 contient n mais pas d_2 alors $(d_1, d_2) \in D_1 \times \{\hat{d_2}\}$ où $\hat{d_2} = d_2$. Enfin si tous deux contiennent n alors il existe bien $\hat{d_1} \in D_1/\rightrightarrows_n$ et $\hat{d_2} \in D_2/\rightrightarrows_n$ tels que $d_1 \rightrightarrows_n \hat{d_1}$ et $d_2 \rightrightarrows_n \hat{d_2}$. De plus comme $(d_1, d_2) \in (D_1 \times D_2) \cap \mathfrak{B}_k$, nous savons que $\mathfrak{d}((d_1, d_2)) \leq k$, donc il existe bien $\delta \in [-k;k]$ t.q. $\max(d_1) - \max(d_2) = \delta$. □

Les corollaires suivants sont triviaux. Ils seront utilisés dans les preuves de terminaison :

Corollary 5.22. *Soient M_1 et M_2 deux ensembles de motifs à translation bornée par rapport à une variable n. Si M_1 et M_2 sont $\rightrightarrows_n$-finis par rapport à n alors, pour tout $k \in \mathbb{N}$ et pour tout $\star \in \{\wedge, \vee\}$: $\{m_1 \star m_2 \mid m_1 \in M_1, m_2 \in M_2\} \cap \mathfrak{B}_k$ est $\rightrightarrows_n$-fini.*

Corollary 5.23. *Soit S (resp. Δ) un ensemble de schémas (resp. formules arithmétiques) à translation bornée par rapport à une variable n. Si S et Δ sont $\rightrightarrows_n$-finis par rapport à n alors, pour tout $k \in \mathbb{N}$ et pour tout $\Pi \in \{\bigwedge, \bigvee\}$: $\{\Pi_{i|\delta}\, m_s \,\&\, c_s \mid s \in S, \delta \in \Delta\} \cap \mathfrak{B}_k$ est $\rightrightarrows_n$-fini.*

Corollary 5.24. *Soient E_1 et E_2 deux ensembles d'expressions arithmétiques à translation bornée par rapport à une variable n. Si E_1 et E_2 sont $\rightrightarrows_n$-finis par rapport à n alors, pour tout $k \in \mathbb{N}$, l'ensemble de formules arithmétiques $\{e_1 = e_2 \mid e_1 \in E_1, e_2 \in E_2\} \cap \mathfrak{B}_k$ est $\rightrightarrows_n$-fini.*

En pratique, nous utiliserons surtout le corollaire suivant (nous disons qu'une suite $(d_k)_{k\in\mathbb{N}}$ de shiftables est $\rightrightarrows_n$-finie ssi l'ensemble $\bigcup\{d_k \mid k \in \mathbb{N}\}$ l'est) :

Corollary 5.25. *Soient deux suites $\rightrightarrows_n$-finies $(d_k)_{k\in\mathbb{N}}$ et $(e_k)_{k\in\mathbb{N}}$ de shiftables à translation bornée. Supposons qu'il existe $k_0 \in \mathbb{N}$ et $q_1, q_2 \in \mathbb{Z}$ tel que pour tout $k \geq k_0$, toute expression de d_k ou de e_k contenant n a la forme $n - k + q$ où $q \in [q_1; q_2]$. Alors la suite de paires $((d_k, e_k))_{k\in\mathbb{N}}$ est $\rightrightarrows_n$-finie.*

Preuve. La forme des expressions implique que la déviation de (d_k, e_k) est inférieure ou égale à $q_2 - q_1$. On peut donc appliquer le théorème 5.20. □

À titre d'exemple, il est facile d'organiser l'ensemble $(D_1 \times D_1) \cap \mathfrak{B}_1$, où D_1 est celui de l'exemple 5.21, sous la forme d'une suite ayant la propriété du corollaire 5.25 avec $[q_1; q_2] = [-1; 1]$.

5.3 Extensions

En pratique, l'égalité à un décalage près n'est pas suffisante pour détecter des cycles. Cependant elle constitue une base simple et facile à améliorer : nous présentons maintenant des "extensions" de raffinements, c.-à-d. des méthodes qui, étant donné un raffinement de bouclage quelconque, permettent de construire un raffinement plus puissant.

Considérons l'exemple récurrent :

$$s \stackrel{\text{def}}{=} p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$$

Comme nous l'avons vu en 4.4.2, Stab va générer[5] le schéma :

$$s' \stackrel{\text{def}}{=} p_0 \wedge (\bigwedge_{i=0}^{n-2} p_i \Rightarrow p_{i+1}) \wedge \neg p_{n-1} \wedge \neg p_n \,\&\, n - 1 \geq 0 \wedge n \geq 0$$

Il est facile de voir que s' boucle sur s (par rapport à l'ordre standard), en revanche nous n'avons pas $s' \rightrightarrows_n s$. En fait nous avons :

$$s' = m_s[n - 1/n] \wedge \neg p_n \,\&\, c_s[n - 1/n] \wedge n \geq 0$$

En quelque sorte $\neg p_n$ et $n \geq 0$ sont "de trop". Mais nous pouvons voir que $\neg p_n$ est *pur* dans s' (c.-à-d. pour tout environnement ρ, $\langle \neg p_n \rangle_\rho$ est pur dans $\langle s' \rangle_\rho$: $\langle p_n \rangle_\rho$ n'apparaît pas positivement dans $\langle s' \rangle_\rho$). Nous pouvons donc évaluer ce littéral à true, ce qui donne :

$$s'' \stackrel{\text{def}}{=} p_0 \wedge (\bigwedge_{i=0}^{n-2} p_i \Rightarrow p_{i+1}) \wedge \neg p_{n-1} \,\&\, n - 1 \geq 0 \wedge n \geq 0 = s[n - 1/n] \,\&\, n \geq 0$$

[5] En fait Stab va générer l'*ensemble* :

$$\{p_0, (\bigwedge_{i=0}^{n-2} p_i \Rightarrow p_{i+1}), \neg p_{n-1}, \neg p_n, n - 1 \geq 0, n \geq 0\}$$

mais, comme expliqué au début de 5.1, dans le but d'avoir une détection de cycle indépendante de la procédure, nous préférons nous ramener à la notion de schéma. Dans le cas de Stab la façon d'obtenir un schéma à partir d'un ensemble étiquetant un nœud est décrite au début de 4.2 : *grosso modo*, nous prenons la conjonction de tous les éléments de l'ensemble.

Il ne reste plus qu'à éliminer $n \geq 0$ pour avoir une égalité à un décalage près. Or nous avons trivialement : $n - 1 \geq 0 \models n \geq 0$, c.-à-d. la contrainte $n \geq 0$ est *redondante*, nous pouvons donc la supprimer et obtenir ainsi $s''' \stackrel{\text{def}}{=} s[n-1/n]$, et il est alors clair que $s''' \rightrightarrows_n s$.

Nous allons maintenant généraliser ces deux simplifications en deux extensions :

- l'*extension du littéral pur*
- l'*extension de la contrainte équivalente*

L'intérêt de ces extensions réside dans le fait qu'elles sont décidables (parfois sous certaines conditions). En outre, combinées avec l'égalité à un décalage près, elles nous permettront d'obtenir tous les résultats de décidabilité dont nous aurons besoin dans ce mémoire.

5.3.1 Pure literal extension

Comme nous l'avons rappelé à la section 2.1 (définition 2.5), un littéral propositionnel est *pur* dans une formule propositionnelle ssi sa négation n'apparaît pas positivement dans cette formule. Dans ce cas nous pouvons évaluer le littéral à true dans la formule en préservant la satisfaisabilité. Cette règle est très courante en déduction automatique propositionnelle (typiquement dans DPLL [DLL62]), bien que souvent omise de nos jours car le temps passé à détecter les littéraux purs compense rarement le gain dû à leur élimination. Dans le cas des schémas, cependant, son omission entraîne généralement la non-terminaison.

La notion de littéral pur doit être adaptée aux schémas. Contrairement au cas propositionnel, nous ne pouvons nous satisfaire d'une notion d'appartenance purement syntaxique. Par exemple, p_{n+1} est pur dans $s_1 \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} \neg p_i$ car $\neg p_{n+1}$ ne peut pas apparaître dans s_1 (définition 3.16), c.-à-d. ne peut pas apparaître dans $p_1, \ldots, p_n$ quel que soit $n \in \mathbb{N}$. En revanche nous ne voulons pas que p_{n+1} soit pur dans $s_2 \stackrel{\text{def}}{=} (\bigwedge_{i=1}^{2n} \neg p_i) \,\&\, n \geq 1$, bien que $\neg p_{n+1}$ n'apparaisse toujours pas, syntaxiquement, dans ce schéma. En fait ce littéral *peut apparaître implicitement* dans s_2 (définition 3.16). En effet il faut tenir compte du fait que $\neg p_i$ est le complément de p_{n+1} si $i = n + 1$. Cette condition ne pouvait être réalisée dans s_1 car i était compris entre 1 et n, mais dans s_2 c'est possible (si $n > 0$) car i est compris entre 1 et $2n$.

En fin de compte nous donnons simplement la définition (sémantique) suivante :

Definition 5.26 (Littéral pur)**.** *Un littéral l est* pur *dans un schéma s ssi pour tout environnement ρ, $\langle l \rangle_\rho$ est pur au sens propositionnel dans $\langle s \rangle_\rho$.*

Il est facile de voir que l est pur dans un schéma s ssi le complémentaire de l ne peut pas apparaître dans s, c.-à-d. $l^{\text{c}} \not\sqsubset_\diamond s$. Par décidabilité de $\sqsubset_\diamond$ (corollaire 3.19), il est donc décidable de savoir si un littéral est pur dans un schéma.

A présent, nous formalisons la notion de remplacement d'un littéral par true. La substitution $s[m/l]$ d'un littéral l par un motif m est définie de façon standard, il s'agit d'une substitution syntaxique, p.ex. : $(\neg p_1 \wedge \bigvee_{i=1}^{n} p_i)[\top/p_1] = \neg\top \wedge \bigvee_{i=1}^{n} p_i$ et non pas $s \stackrel{\text{def}}{=} \neg\top \wedge (\top \wedge \bigvee_{i=2}^{n} p_i)$ comme nous pourrions peut-être nous y attendre. Notons que ce serait en fait une erreur d'obtenir s car nous ne savons pas si $n \geq 1$ ou non. Cependant $(\neg p_1 \wedge \bigvee_{i=1}^{n} p_i \,\&\, n \geq 1)[\top/p_1] = \neg\top \wedge \bigvee_{i=1}^{n} p_i \,\&\, n \geq 1$ (c.-à-d. même en sachant que $n \geq 1$, nous nous tenons à une substitution syntaxique).

Le lemme suivant (trivial) permet alors de prouver le résultat clé sur les littéraux purs (proposition 5.28).

Lemma 5.27. *Pour tout schéma s, tout littéral l et tout environnement ρ :*

$$\langle s \rangle_\rho[\top/\langle l \rangle_\rho] = \langle s[\top/l] \rangle_\rho[\top/\langle l \rangle_\rho]$$

Proposition 5.28. *Soit l un littéral pur dans un schéma s.*

Si s a un modèle $\mathfrak{M}$ alors $s[\top/l]$ a un modèle $\mathfrak{N}$ t.q. $\mathfrak{M}_{\text{arith}}(n) = \mathfrak{N}_{\text{arith}}(n)$ pour tout paramètre n de s.

Réciproquement, si $s[\top/l]$ a un modèle $\mathfrak{M}$ alors s a un modèle $\mathfrak{N}$ t.q. pour tout paramètre n de s : $\mathfrak{N}_{\text{arith}}(n) = \mathfrak{M}_{\text{arith}}(n)$.

Preuve. Soit $\mathfrak{M}$ un modèle de s. $\langle s \rangle_{\mathfrak{M}_{\text{arith}}}$ est donc satisfaisable (au sens propositionnel). Par ailleurs comme l est pur dans s, $\langle l \rangle_{\mathfrak{M}_{\text{arith}}}$ est pur dans $\langle s \rangle_{\mathfrak{M}_{\text{arith}}}$. Donc, d'après le théorème 2.6, $\langle s \rangle_{\mathfrak{M}_{\text{arith}}}[\top/\langle l \rangle_{\mathfrak{M}_{\text{arith}}}]$ est satisfaisable. D'après la proposition précédente, nous avons :

$$\langle s \rangle_{\mathfrak{M}_{\text{arith}}}[\top/\langle l \rangle_{\mathfrak{M}_{\text{arith}}}] = \langle s[\top/l] \rangle_{\mathfrak{M}_{\text{arith}}}[\top/\langle l \rangle_{\mathfrak{M}_{\text{arith}}}]$$

Par conséquent $\langle s[\top/l]\rangle_{\mathfrak{M}_{\text{arith}}}[\top/\langle l\rangle_{\mathfrak{M}_{\text{arith}}}]$ est aussi satisfaisable. Par ailleurs comme $\langle l\rangle_{\mathfrak{M}_{\text{arith}}}$ est pur dans $\langle s\rangle_{\mathfrak{M}_{\text{arith}}}$, $\langle l\rangle_{\mathfrak{M}_{\text{arith}}}$ est aussi pur dans $\langle s[\top/l]\rangle_{\mathfrak{M}_{\text{arith}}}$, donc la satisfaisabilité de $\langle s[\top/l]\rangle_{\mathfrak{M}_{\text{arith}}}[\top/\langle l\rangle_{\mathfrak{M}_{\text{arith}}}]$ est équivalente à la satisfaisabilité de $\langle s[\top/l]\rangle_{\mathfrak{M}_{\text{arith}}}$ (toujours par le théorème 2.6). Donc $\langle s[\top/l]\rangle_{\mathfrak{M}_{\text{arith}}}$ est satisfaisable.

Nous pouvons donc définir $\mathfrak{N}_{\text{prop}}$ comme l'un des modèles de $\langle s[\top/l]\rangle_{\mathfrak{M}_{\text{arith}}}$ et $\mathfrak{N}_{\text{arith}}$ comme $\mathfrak{M}_{\text{arith}}$. $\mathfrak{N}$ est clairement un modèle de $s[\top/l]$ et nous avons évidemment $\mathfrak{M}_{\text{arith}}(n) = \mathfrak{N}_{\text{arith}}(n)$ pour tout paramètre n de s.

La preuve de la réciproque est symétrique. □

L'étape suivante est de substituer *tous les litteraux purs* d'un schéma par $\top$. Il y a un léger problème par rapport au cas propositionnel : il y a une infinité de littéraux purs qui peuvent apparaître dans un schéma. Par exemple pour tout $i \in [1; 2n]$, p_i peut apparaître implicitement dans $\neg p_n \vee \bigvee_{i=1}^{2n} p_i$: auquel cas il faut éliminer une infinité de littéraux, ce qui ne terminerait pas en général. Cependant comme nous considérons une forme syntaxique de substitution des littéraux, la substitution par $\top$ de littéraux pouvant apparaître implicitement ne modifiera pas le schéma (dans l'exemple ci-dessus, seule la substitution de p_n peut modifier le schéma). Ainsi nous pouvons nous restreindre aux littéraux appartenant explicitement au schéma, qui sont, eux, en quantité finie. Nous notons $\text{pur}(s)$ le schéma s dont tous les littéraux purs ont été substitués par $\top$, $\text{pur}(s)$ est calculable d'après ce qui précède.

Definition 5.29 (Extension du littéral pur)**.** *Soit $\triangleright$ un raffinement de bouclage par rapport à l'ordre standard. Nous appelons* extension du littéral pur *de $\triangleright$, la relation $\triangleright'$ définie par : $s_1 \triangleright' s_2 \Leftrightarrow \text{pur}(s_1) \triangleright \text{pur}(s_2)$.*

Proposition 5.30. *L'extension du littéral pur d'un raffinement de bouclage par rapport à l'ordre standard est un raffinement de bouclage par rapport à l'ordre standard.*

Preuve. Soient s_1 et s_2 deux schémas tels que $s_1 \triangleright' s_2$, c.-à-d. $\text{pur}(s_1) \triangleright \text{pur}(s_2)$. Soit $\mathfrak{M}_1$ un modèle de s_1. D'après la proposition 5.28, il existe un modèle $\mathfrak{M}'_1$ de $\text{pur}(s_1)$ t.q. $\mathfrak{M}'_{1\text{arith}}(n) = \mathfrak{M}_{1\text{arith}}(n)$ pour tout paramètre n de s_1. Donc, comme $\triangleright$ est un raffinement de bouclage par rapport à l'ordre standard, il existe un modèle $\mathfrak{M}'_2$ de $\text{pur}(s_2)$ t.q. $\mathfrak{M}'_{2\text{arith}}(n) < \mathfrak{M}'_{1\text{arith}}(n)$ pour un paramètre n de $\text{pur}(s_1)$ (et donc de s_1) et $\mathfrak{M}'_{2\text{arith}}(p) \leq \mathfrak{M}'_{1\text{arith}}(p)$ pour tout autre paramètre p. Alors, toujours d'après la proposition 5.28, il existe un modèle $\mathfrak{M}_2$ de s_2 t.q. $\mathfrak{M}_{2\text{arith}}(n) = \mathfrak{M}'_{2\text{arith}}(n)$ pour tout paramètre n de s_2. Ainsi, nous avons construit à partir d'un modèle $\mathfrak{M}_1$ de s_1 un modèle $\mathfrak{M}_2$ de s_2, tel que $\mathfrak{M}_2 \prec_n \mathfrak{M}_1$. □

Example 5.31. *Nous savons que $\bigwedge_{i=1}^{n-1} p_i$ est égal à $\bigwedge_{i=1}^{n} p_i$ à un décalage près. Soit $s \stackrel{def}{=} \neg p_n \wedge \bigwedge_{i=1}^{n-1} p_i$, s n'est pas égal à $\bigwedge_{i=1}^{n} p_i$ à un décalage près. Mais le littéral $\neg p_n$ est pur dans s. Par conséquent $\text{pur}(s) = \bigwedge_{i=1}^{n-1} p_i$. Donc s boucle sur $\bigwedge_{i=1}^{n} p_i$ avec l'extension du littéral pur de l'égalité à un décalage près.*

De même, s boucle sur $p_{n+1} \wedge \bigwedge_{i=1}^{n} p_i$ car p_{n+1} est pur dans $p_{n+1} \wedge \bigwedge_{i=1}^{n} p_i$.

5.3.2 Equivalent constraint extension

Il est fréquent qu'une procédure (Stab en particulier, mais aussi Dpll* comme nous le verrons au chapitre 8) génère des séquences divergentes de contraintes de la forme suivante :

$$\begin{array}{l} n > 0 \\ n > 0 \wedge n - 1 > 0 \\ n > 0 \wedge n - 1 > 0 \wedge n - 2 > 0 \\ \ldots \end{array}$$

Beaucoup d'informations sont redondantes dans ces contraintes, par exemple $n > 0$ est subsumé par $n - 1 > 0$. Comme nous l'avons vu ces informations peuvent être un obstacle à la détection de cycle. Les deux extensions suivantes visent à supprimer ces informations.

Definition 5.32 (Extension de la contrainte équivalente)**.** *Soit $\triangleright$ un raffinement de bouclage par rapport à l'ordre standard. Nous appelons* extension de la contrainte équivalente *de $\triangleright$, la relation $\triangleright'$ définie par : $s_1 \triangleright' s_2$ ssi il existe s'_1, s'_2 tels que $m_{s'_i} = m_{s_i}$, $c_{s'_i} \equiv c_{s_i}$ $(i \in \{1, 2\})$ et $s'_1 \triangleright s'_2$.*

Example 5.33. *Soient* $s_1 \stackrel{def}{=} \top \,\&\, 1 \leq n+2 \wedge n \geq 0$ *et* $s_2 \stackrel{def}{=} \top \,\&\, 1 \leq n+3$. *Alors* s_1 *boucle sur* s_2 *avec l'extension de la contrainte équivalente de l'égalité à un décalage près (c.-à-d. que* $\triangleright$ *est l'égalité à un décalage près). En effet, posons* $s'_1 \stackrel{def}{=} \top \,\&\, 1 \leq n+2$ *et* $s'_2 \stackrel{def}{=} s_2$. *Alors* s'_1 *est bien égal à* s'_2 *à un décalage près et :*

$$\begin{aligned} m_{s_2} &= m_{s'_2} = \top \\ c_{s'_2} &= c_{s_2} = 1 \leq n+3 \\ m_{s_1} &= m_{s'_1} = \top \\ c_{s_1} &= 1 \leq n+2 \wedge n \geq 0 \\ c_{s'_1} &= 1 \leq n+2 \end{aligned}$$

Il est facile de voir que $c_{s'_1} \equiv c_{s_1}$ *et il est trivial que* $c_{s'_2} \equiv c_{s_2}$, *d'où le résultat.*

Proposition 5.34. *L'extension de la contrainte équivalente d'un raffinement de bouclage par rapport à l'ordre standard est un raffinement de bouclage par rapport à l'ordre standard.*

Preuve. Soient s_1 et s_2 tels que $s_1 \rhd' s_2$, où $\rhd'$ est l'extension de la contrainte équivalente pour un raffinement de bouclage $\rhd$. Il existe donc s'_1 et s'_2 tels que $m_{s'_i} = m_{s_i}$, $c_{s'_i} \equiv c_{s_i}$ $(i \in \{1,2\})$ et $s'_1 \rhd s'_2$. Soit un modèle $\mathfrak{M}_1$ de s_1. Comme $m_{s'_1} = m_{s_1}$ et $c_{s'_1} \equiv c_{s_1}$, $\mathfrak{M}_1$ est aussi un modèle de s'_1. Donc, comme $\rhd$ est un raffinement de bouclage par rapport à l'ordre standard, il existe un modèle $\mathfrak{M}_2$ de s'_2 t.q. $\mathfrak{M}_{2\text{arith}}(n) < \mathfrak{M}_{1\text{arith}}(n)$ pour un paramètre n de s'_1 (et donc de s_1) et $\mathfrak{M}_{2\text{arith}}(p) \leq \mathfrak{M}_{1\text{arith}}(p)$ pour tout autre paramètre p. Alors, comme $m_{s'_2} = m_{s_2}$ et $c_{s'_2} \equiv c_{s_2}$, $\mathfrak{M}_2$ est aussi un modèle de s_2. Ainsi, nous avons construit à partir d'un modèle $\mathfrak{M}_1$ de s_1 un modèle $\mathfrak{M}_2$ de s_2, tel que $\mathfrak{M}_2 \prec_n \mathfrak{M}_1$. □

L'inconvénient de cette extension est qu'elle n'est pas nécessairement décidable : en effet, savoir si $s_1 \rhd' s_2$ nécessite de rechercher s'_1 et s'_2 avec les propriétés voulues. Il n'y a, a priori, aucune raison pour que cette recherche termine. En revanche, si nous prenons pour $\rhd$ l'égalité à un décalage près, alors cette extension est décidable :

Proposition 5.35. *Soit* $\rhd'$ *l'extension de la contrainte équivalente pour l'égalité à un décalage près. Pour tous schémas* s_1 *et* s_2 *nous avons* $s_1 \rhd' s_2$ *ssi :*

$$\begin{cases} m_{s_1} \rightrightarrows_n^k m_{s_2} \text{ pour un } k > 0 \text{ et } c_{s_1} \Leftrightarrow c_{s_2}[n - k/n] & \text{si } m_{s_1} \text{ contient } n \\ m_{s_1} = m_{s_2} \text{ et } \exists k > 0 (c_{s_1} \Leftrightarrow (c_{s_2}[n - k/n])) & \text{si } m_{s_1} \text{ ne contient pas } n \end{cases}$$

Preuve. Supposons $s_1 \rhd' s_2$. Il existe donc s'_1 et s'_2 tels que $m_{s'_i} = m_{s_i}$, $c_{s'_i} \equiv c_{s_i}$ $(i \in \{1,2\})$ et $s'_1 \rightrightarrows_n s'_2$. Comme $s'_1 \rightrightarrows_n s'_2$, nous avons $m_{s'_1} \rightrightarrows_n m_{s'_2}$ et donc $m_{s_1} \rightrightarrows_n m_{s_2}$ (et $m_{s_1} = m_{s_2}$ si m_{s_1} ne contient pas n). Par conséquent il existe un $k > 0$ t.q. $m_{s_1} \rightrightarrows_n^k m_{s_2}$. De même, nous avons $c_{s'_1} \rightrightarrows_n^k c_{s'_2}$, donc $c_{s'_1} = c_{s'_2}[n - k/n]$. Par conséquent, comme $c_{s'_1} \equiv c_{s_1}$ et $c_{s'_2} \equiv c_{s_2}$, nous avons $c_{s_1} \equiv c_{s_2}[n - k/n]$. Donc les formules $c_{s_1} \Leftrightarrow c_{s_2}[n - k/n]$ et $\exists k > 0 (c_{s_1} \Leftrightarrow c_{s_2}[n - k/n])$ sont toutes deux valides. Ce qui permet de conclure, que m_{s_1} contienne n ou pas.

Nous montrons maintenant la réciproque. Supposons que m_{s_1} contienne n, que $m_{s_1} \rightrightarrows_n^k m_{s_2}$ pour un $k > 0$ et que $c_{s_1} \Leftrightarrow c_{s_2}[n - k/n]$. On pose $s'_1 \stackrel{\text{def}}{=} m_{s_1} \,\&\, c_{s_2}[n - k/n]$. On a alors $s'_1 \rightrightarrows_n s_2$. Comme, d'autre part, $m_{s_1} = m_{s'_1}$ et $c_{s_1} \equiv c_{s_2}[n - k/n]$, nous obtenons bien $s_1 \rhd' s_2$. Supposons maintenant que m_{s_1} ne contienne pas n, que $m_{s_1} = m_{s_2}$ et que $\exists k > 0 (c_{s_1} \Leftrightarrow (c_{s_2}[n - k/n]))$. Soit donc $k > 0$ vérifiant cette dernière assertion. On pose $s'_1 \stackrel{\text{def}}{=} m_{s_1} \,\&\, c_{s_2}[n - k/n]$ et nous concluons comme précédemment. □

Dans ce cas, le problème de savoir si deux schémas s_1 et s_2 sont tels que $s_1 \rhd' s_2$ est donc décidable : si m_{s_1} contient n, nous regardons d'abord si les motifs sont égaux à un décalage près. Si c'est le cas nous obtenons un k tel que $m_{s_1} \rightrightarrows_n^k m_{s_2}$ et nous pouvons vérifier, avec n'importe quelle procédure de décision pour l'arithmétique linéaire, que $c_{s_1} \Leftrightarrow c_{s_2}[n - k/n]$. Si m_{s_1} ne contient pas n, nous regardons si $m_{s_1} = m_{s_2}$ et nous vérifions que nous avons bien $\exists k > 0 (c_{s_1} \Leftrightarrow c_{s_2}[n - k/n])$, une fois encore avec une procédure de décision pour l'arithmétique linéaire. Par conséquent, dès que nous utiliserons l'égalité à un décalage près avec l'extension de la contrainte équivalente, *nous pourrons travailler à équivalence près sur les contraintes.*

Nous donnons finalement une extension plus faible mais dont la décidabilité est assurée, quel que soit le raffinement.

Definition 5.36. *Soit un schéma s, nous appelons* forme normale sans redondance *de s toute forme normale de s par le système de réécriture suivant :*

$$\begin{array}{ll} m \,\&\, c_1 \wedge \ldots \wedge c_k \rightarrow m \,\&\, c_1 \wedge \ldots \wedge c_{k-1} & \textit{si } \{c_1, \ldots, c_{k-1}\} \models c_k \\ m \,\&\, c \rightarrow m \,\&\, \bot & \textit{si } c \textit{ est insatisfaisable} \end{array}$$

Une telle forme normale sans redondance est calculable, par décidabilité de l'arithmétique linéaire. La réécriture termine car le nombre de contraintes diminue strictement. Notons qu'il n'y a pas de forme normale unique, mais le nombre total de formes normales différentes est fini, ce qui assure la décidabilité de l'extension suivante :

Definition 5.37 (Extension de la contrainte redondante)**.** *Soit $\triangleright$ un raffinement de bouclage par rapport à l'ordre standard. Nous appelons* extension de la contrainte redondante *de $\triangleright$, la relation $\triangleright'$ définie par : $s_1 \triangleright' s_2$ ssi il existe des formes normales sans redondance s'_1 et s'_2 de s_1 et s_2 telles que $s'_1 \triangleright s'_2$.*

Proposition 5.38. *L'extension de la contrainte redondante d'un raffinement de bouclage par rapport à l'ordre standard est un raffinement de bouclage par rapport à l'ordre standard.*

Preuve. Il est facile de montrer que s a un modèle $\mathfrak{M}$ ssi toute forme normale sans redondance de s a un modèle $\mathfrak{N}$ t.q. $\mathfrak{N}_{\text{arith}}(n) = \mathfrak{M}_{\text{arith}}(n)$ pour tout paramètre n de s. Ensuite nous concluons exactement de la même façon que dans la preuve de la proposition 5.30. □

5.3.3 Generalisation

Nous pouvons facilement généraliser les extensions précédentes :

Proposition 5.39. *Soit $\triangleright$ un raffinement de bouclage par rapport à l'ordre standard. Soit $\star$ une relation binaire t.q. pour tous schémas s_1 et s_2 tels que $s_1 \star s_2$, s_1 a un modèle $\mathfrak{M}$ équivaut à s_2 a un modèle $\mathfrak{N}$ et $\mathfrak{M}_{\text{arith}}(n) = \mathfrak{N}_{\text{arith}}(n)$ pour tout paramètre n de s.*

Alors la relation $\triangleright'$ définie par $s_1 \triangleright' s_2$ ssi il existe s'_1 et s'_2 t.q. $s_1 \star s'_1$, $s_2 \star s'_2$ et $s_1 \triangleright s_2$, est un raffinement de bouclage par rapport à l'ordre standard.

Nous pouvons ainsi appeler la relation $\triangleright'$ la "$\star$-extension" de $\triangleright$. Bien sûr, cette relation n'a un intérêt que si $\triangleright'$ permet de détecter plus de cycles que $\triangleright$. Nous pouvons considérer que $\star$ permet de travailler avec des formes normales qui sont mieux adaptées à la détection de cycle que leurs formes originales. En observant le théorème précédent et, en particulier, le fait que $\star$ doive être "compatible" avec la satisfaisabilité, nous pouvons voir que ces extensions seraient considérées, dans un autre contexte, comme des règles de simplifications ou des raffinements de la procédure de preuve destinés à améliorer son efficacité (réduire le nombre de pas de preuve ou la taille des formules). Dans le contexte des schémas, ces extensions sont importantes non seulement pour rendre la procédure plus efficace, mais également pour la *terminaison* (voir chapitres 6 et 9).

Notons finalement que toutes ces extensions et la notion même de bouclage pourraient être définies directement en tant que règles dans une procédure de preuve. Procéder de la façon dont nous l'avons fait ici à l'avantage de conserver une description haut niveau des procédures. Enfin, nous pourrions peut-être définir un seul et même système qui serait capable à la fois de traiter les schémas et la détection de cycle. Un tel système serait en revanche beaucoup trop général pour être automatisé.

En fait ceci nous rapprocherait probablement des preuves cycliques (voir chapitre 10) où la relation de bouclage est simplement l'égalité entre les étiquettes de deux nœuds. En revanche, le système associé doit être suffisamment puissant pour ramener un nœud sous une forme qui assure l'égalité avec un nœud antérieur (par analogie avec les schémas, c'est comme si STAB lui-même était capable de prouver qu'un schéma avec littéral non pur est équivalent au même schéma sans littéral pur). De plus il faut un critère sur la forme de la preuve pour assurer qu'un cycle est bien fondé. Pour résumer les preuves cycliques sont très proches de ce que nous présentons ici, et probablement plus générales, mais elles sont très certainement moins adaptées à l'automatisation.

5.4 Back to STAB

Nous revenons sur l'exemple présenté en 4.4.2 mais en utilisant la détection de cycle : les tableaux générés par STAB sont particulièrement adaptés à la détection de cycles, car les règles de décomposition

des itérations visent précisément à relier l'instance n du schéma à l'instance $n-1$. Le tableau obtenu est donné en figure 5.1.

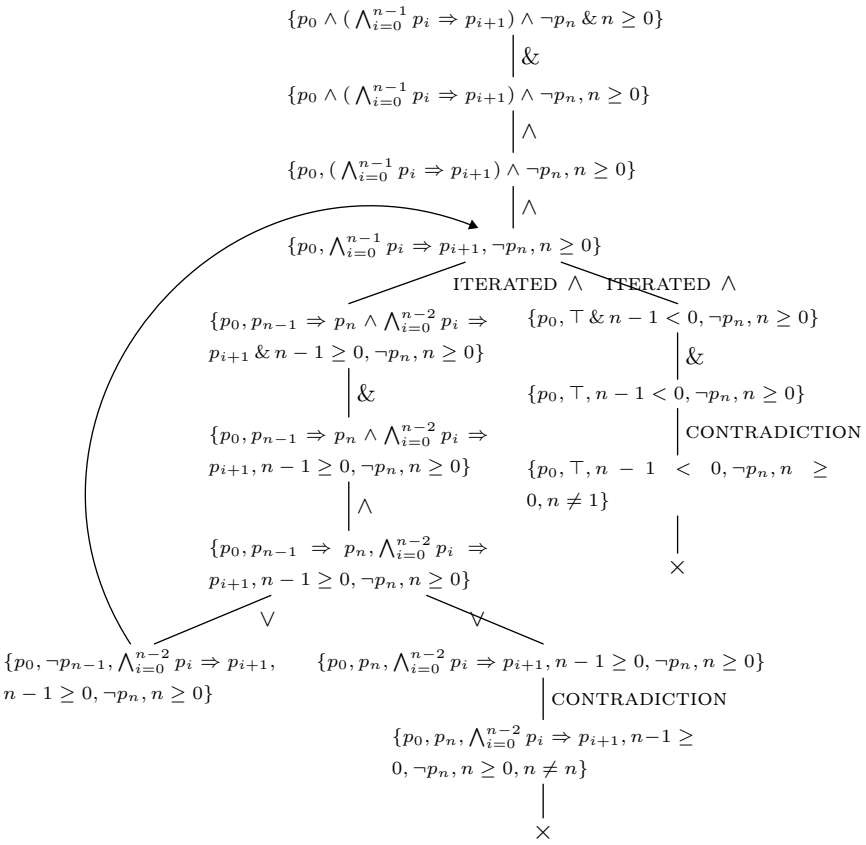


Figure 5.1: Application de la détection de cycle à l'exemple de 4.4.2.

Cette fois la flèche représente la détection *formelle* d'un cycle :

- nous commençons par exprimer les schémas correspondant aux deux étiquettes :

$$s_1 \stackrel{\text{def}}{=} p_0 \wedge \neg p_{n-1} \wedge \bigwedge_{i=0}^{n-2} p_i \Rightarrow p_{i+1} \wedge \neg p_n \,\&\, n-1 \geq 0 \wedge n \geq 0$$

  pour le bourgeon (l'origine de la flèche), et :

$$s_2 \stackrel{\text{def}}{=} p_0 \wedge \bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1} \wedge \neg p_n \,\&\, n \geq 0$$

  pour le compagnon (la destination de la flèche).

- nous voyons que nous n'avons pas égalité à un décalage près, en revanche nous pouvons appliquer les extensions. Nous voyons facilement que $\neg p_n$ est pur dans s_1, nous pouvons donc éliminer ce littéral pour obtenir :

$$s_1' \stackrel{\text{def}}{=} p_0 \wedge \neg p_{n-1} \wedge \bigwedge_{i=0}^{n-2} p_i \Rightarrow p_{i+1} \,\&\, n-1 \geq 0 \wedge n \geq 0$$

- puis nous pouvons appliquer l'extension de la contrainte redondante (ou équivalente), c.-à-d. éliminer les contraintes redondantes :

$$s_1'' \stackrel{\text{def}}{=} p_0 \wedge \neg p_{n-1} \wedge \bigwedge_{i=0}^{n-2} p_i \Rightarrow p_{i+1} \,\&\, n-1 \geq 0$$

Nous constatons alors que $s_1'' \rightrightarrows_n s_2$, modulo commutativité et associativité, d'où le cycle.

Nous constatons donc que cet exemple qui ne terminait pas auparavant peut maintenant être décidé grâce à la détection de cycle. Qui plus est tous les outils utilisés pour ce faire sont des outils dont nous avons démontré la calculabilité. Pour l'instant il est difficile de caractériser précisément les schémas que nous allons être capable de démontrer grâce à ces outils. Le chapitre suivant étudie une classe de schémas pour laquelle nous prouvons la complétude (pour la réfutation) et donc la décidabilité du problème de satisfaction. Ceci nous permettra de mieux appréhender la puissance des outils développés dans ce chapitre.

CHAPTER 6

Regular Schemata

Nous présentons dans ce chapitre une première classe décidable de schémas : les *schémas réguliers*. Les principales caractéristiques de ces schémas sont qu'ils sont monadiques et que leurs itérations ne peuvent pas être imbriquées.

En 6.1 nous définissons les schémas réguliers. Nous prouvons la complétude de cette classe en montrant que STAB termine sur ces schémas 6.2. Puis nous étudions la complexité du problème de la satisfaisabilité pour (une sous-classe de) les schémas réguliers 6.4. Pour faciliter cette étude, nous définissons une procédure à base d'automates, appelée SCHAUT, qui n'est en fait qu'une spécialisation de STAB aux schémas réguliers 6.3.

La complétude des schémas réguliers a été démontrée dans [ACP09b]. La définition de SCHAUT et la preuve de sa complexité est donnée dans [ACP10c]. Les résultats présentés ici sont plus généraux que ceux de [ACP10c], car le coefficient de n dans les indices apparaissant à l'extérieur des itérations peut être quelconques (et pas seulement de la forme $n + q$).

6.1 Definition

Definition 6.1. *Un schéma est :*

- monadique *ssi toutes ses propositions indexées sont monadiques*[1] *;*
- aligné sur $[e; f]$ *ssi toutes ses itérations ont le même domaine* $e \leq i \wedge i \leq f$ *(ou une formule équivalente à* $e \leq i \wedge i \leq f$*) à la variable* i *près, et où* e *et* f *ne contiennent pas* i*;*
- plat *ssi il ne contient pas d'itérations imbriquées, c.-à-d. : toutes ses itérations sont de la forme* $\Pi_{i|\delta}\, m$ *où* m *ne contient pas d'itération;*
- à propagation bornée *ssi toute proposition indexée apparaissant dans une itération* $\Pi_{i|\delta}\, m$ *est de la forme* p_{i+q} *où* $q \in \mathbb{Z}$*.*

Notons que, dans un schéma à propagation bornée, seules les propositions indexées apparaissant dans une itération *sont contraints : les littéraux apparaissant en dehors des itérations sont quelconques, nous pouvons avoir p.ex.* p_{2n} *ou* p_{-n+1}*.*

Definition 6.2 (Schémas réguliers)**.** *Un schéma est* régulier *ssi il n'a qu'un paramètre* n*, est monadique, plat, à propagation bornée et aligné sur* $[q_1; n - q_2]$ *où* $q_1, q_2 \in \mathbb{Z}$ *.*

[1] c.-à-d. qu'elles n'ont qu'un indice, voir définition page 18.

De plus rappelons que, comme précisé en p.43, nous supposons qu'il existe un $k \in \mathbb{N}$ t.q. la contrainte du schéma entraîne $n \geq k$[2].

Les schémas réguliers sont très contraints par rapport aux schémas en général, ceci pour faciliter la preuve de complétude. Nous donnons donc quelques exemples pour mieux cerner ce que nous pouvons exprimer via les schémas réguliers. Notons que plusieurs contraintes peuvent facilement être assouplies, p.ex. le fait que toutes les itérations aient le même domaine : les itérations de $s \stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} p_i \wedge \bigvee_{i=1}^{n+1} \neg p_i \,\&\, n \geq 0$, n'ont pas le même domaine mais s est équivalent à $\bigwedge_{i=1}^{n} p_i \wedge (\bigvee_{i=1}^{n} \neg p_i \vee \neg p_{n+1}) \,\&\, n \geq 0$ dont toutes les itérations ont bien le même domaine. Ce type de transformations sera esquissé à la section 9.A.

Example 6.3. *L'exemple récurrent* $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$ *est un schéma régulier :*

- *il n'a qu'un paramètre* n*;*
- *il est monadique car* p *n'a qu'un indice;*
- *il est aligné car il n'a qu'une itération, elle a bien la forme voulue avec* $q_1 = 1$ *et* $q_2 = 1$*;*
- *il est plat car il ne contient qu'une itération;*
- *il est à propagation bornée car* i*,* $i + 1$ *ont bien la forme voulue.*

Example 6.4. *Nous pouvons spécifier avec les schémas réguliers un circuit additionneur à propagation de retenue. Nous représentons un vecteur de* n *bits* $v = V_1, \ldots, V_n$ *par une proposition indexée monadique* v_i*. Supposons que le circuit calcule la somme* s *de deux vecteurs* x *et* y*, nous obtenons alors le schéma suivant :*

$$\text{adder}(x, y, c, s) \stackrel{def}{=} (\bigwedge_{i=1}^{n} \text{sum}_i) \wedge (\bigwedge_{i=1}^{n} \text{carry}_i) \wedge \neg \text{carry}_1$$

où :

- $\text{sum}_i \stackrel{def}{=} s_i \Leftrightarrow (x_i \oplus y_i) \oplus c_i$
- $\text{carry}_i \stackrel{def}{=} c_{i+1} \Leftrightarrow (x_i \wedge y_i) \vee (y_i \wedge c_i) \vee (x_i \wedge c_i)$
- $c_1, \ldots, c_n$ *représente la retenue du circuit propagée de bit en bit.*

Il est facile de voir que ce schéma est régulier :

- *il n'a qu'un paramètre* n*;*
- *il est monadique car* x*,* y*,* c*,* s *n'ont qu'un indice;*
- *il est aligné car les deux itérations ont le même domaine; par ailleurs elles ont bien la forme voulue avec* $q_1 = 1$ *et* $q_2 = 0$*;*
- *il est plat car les deux seules itérations apparaissent à des positions parallèles;*
- *il est à propagation bornée car les seules expressions entières de* sum_i *et* carry_i *sont* i *et* $i + 1$*.*

Nous pouvons ensuite exprimer les propriétés suivantes de ce circuit :

- 0 *est élément neutre :*

$$\text{adder}(x, y, c, s) \wedge \left(\bigwedge_{i=1}^{n} \neg y_i\right) \wedge \left(\bigvee_{i=1}^{n} x_i \oplus s_i\right) \,\&\, n \geq 0$$

Ce schéma est insatisfaisable ssi 0 *est élément neutre. En effet :*

[2] En fait il est possible de montrer que tout schéma régulier de contrainte quelconque est équivalent (pour la satisfaisabilité) à un schéma régulier dont la contrainte entraîne $n \geq k$ pour un certain $k \in \mathbb{Z}$: il est facile de prouver qu'il existe un $n_0 \in \mathbb{Z}$ tel que toutes les instances de rang inférieur à n_0 sont équivalentes pour la satisfaisabilité (en fait elles sont toutes équivalentes à une même formule propositionnelle à un décalage près de certains indices). En particulier, nous devons au moins avoir $n_0 \leq q_1 - 1$, c.-à-d. que n_0 soit suffisamment petit pour que toutes les itérations soient vides. Il suffit ensuite d'ajouter la contrainte $n \geq n_0$ au schéma d'origine.

- $\bigwedge_{i=1}^{n} \neg y_i$ *exprime le fait que le deuxième opérande vaut zéro (c.-à-d. que tous ses bits sont nuls);*
- $\bigvee_{i=1}^{n} x_i \oplus s_i$ *exprime le fait que la sortie est différente de la première opérande (c.-à-d. qu'il existe un bit de x qui est différent du bit correspondant de y).*

- *l'addition est commutative :*

$$\text{adder}(x, y, c, s) \wedge \text{adder}(y, x, c', t) \wedge \left(\bigvee_{i=1}^{n} s_i \oplus t_i \right) \,\&\, n \geq 0$$

Ce schéma est insatisfaisable ssi l'additionneur est commutatif. En effet, s est le résultat de l'addition de x et y, et t est le résultat de l'addition de y et x. Dès lors, $\bigvee_{i=1}^{n} s_i \oplus t_i$ exprime le fait que $s \neq t$, ce qui est impossible si l'additionneur est commutatif.

- *l'addition est associative :*

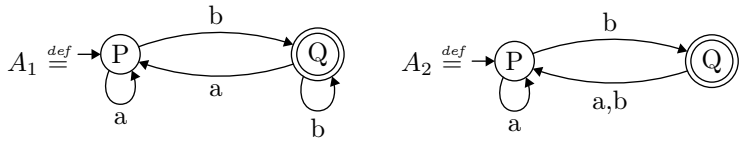
$$\begin{array}{c} \text{adder}(x, y, c, s') \wedge \text{adder}(s', z, c', s) \wedge \text{adder}(y, z, c'', t') \\ \wedge \text{adder}(x, t', c''', t) \wedge (\bigvee_{i=1}^{n} s_i \oplus t_i) \,\&\, n \geq 0 \end{array}$$

Par un raisonnement similaire aux propriétés précédentes, ce schéma est insatisfaisable ssi l'additionneur est associatif.

Tous ces schémas sont réguliers, et insatisfaisables (bien que nous ne le prouvons pas ici).

Example 6.5. *Nous pouvons aussi exprimer un additionneur à retenue anticipée et l'équivalence de cet additionneur avec l'addition à propagation de retenue. Nous pouvons aussi spécifier un circuit effectuant la comparaison sur les entiers. Ainsi toute formule de l'arithmétique de Presburger (sans quantificateurs) sur les vecteurs de bits peut être exprimée avec les schémas réguliers.*

Example 6.6. *Soient les deux automates suivants :*



Il est facile de voir que tous les mots reconnus par A_2 le sont par A_1, c.-à-d. le langage de A_2 est inclus dans celui de A_1. Nous pouvons exprimer ce problème avec des schémas de formules : le paramètre désigne la longueur d'une dérivation; si nous sommes à l'état P à la lecture du i^{th} caractère alors nous avons la proposition P_i; si le i^{th} caractère est a alors nous avons a_i.

Ainsi A_1 est spécifié par :

$$A_1(P, Q) \stackrel{def}{=} P_1 \wedge \left(\bigwedge_{i=1}^{n} \begin{array}{c} P_i \Rightarrow (a_i \Rightarrow P_{i+1} \wedge b_i \Rightarrow Q_{i+1}) \\ \wedge Q_i \Rightarrow (a_i \Rightarrow P_{i+1} \wedge b_i \Rightarrow Q_{i+1}) \end{array} \right)$$

et A_2 par :

$$A_2(P, Q) \stackrel{def}{=} P_1 \wedge \left(\bigwedge_{i=1}^{n} \begin{array}{c} P_i \Rightarrow (a_i \Rightarrow P_{i+1} \wedge b_i \Rightarrow Q_{i+1}) \\ \wedge Q_i \Rightarrow (a_i \Rightarrow P_{i+1} \wedge b_i \Rightarrow P_{i+1}) \end{array} \right)$$

Il faut aussi spécifier l'unicité des différents états et transitions :

$$\text{uni}(P, Q) \stackrel{def}{=} (\bigwedge_{i=1}^{n} P_i \Leftrightarrow \neg Q_i) \wedge (\bigwedge_{i=1}^{n} a_i \Leftrightarrow \neg b_i)$$

Et nous concluons en supposant que nous sommes dans l'état final de A_2 au rang n mais que nous ne sommes pas dans l'état final de A_1, ce qui est évidemment impossible si le langage de A_2 est inclus dans celui de A_1 :

$$A_1(P^1, Q^1) \wedge A_2(P^2, Q^2) \wedge \text{uni}(P^1, Q^1) \wedge \text{uni}(P^2, Q^2) \wedge Q_n^2 \wedge \neg Q_n^1 \,\&\, n \geq 0$$

Tous les schémas utilisés sont réguliers. Notons que cette traduction est facile à automatiser. Par ailleurs elle semble généralisable aux automates alternants, et le résultat est toujours un schéma régulier.

Note. Cette possibilité d'exprimer des automates par des schémas réguliers, et la procédure SchAut, basée sur des automates, présentée en section 6.3, jettent les bases pour une étude approfondie des relations entre langages réguliers et schémas réguliers[3]. Nous pourrions typiquement introduire une notion de *définissabilité* pour les schémas réguliers, dans le but d'obtenir un résultat similaire à celui de Büchi pour la logique monadique du second ordre faible [Bü59].

Example 6.7. *En revanche le schéma suivant, formalisant le principe des pigeonniers,* n'est pas régulier *:*

$$\left(\bigwedge_{i=1}^{n+1}\bigvee_{j=1}^{n} p_{i,j}\right) \wedge \left(\bigwedge_{i_1=1}^{n+1}\bigwedge_{i_2 \neq i_1 \,\wedge 1\leq i_2\wedge i_2\leq n+1}\bigwedge_{j=1}^{n}(p_{i_1,j} \Rightarrow \neg p_{i_2,j})\right)$$

où n représente le nombre de pigeonniers et $p_{i,j}$ signifie "le i^{th} pigeon va dans le j^{th} pigeonnier". Le membre gauche de la conjonction signifie "chaque pigeon a un pigeonnier", et le membre droit signifie "chaque pigeonnier ne contient qu'un pigeon".

Ce schéma n'est pas régulier car il contient des propositions non monadiques. De plus il n'est ni plat, ni aligné.

Example 6.8. *De même, l'exemple suivant n'est pas un schéma régulier :*

$$s \stackrel{def}{=} \neg p_1 \wedge \left(\bigwedge_{i=1}^{n} p_i \Rightarrow p_{2i}\right) \wedge \left(\bigwedge_{i=n+1}^{2n-1} \neg p_i\right)$$

En effet il n'est pas aligné et, plus important, il n'est pas à propagation bornée car il contient le littéral p_{2i}.

Nous avons : $\langle s\rangle_{\{n\mapsto k\}}$ est satisfaisable ssi k est une puissance de 2. En pratique il est utile pour spécifier des circuits dont la conception n'est correcte que si le nombre de bits est une puissance de deux. Il s'agit typiquement de circuits du type "diviser pour régner", c.-à-d. des circuits composés de deux sous-circuits traitant chacun la moitié des bits de l'entrée. Chacun de ces sous-circuits est lui-même composé de deux sous-circuits, etc. Nous pouvons ainsi concevoir, p.ex., des circuits calculant le bit de parité ou des circuits additionneurs optimisés [Gup94].

6.2 Termination

Nous prouvons la complétude pour la réfutation des schémas réguliers en démontrant que Stab termine lorsqu'il prend un schéma régulier en entrée. En effet, lorsque Stab termine, soit le dernier tableau contient une branche (irréductible), soit toute branche du tableau est close. Dans le premier cas, nous savons par correction (théorème 4.5) que le schéma est satisfaisable. Dans le deuxième cas, nous savons par complétude pour la satisfaisabilité (théorème 4.9) que le schéma est insatisfaisable.

Pour assurer la terminaison de Stab nous définissons la stratégie τ suivante :

- les règles d'itérations sont appliquées en dernier ressort, lorsque plus aucune autre règle ne peut s'appliquer;
- les règles d'itérations sont appliquées uniquement sur les itérations de longueur maximale par rapport à l'ordre partiel $\preceq$ sur les expressions entières [4], défini p.11;
- nous utilisons pour la détection de cycle l'égalité à un décalage près avec les extensions du littéral pur et de la contrainte équivalente[5].

L'intuition derrière τ est la suivante :

- nous effectuons tous les calculs *propositionnels* possibles;

[3] Notons que l'emploi du terme "régulier" dans "schéma régulier" est complètement fortuit.

[4] Pour rappel, la longueur d'une itération de la forme $\Pi_{i=e}^{f} m$ est l'expression $f - e + 1$; p.ex. si un nœud est étiqueté par $\{\bigwedge_{i=1}^{n}\ldots, \bigwedge_{i=1}^{n-1}\ldots\}$, alors $\bigwedge_{i=1}^{n}\ldots$ a la plus grande longueur.

[5] Il est clair que l'ordre dans lequel on compose ces extensions n'a pas d'importance.

- puis nous déroulons une itération de longueur maximale;
- nous recommençons les calculs propositionnels sur les schémas introduits par le déroulage, puis nous déroulons une *autre* itération (car l'itération précédente n'est plus de longueur maximale);
- etc. jusqu'à ce que toutes les itérations aient été déroulées (toutes les itérations ont alors la *même longueur*);
- nous recommençons avec la première itération.

Il est facile de prouver que τ préserve la complétude pour la satisfaisabilité : évidemment, la mesure présentée p.35 diminue toujours; il faut juste vérifier que les restrictions introduites par τ ne peuvent pas mener à une situation où un nœud est irréductible et insatisfaisable sans que la branche soit fermée, c.-à-d. il faut vérifier que le lemme 4.4 est préservé. Il est facile de voir que la preuve est toujours valable.

Definition 6.9. *Nous appelons* nœud d'alignement *un nœud tel que seules les règles d'itération peuvent s'appliquer et toutes les itérations ont la même longueur.*

Theorem 6.10. *τ termine sur tout schéma régulier.*

Dans la preuve et dans toute la suite du chapitre, nous utiliserons les notations suivantes (s est un schéma régulier) :

$$\mathrm{E}_q(s) \stackrel{\text{def}}{=} \{q \in \mathbb{Z} \mid p_q \text{ apparaît dans } s \text{ pour un quelconque } p\}$$
$$\mathrm{E}_{q\mathrm{n}}(s) \stackrel{\text{def}}{=} \{q_1 \in \mathbb{Z} \mid p_{q_1 n+q_2} \text{ apparaît dans } s \text{ pour un quelconque } p\}$$
$$\mathrm{E}_{\mathrm{n}+q}(s) \stackrel{\text{def}}{=} \{q_2 \in \mathbb{Z} \mid p_{q_1 n+q_2} \text{ apparaît dans } s \text{ pour un quelconque } p\}$$
$$\mathrm{E}_{\mathrm{i}+q}(s) \stackrel{\text{def}}{=} \{q \in \mathbb{Z} \mid p_{i+q} \text{ apparaît dans } s \text{ pour un quelconque } p\}$$

Les preuves de certains cas seront "survolées" dans la démonstration suivante : elles seront détaillées dans la preuve de complexité (section 6.4).

Preuve. Soit un schéma régulier s dont toutes les itérations ont pour domaine $q_1 \leq i \wedge i \leq n - q_2$, ($q_1, q_2 \in \mathbb{Z}$ fixés). Supposons qu'il existe une branche infinie. Il y a donc une infinité de nœuds dans cette branche et il est facile de voir qu'il y aussi une infinité de nœuds d'alignement d'après la proposition 2.20. Il y a donc un ensemble infini de schémas associés aux nœuds d'alignement. Nous montrons que cet ensemble ne peut qu'être fini à un décalage près (avec les extensions du littéral pur et de la contrainte équivalente). Par conséquent la règle de bouclage s'applique nécessairement, au pire quand toutes les formules possibles auront été générées. Nous notons s_k le schéma du k^{th} nœud d'alignement. Nous montrons que l'ensemble $\{s_k \mid k \in \mathbb{N}\}$, noté par la suite $(s_k)_{k\in\mathbb{N}}$, est fini à un décalage près. Comme nous ne considérons que les nœuds d'alignement, chaque nœud est étiqueté uniquement par des littéraux, des itérations ou des contraintes (puisque les règles d'extension propositionnelles s'appliquent sur toute conjonction ou disjonction). Donc nous pouvons exprimer s_k sous la forme $L_k \wedge IT_k \,\&\, c_k$ où L_k et IT_k sont des conjonctions de littéraux et d'itérations (respectivement) et c_k est la contrainte de s_k. Nous montrons que $(L_k)_{k\in\mathbb{N}}$, $(IT_k)_{k\in\mathbb{N}}$ et $(c_k)_{k\in\mathbb{N}}$ sont tous finis à un décalage près, puis nous obtenons le résultat pour $(s_k)_{k\in\mathbb{N}}$ en appliquant le corollaire 5.25. Nous décomposons donc la preuve en trois lemmes : un pour $(IT_k)_{k\in\mathbb{N}}$, un pour $(L_k)_{k\in\mathbb{N}}$ et un pour $(c_k)_{k\in\mathbb{N}}$.

Lemma 6.11. *$(IT_k)_{k\in\mathbb{N}}$ est finie à un décalage près. De plus pour tout $k \in \mathbb{N}$: IT_k est à translation bornée et la seule expression contenant n dans IT_k est $n - q_2 - k$ (ces propriétés sont nécessaires à l'application du corollaire 5.25).*

Preuve. Tout élément de IT_k a la forme $\Pi_{i=q_1}^{n-q_2-k} m$ tel que $\Pi_{i=q_1}^{n-q_2} m$ apparaît dans la formule d'origine, en effet k est le nombre de nœuds d'alignement précédant le nœud courant (incluant ce dernier), c.-à-d. le nombre d'application des règles d'itération sur chacune des itérations. Il existe donc une quantité finie de telles itérations. Qui plus est, comme m ne peut pas contenir n (par définition des schémas réguliers), il est clair que la suite $(\Pi_{i=q_1}^{n-q_2-k} m)_{k\in\mathbb{N}}$ est finie à un décalage près. C'est donc aussi le cas de $(IT_k)_{k\in\mathbb{N}}$. Par définition des schémas réguliers, la seule expression contenant n est $n - q_2 - k$, $(IT_k)_{k\in\mathbb{N}}$ est donc bien à translation bornée. □

Lemma 6.12. *$(c_k)_{k\in\mathbb{N}}$ est fini à un décalage près avec l'extension de la contrainte équivalente. De plus il existe k_0 tel que pour tout $k \geq k_0$, c_k est à translation bornée et toute expression contenant n a la forme $n - k + q$, où $q \in \mathbb{Z}$ appartient à un intervalle borné indépendamment de k.*

Preuve. Pour tout $k \in \mathbb{N}$, supposons que toutes les contraintes redondantes sont éliminées dans c_k. Nous séparons la suite $(c_k)_{k\in\mathbb{N}}$ en trois :

- les contraintes introduites par les règles d'itération (notées $(c_k^{\text{it}})_{k\in\mathbb{N}}$);
- la contrainte du schéma initial;
- les contraintes introduites par la règle de clôture.

Les contraintes qui ont été introduites par les règles d'itération au rang k sont de la forme $q_1 \leq n-q_2-k'$ (branche gauche des règles d'itérations) ou $q_1 > n-q_2-k'$ (branche droite), avec $k' \leq k$. Dans le deuxième cas les itérations sont toutes supprimées donc il ne reste que des formules propositionnelles et il est trivial que Stab termine dans ces cas là, impossible puisque nous sommes dans une branche infinie. Ainsi les contraintes introduites sont $q_1 \leq n - q_2$, $q_1 \leq n - q_2 - 1$, $q_1 \leq n - q_2 - 2$, etc. Donc $c_k^{\text{it}} = \bigwedge_{k'\in[0;k]} q_1 \leq n - q_2 - k'$, qui grandit indéfiniment avec k. Mais il est clair que, pour tout $k' < k$: $q_1 \leq n - q_2 - k \models q_1 \leq n - q_2 - k'$. Autrement dit, toutes les contraintes $q_1 \leq n - q_2 - k'$ pour $k' < k$ sont redondantes par rapport à $q_1 \leq n - q_2 - k$. Donc, comme nous utilisons l'extension de la contrainte équivalente, c_k^{it} se réduit à $q_1 \leq n - q_2 - k$ et $(c_k^{\text{it}})_{k\in\mathbb{N}}$ est fini à un décalage près. De plus $-q_2$ est indépendant de k, nous avons donc bien la forme voulue dans l'énoncé.

Nous considérons maintenant la contrainte du schéma initial c_s, mais pas seule : nous lui associons c_k^{it}. Ainsi la suite que nous étudions est $(c_k^{\text{it}} \wedge c_s)_{k\in\mathbb{N}}$, c.-à-d. $(q_1 \leq n - q_2 - k \wedge c_s)_{k\in\mathbb{N}}$ qui n'est, a priori, pas finie à un décalage près : c_s contient n (sauf cas triviaux) mais ne dépend pas de k. Nous montrons que, pour k suffisamment grand, c_s est redondante par rapport à $q_1 \leq n - q_2 - k$. Comme s est régulier il n'a qu'un paramètre n, donc sa contrainte n'a qu'une variable libre n. Par élimination des quantificateurs et mise sous forme normale disjonctive, cette contrainte est une disjonction de formules arithmétiques de la forme $q|n \wedge q \leq n$ ou $q|n \wedge n \leq q'$ ou $q|n \wedge q' \leq n \wedge n \leq q''$ où $q', q'' \in \mathbb{Z}$, $q \in \mathbb{N}$. Alors, toute formule de la forme $q \leq n$ est conséquence de $n \geq q_1 + q_2 + k$ si k est suffisamment grand et toute formule de la forme $n \leq q$ est en contradiction avec $n \geq q_1 + q_2 + k$ si k est suffisamment grand. Donc, si k est suffisamment grand, $c_k^{\text{it}} \wedge c_s$ est conséquence de $c_k^{\text{it}} \wedge q_1'|n \vee q_2'|n \vee \ldots$, avec $q_1', q_2', \ldots \in \mathbb{N}$. Or, pour tout $q \in \mathbb{N}$, $q|n$ est équivalent à $q|n - q$, et donc, par récurrence, à $q|n - \lceil \frac{k}{q} \rceil q$. Nous pouvons donc écrire cette formule sous la forme $q|n - k + (k - \lceil \frac{k}{q} \rceil q)$, nous avons alors : $k - \lceil \frac{k}{q} \rceil q \leq q$. De plus q est inférieure au maximum des q' tels que $q'|n$ apparaît dans la forme normale disjonctive sans quantificateurs de c_s, donc indépendante de k. Ainsi $c_k^{\text{it}} \wedge q|n - \lceil \frac{k}{q} \rceil q$ est fini à un décalage près, d'après le corollaire 5.25. Au final, $(c_k^{\text{it}} \wedge c_s)_{k\in\mathbb{N}}$ est bien finie à un décalage près avec l'extension de la contrainte équivalente, et toute expression contenant n a bien la forme $n - k + q$ où q appartient à un intervalle borné indépendamment de k.

Enfin nous considérons les contraintes introduites par la règle de clôture. La suite de contraintes ainsi introduites a la forme $(\bigwedge_i e_i \neq f_i)_{k\in\mathbb{N}}$. Nous considérons en fait la suite $(c_k^{\text{it}} \wedge \bigwedge_i e_i \neq f_i)_{k\in\mathbb{N}}$, l'idée étant une fois encore que $c_k^{\text{it}} = q_1 \leq n - q_2 - k$ va rendre redondante les contraintes empêchant la finitude à un décalage près. Chaque contrainte introduite par clôture a la forme $e \neq f$.

- Si ni e, ni f ne contiennent n alors la contrainte ne pose pas de problème pour l'égalité à un décalage près.
- Si tous deux contiennent n alors la contrainte a la forme $qn + r \neq q'n + r'$, équivalent à $(q - q')n + r - r' \neq 0$. Si $q = q'$ alors cette contrainte est trivialement valide ou insatisfaisable. Si $q \neq q'$ alors, comme nous avons $q_1 \leq n - q_2 - k$, c.-à-d. $n \geq q_1 + q_2 + k$, cette contrainte est redondante si k est suffisamment grand.
- Si e contient n, $f \in \mathbb{Z}$ et e appartient au schéma d'origine; alors f apparaît nécessairement dans la formule initiale car Stab n'introduit pas d'expressions arithmétiques qui soient de simples entiers. Donc la contrainte sera redondante par rapport à $q_1 \leq n - q_2 - k$ si k est suffisamment grand.
- Si e contient n, $f \in \mathbb{Z}$ et e provient du dépliage d'une itération; une fois encore f apparaît nécessairement dans la formule initiale Soit $k' \leq k$ tel que e provient du k'^{th} dépliage d'une itération. Comme les indices sont de la forme $i + q$ pour $q \in \mathrm{E}_{\mathrm{i}+q}(s)$, e a la forme $n - q_2 - k' + q$. Dans ce

cas il est facile de voir que si $k' \leq q_1 + k - f + q - 1$ alors $e \neq f$ est redondante par rapport à $q_1 \leq n - q_2 - k$. Donc nous pouvons restreindre la forme de e à $n - k - q_2 + q'$ où $q' \leq q_1 + f$, et même, comme f provient du schéma d'origine, $f \leq \min(\mathrm{E}_q(s))$, donc $q' \leq q_1 + \min(\mathrm{E}_q(s))$. Par ailleurs, comme $q \in \mathrm{E}_{\mathrm{i}+q}(s)$, $q \geq \min(\mathrm{E}_{\mathrm{i}+q}(s))$ donc $q' \geq \min(\mathrm{E}_{\mathrm{i}+q}(s))$. Ainsi e a la forme $n-k-q_2+q'$ où $q' \in [\min(\mathrm{E}_{\mathrm{i}+q}(s)); q_1 + \min(\mathrm{E}_q(s))]$. L'ensemble de contraintes générées est donc un sous-ensemble de :

$$\left(\bigcup_{\substack{Q \subseteq [\min(\mathrm{E}_{\mathrm{i}+q}(s)); q_1 + \min(\mathrm{E}_q(s))] \\ E \subseteq \mathrm{E}_q(s)}} \bigwedge_{\substack{q' \in Q \\ f \in E}} n - k - q_2 - q' \neq f \right)_{k \in \mathbb{N}}$$

qui est fini à un décalage près (avec l'extension de la contrainte équivalente) car $[\min(\mathrm{E}_{\mathrm{i}+q}(s)); q_1 + \min(\mathrm{E}_q(s))]$ et $\mathrm{E}_q(s)$ ne dépendent pas de k. De plus Q est fini indépendante de k, et q_2 est indépendant de k, donc toute expression contenant n a bien la forme $n - k + q$ où q appartient à un intervalle borné indépendamment de k.

Nous pouvons donc conclure dans chacun des cas, ce qui permet de montrer le résultat voulu. Notons que toutes les expressions de la forme qn sont éliminées par redondance, donc les contraintes sont bien à translation bornée si k est suffisamment grand. □

Lemma 6.13. *$(L_k)_{k \in \mathbb{N}}$ est fini à un décalage près avec l'extension du littéral pur. De plus il existe k_0 tel que pour tout $k \geq k_0$, L_k est à translation bornée et toute expression contenant n a la forme $n - k + q$, où $q \in \mathbb{Z}$ appartient à un intervalle borné indépendamment de k.*

Preuve. Nous séparons L_k de la façon suivante :

- L_k^q est la conjonction des littéraux de L_k ayant un entier pour indice;
- L_k^{n} est la conjonction des littéraux de L_k dont l'indice contient n et qui étaient présents dans le schéma d'origine;
- $L_k^{[\mathrm{n/i}]}$ est la conjonction des autres littéraux de L_k, c.-à-d. ceux qui sont issus du déroulage d'une itération.

$(L_k^q)_{k \in \mathbb{N}}$ est trivialement fini à un décalage près : il ne contient pas n.

Nous montrons que tout littéral de L_k^{n} est pur à partir d'un certain rang. Soit donc un indice e d'un littéral de L_k^{n}. Pour montrer que le littéral est pur il faut montrer que pour tout autre littéral d'indice f, nous avons $\rho(e) \neq \rho(f)$ quel que soit l'environnement ρ. Tout d'abord, nous montrons qu'à partir d'un certain nombre de dépliages (c.-à-d. si k est suffisamment grand), aucun dépliage ultérieur ne peut introduire de littéral d'indice f pour lequel nous ayons $\rho(e) = \rho(f)$ (nous pourrons ensuite considérer que l'ensemble de littéraux dont l'indice peut être égal à e est figé). C'est le cas si pour tout environnement ρ nous avons $\rho(n - q_2 - k + \max(\mathrm{E}_{\mathrm{i}+q}(s))) < \rho(e)$. Cette inéquation est équivalente à $\rho(k) > \rho(n - q_2 + \max(\mathrm{E}_{\mathrm{i}+q}(s)) - e)$, et comme n est la seule variable dans ces expressions : $k > \rho(n) - q_2 + \max(\mathrm{E}_{\mathrm{i}+q}(s)) - e$. Comme L_k^{n} ne contient que des littéraux présents depuis le début de la procédure, e est indépendant de k. Donc l'inéquation est équivalente à $k > \rho(n) + q$ où q est indépendant de k. Il suffit donc de prendre $k > q$ pour que l'inéquation soit vérifiée. Ainsi, pour tout k supérieur à q, aucun dépliage ne peut introduire de littéral d'indice f pour lequel nous ayons $\rho(e) = \rho(f)$. Il suffit donc de montrer que pour tout littéral d'indice f apparaissant dans L_k, avec $k > q$, nous avons $\rho(e) \neq \rho(f)$ quel que soit l'environnement ρ. Supposons que e a la forme $qn+r$ et f a la forme $q'n+r'$. Nous obtenons le résultat ssi $e \neq f$, c.-à-d. $(q - q')n + r - r' \neq 0$. Nous ne pouvons pas avoir $q = q'$ et $r = r'$ car la règle de clôture aurait alors introduit une contrainte insatisfaisable et la branche serait finie, ce qui est contradictoire avec l'hypothèse qu'elle est infinie. Si $q = q'$ et $r \neq r'$ le résultat est trivial. Si $q \neq q'$, il suffit de prendre k suffisamment grand pour que l'inégalité soit vérifiée. Nous pouvons donc finalement considérer que $L_k^{\mathrm{n}} = \emptyset$.

Par ailleurs, une fois que ces littéraux sont purs, nous pouvons considérer que le schéma est à translation bornée (définition 5.18) : en effet seuls les littéraux de L_k^{n} peuvent avoir un indice contenant n et qui ne soit pas de la forme $n + q$, $q \in \mathbb{Z}$ (puisque, d'une part, aucun littéral de L_k^q ne contient n et que, d'autre part, tout littéral de $L_k^{[\mathrm{n/i}]}$ provient du corps d'une itération, et donc a bien la forme $n+q$, $q \in \mathbb{Z}$,

car tout schéma régulier est à propagation bornée). Donc, pour ce qui est de la détection de cycle, le schéma est bien à translation bornée quand ces littéraux sont purs.

Enfin soit un littéral de $L_k^{[\mathrm{n/i}]}$. Nous montrons que, *si ce littéral n'est pas pur*, alors son indice a la forme $n-k+q$ où $q \in \mathbb{Z}$ appartient à un intervalle borné indépendamment de k. Dans tous les cas, son indice a la forme $n-q_2-k'+q$ où $k' \leq k$ et $q \in \mathrm{E}_{\mathrm{i}+q}(s)$. Supposons que ce littéral ne soit pas pur. L_k contient donc un littéral de signe opposé et de même indice. D'après le paragraphe précédent, nous pouvons supposer que ce littéral n'appartient pas à L_k^{n}. Donc soit ce littéral est un autre littéral de $L_k^{[\mathrm{n/i}]}$, soit il apparaît implicitement dans une itération. Dans le premier cas, la règle de clôture pourrait s'appliquer, ce qui est contradictoire avec le fait que nous considérons uniquement des nœuds d'alignement. Donc la seule possibilité est que ce littéral apparaisse implicitement dans une itération. Nous avons donc $-k'+q \leq -k+\max(\mathrm{E}_{\mathrm{i}+q}(s))$. Par ailleurs nous avons nécessairement $-k'+q \geq -k'+\min(\mathrm{E}_{\mathrm{i}+q}(s))$, et donc $-k'+q \geq -k+\min(\mathrm{E}_{\mathrm{i}+q}(s))$. Donc l'indice de ce littéral est compris entre $n-q_2-k+\min(\mathrm{E}_{\mathrm{i}+q}(s))$ et $n-q_2-k-\max(\mathrm{E}_{\mathrm{i}+q}(s))$. Ainsi tout littéral non pur de $L_k^{[\mathrm{n/i}]}$ apparaît dans :

$$L'_k \stackrel{\text{def}}{=} \bigwedge_{\substack{p \text{ est une prop. indexée} \\ j \in [\min(\mathrm{E}_{\mathrm{i}+q}(s)); \max(\mathrm{E}_{\mathrm{i}+q}(s))]}} p_{n-q_2-k+j} \wedge \neg p_{n-q_2-k+j}$$

$(L'_k)_{k\in\mathbb{N}}$ est clairement fini à un décalage près. Ainsi $(L_k^{[\mathrm{n/i}]})_{k\in\mathbb{N}}$, et donc finalement $(L_k)_{k\in\mathbb{N}}$, est fini à un décalage près avec l'extension du littéral pur. De plus tout indice contenant n a bien la forme $n-k+q$ où $q \in \mathbb{Z}$ appartient à un intervalle borné indépendamment de k. □

Au final, toutes les suites sont finies à un décalage près. De plus elles sont toutes à translation bornée à partir d'un certain rang et toute expression contenant n est de la forme $n-k+q$ où $q \in \mathbb{Z}$ appartient à un intervalle borné indépendamment de k. Ainsi ces trois suites vérifient les conditions d'applications du corollaire 5.25. Nous pouvons donc appliquer ce dernier pour "combiner" toutes ces suites. Nous en concluons que l'ensemble des schémas apparaissant dans la branche infinie est fini à un décalage près. Par conséquent la règle de bouclage s'applique nécessairement, ce qui contredit le fait que la branche est infinie. □

6.3 SchAut

Comme les schémas réguliers ont une forme très particulière, nous pouvons, plutôt qu'utiliser Stab, définir une procédure de preuve dédiée aux schémas réguliers. Ceci est particulièrement utile pour pouvoir étudier simplement la complexité du problème de la satisfaisabilité comme nous le ferons en 6.4. Ainsi nous définissons SchAut (pour **Sch**ema **Aut**omaton) une procédure qui, à partir d'un schéma, construit un automate dont le langage est vide ssi le schéma est insatisfaisable. Un point intéressant des schémas réguliers est que toutes leurs itérations ont la même longueur. Ainsi SchAut va construire un automate qui prend en entrée un entier représentant cette longueur et l'accepte ssi l'instance correspondante du schéma a un modèle propositionnel. Cela permet en particulier de se débarrasser des formules arithmétiques ce qui simplifie grandement le système.

Cependant, cette simplicité a un prix : si un schéma régulier aligné sur $[q_1; n-q_2]$ est fourni en entrée de SchAut, alors sa contrainte doit être égale à $n-q_2 \geq q_1-1$. Cette restriction revient à imposer que la longueur des itérations soit positive ou nulle. Ceci n'est pas très contraignant en pratique : il semble naturel de considérer des itérations dont la longueur soit uniquement positive.[6] *À partir de maintenant, nous ne considérons donc que des schémas dont les itérations sont alignées sur* $[q_1; n-q_2]$ *et dont la contrainte a la forme* $n-q_2 \geq q_1-1$ *où* n *est le paramètre du schéma.*

Les deux opérations suivantes sont nécessaires à la définition de SchAut :

Definition 6.14. *Soit un schéma régulier* $m \,\&\, n-q_2 \geq q_1-1$.

- *la* base *de* m*, notée* $\mathrm{base}(m)$*, est définie par* $\mathrm{base}(m) \stackrel{def}{=} m[q_2+q_1-1/n]$*;*

[6] Par ailleurs, nous pouvons souvent nous ramener à un schéma dont la contrainte a cette forme, par élimination des quantificateurs et en s'aidant de la transformation évoquée en note de pied de page p.58.

- *l'*induction *de m, notée* $\mathrm{ind}(m)$*, est obtenue en remplaçant chaque itération* $\Pi_{i=q_1}^{n-q_2} m'$ *par* $m'[n - q_2/i] \,\pi\, \Pi_{i=q_1}^{n-q_2-1} m'$ *(le schéma obtenu alors est noté* $\mathrm{ind}(m)$*) puis en remplaçant chaque occurrence de n par $n+1$, où* π *est le connecteur de* π*.*

Example 6.15. *Soit l'exemple récurrent* $s \stackrel{def}{=} p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$*, alors* $q_1 = 0$*,* $q_2 = 1$*,* $\mathrm{base}(s) = p_0 \wedge \neg p_0$ *et* $\mathrm{ind}(s) = p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge p_{n+1} \Rightarrow p_{n+2} \wedge \neg p_{n+2}$*.*

Nous pouvons maintenant définir SchAut : cette procédure construit un automate prenant en entrée un entier représenté par un mot sur l'alphabet $\{0, \mathrm{s}\}$. Par exemple, 5 est représenté par $\mathrm{s} \cdot \mathrm{s} \cdot \mathrm{s} \cdot \mathrm{s} \cdot \mathrm{s} \cdot 0$. Dès lors l'automate a_s généré à partir du schéma s est conçu de façon à accepter un entier k ssi l'instance de s obtenue en fixant la longueur des itérations à k est (propositionnellement) satisfaisable, c.-à-d. ssi $\langle s \rangle_{\{n \mapsto k+q_1+q_2-1\}}$ est satisfaisable. Par conséquent s est insatisfaisable ssi le langage reconnu par l'automate est vide (problème dont la décidabilité est triviale).

Comme nous allons le voir, SchAut est très proche de Stab : en fait, en supprimant les cycles de l'automate, nous obtenons un arbre qui peut être interprété comme un tableau de Stab. Les cycles correspondent précisément aux applications de la règle de bouclage. Dans Stab chaque nœud était étiqueté par des motifs et des contraintes. Ici, les propriétés des schémas réguliers font que nous pouvons nous passer des contraintes. Les états de l'automate sont donc des ensembles de motifs uniquement.

Definition 6.16 (SchAut)**.** *Soit un schéma régulier* $s \stackrel{def}{=} m \,\&\, n - q_2 \geq q_1 - 1$*. Nous notons* a_s *l'automate construit par* SchAut *à partir de s. L'alphabet d'entrée de* a_s *est* $\{0, \mathrm{s}\}$ *et ses états sont des ensembles de motifs*[7] *alignés sur le même intervalle* $[q_1; n - q_2]$*.*

La construction de a_s *est donnée par un système de réécriture. La notation* $M \stackrel{x}{\Longrightarrow} M_1 \,\&\, M \stackrel{y}{\Longrightarrow} M_2$*, où M, M_1, M_2 sont des états, représente la règle* $\langle Q \cup \{M\}, q_0, F, T\rangle \rightarrow \langle Q \cup \{M, M_1, M_2\}, q_0, F, T \cup \{(M, x, M_1), (M, y, M_2)\}\rangle$*, c.-à-d. que nous ajoutons à l'automate deux états* M_1 *et* M_2 *avec les transitions* (M, x, M_1) *et* (M, y, M_2)*. De même,* $M \stackrel{x}{\Longrightarrow} M'$ *représente la règle* $\langle Q \cup \{M\}, q_0, F, T\rangle \rightarrow \langle Q \cup \{M, M'\}, q_0, F, T \cup \{(M, x, M')\}\rangle$*. Nous définissons alors* SchAut *comme l'ensemble de règles suivant :*

- $\wedge$ *:*
$$M \cup \{m_1 \wedge m_2\} \stackrel{\epsilon}{\Longrightarrow} M \cup \{m_1, m_2\}$$
- $\vee$ *:*
$$M \cup \{m_1 \vee m_2\} \stackrel{\epsilon}{\Longrightarrow} M \cup \{m_1\} \,\&\, M \cup \{m_1 \vee m_2\} \stackrel{\epsilon}{\Longrightarrow} M \cup \{m_2\}$$
- contradiction *:*
$$M \cup \{p_e, \neg p_e\} \stackrel{\epsilon}{\Longrightarrow} \{\bot\}$$
- pure *:*
$$M \cup p_e \stackrel{\epsilon}{\Longrightarrow} M \text{ si } p_e \text{ est pur dans } (\bigwedge_{m \in M} m) \,\&\, n - q_2 \geq q_1 - 1$$
- descente *:*
$$M \stackrel{0}{\Longrightarrow} \{\mathrm{base}(m) \mid m \in M\} \,\&\, M \stackrel{\mathrm{s}}{\Longrightarrow} \{\mathrm{ind}(m) \mid m \in M\} \text{ si } M \notin \{\emptyset, \{\top\}, \{\bot\}\}$$

Nous imposons de plus qu'une seule règle soit appliquée sur un état donné, et que descente *n'est appliquée que si aucune autre règle ne peut l'être.*

Nous notons alors a_s *une forme normale (arbitraire), par* SchAut*, de l'automate suivant :*

$$\langle \{S\}, \{m_s\}, \{\emptyset\}, \{(\emptyset, 0, \emptyset), (\emptyset, \mathrm{s}, \emptyset), (\{\top\}, \epsilon, \emptyset)\}\rangle$$

(nous montrerons en section 6.4 que SchAut *termine tout le temps).*

Example 6.17. *Soit l'exemple récurrent* $s \stackrel{def}{=} p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$ *(la contrainte est bien équivalente à* $n - 1 \geq 0 - 1$*), un automate possible pour* a_s *est donné en figure 6.1.*

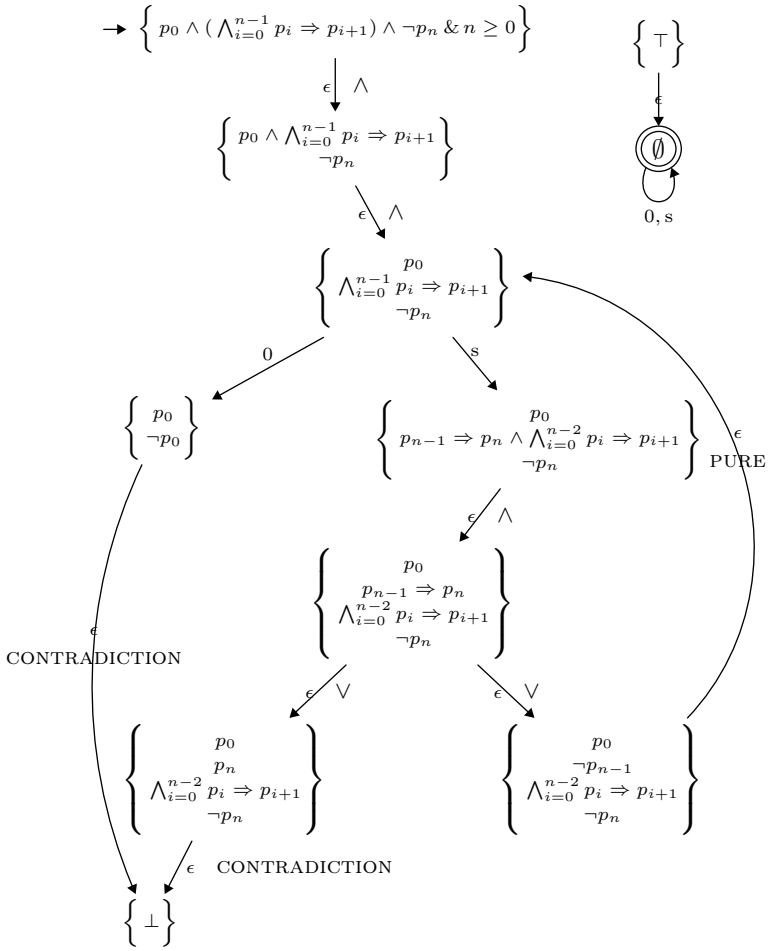


Figure 6.1: Application de SchAut à $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$

Notons que SchAut termine ssi l'ensemble d'états et de transitions ajoutés par la réécriture est fini. Quand SchAut termine, il peut y avoir plusieurs formes normales distinctes. Cependant ces formes normales ne sont distinctes que selon l'ordre des ϵ-règles et selon l'ordre dans lequel nous décomposons les formules. Nous pourrions imposer une forme normale unique en donnant une priorité entre les ϵ-règles et les formules mais ce n'est pas nécessaire pour la suite.

Proposition 6.18. *Soient un schéma s et un état M de a_s. Alors soit M appartient à $\{\emptyset, \{\top\}, \{\bot\}\}$, soit M s'est vu appliqué une et une seule des règles de* SchAut.

Si cette règle est $\wedge$ (resp. $\vee$, contradiction, pure, descente), M est appelé un $\wedge$-*état* (resp. $\vee$-*état*, *c-état*, *p-état*, *d-état*). Les $\wedge$-états, $\vee$-états et p-états sont appelés des *états propositionnels*. Pour un d-état M, M^0 et M^{s} désignent les états tels que $(M, 0, M^0)$ et $(M, \mathrm{s}, M^{\mathrm{s}})$ sont les transitions ajoutées à a_s par descente.

Enfin nous énonçons le théorème principal de SchAut :

Theorem 6.19. *Soit un schéma régulier s de contrainte $n - q_2 \geq q_1 - 1$. $s[k + q_1 + q_2 - 1/n]$ est satisfaisable ssi k est accepté par a_s.*

La preuve est similaire aux preuves de correction et complétude de Stab. Nous la présentons en annexe 6.A.

Remarquons que ce théorème est plus général que les résultats sur Stab (dans le cas des automates réguliers) car l'automate fournit une description explicite des valeurs du paramètre pour lesquelles le schéma est satisfaisable. A contrario, un tableau permet uniquement d'assurer l'*existence* d'une telle

[7] les motifs *sont* les états et pas seulement les étiquettes de ces états

valeur. De plus nous n'avons pas besoin d'utiliser de procédure pour l'arithmétique comme il n'y a pas de contrainte dans les états de l'automate (contrairement aux nœuds de STAB).

6.4 Complexity

Comme nous l'avons vu dans la preuve de complétude (6.2), le cœur de la terminaison réside dans l'élimination des littéraux purs et des contraintes redondantes. Avec SCHAUT, nous nous débarrassons des contraintes donc seule l'élimination des littéraux purs est importante. Nous nous intéressons maintenant à la complexité de cette procédure. Comme nous allons le voir, cela revient à se demander au bout de "combien de temps" un littéral devient pur. Nous pouvons ainsi obtenir une borne sur le nombre maximal d'états dans a_s, et en déduire la complexité de SCHAUT.

Note. Comme nous n'avons pas encore prouvé que SCHAUT termine, nous ne pouvons pas, rigoureusement, raisonner avec a_s pour montrer qu'il est fini. Dans la suite nous appellerons en fait a_s "l'automate limite" construit par SCHAUT, qui peut, *a priori*, être infini. Les définitions de la section 2.5 s'étendent de manière immédiate aux automates infinis.

6.4.1 Nombre maximal d'états

Dans toute la suite, s est un schéma régulier de contrainte $n - q_2 \geq q_1 - 1$. μ dénote le plus grand nombre apparaissant dans s, plus précisément, nous définissons $\mu(s)$ par induction :

$$\mu(\top) \stackrel{\text{def}}{=} \mu(\bot) \stackrel{\text{def}}{=} 0$$
$$\mu(p_{qn+r}) \stackrel{\text{def}}{=} \mu(\neg p_{qn+r}) \stackrel{\text{def}}{=} \max(|q|, |r|)$$
$$\mu(m_1 \wedge m_2) \stackrel{\text{def}}{=} \mu(m_1 \vee m_2) \stackrel{\text{def}}{=} \max(\mu(m_1), \mu(m_2))$$
$$\mu\left(\bigwedge_{i=q_1}^{n-q_2} m\right) \stackrel{\text{def}}{=} \mu\left(\bigvee_{i=q_1}^{n-q_2} m\right) \stackrel{\text{def}}{=} \max(|q_1|, |q_2|, \mu(m))$$

Ainsi le plus grand nombre apparaissant dans s est inférieur à $\mu(s)$ et le plus petit nombre est supérieur à $-\mu(s)$. Par la suite $\mu(s)$ sera simplement écrit μ. De plus simplifier les preuves, nous supposerons que $\mu > 0$.

Dans un premier temps, nous démontrons que les littéraux apparaissant dans un d-état quelconque sont en quantité finie. Nous distinguons différents cas selon la forme des littéraux considérés.

Littéraux avec indice entier.

Proposition 6.20. *Soit un littéral d'indice $q \in \mathbb{Z}$ apparaissant dans un d-état quelconque de a_s. Alors $-\mu \leq q \leq \mu$.*

Preuve. Les seuls littéraux dont l'indice est un entier sont ceux apparaissant dans s et ceux introduits par SCHAUT via *base* (dans DESCENTE). Comme nous considérons un d-état, *base* n'a pas encore été appliquée. Les seuls littéraux possibles sont donc ceux de s. Donc $s \in \mathrm{E}_q(s)$ et nous concluons avec la définition de $\mathrm{E}_q(s)$. □

Littéraux avec indice de la forme $n + q$, $q \in \mathbb{Z}$.

Proposition 6.21. *Soit un littéral d'indice $n+q$, $q \in \mathbb{Z}$, apparaissant dans* $\mathrm{ind}(s)$*, alors* $\mathrm{E_{n+}}_q(\mathrm{ind}(s)) \subseteq \{1 - q_2 + q' \mid q' \in \mathrm{E_{i+}}_q(s)\} \cup \{q' + 1 \mid q' \in \mathrm{E_{n+}}_q(s)\}$.

Preuve. $\mathrm{ind}(s)$ est obtenu en remplaçant chaque itération $\Pi_{i=q_1}^{n-q_2} m$ par $m[n - q_2/i] \sqcap \Pi_{i=q_1}^{n-q_2-1} m$ (noté, rappelons-le, $\mathrm{ind}'(s)$) puis en remplaçant chaque occurrence de n par $n + 1$. Donc tout littéral d'indice $n + q$ provient d'un littéral d'indice $n + q - 1$ apparaissant dans $\mathrm{ind}'(s)$. Comme ce dernier littéral contient n il apparaît nécessairement en dehors d'une itération. Donc soit c'est un littéral qui provient du remplacement d'une itération $\Pi_{i=q_1}^{n-q_2} m$ par $m[n - q_2/i] \sqcap \Pi_{i=q_1}^{n-q_2-1} m$, soit c'est un littéral qui était déjà dans s. Dans le premier cas ce littéral est issu de la substitution $[n - q_2/i]$ appliquée à un littéral dont l'indice a la forme $i + q'$, donc $q - 1 \in \{q_2 + q' \mid q' \in \mathrm{E_{i+}}_q(s)\}$. Dans le deuxième cas, comme l'indice $n + q - 1$ apparaissait déjà dans s, nous avons : $q - 1 \in \{q' + 1 \mid q' \in \mathrm{E_{n+}}_q(s)\}$. □

Soit un d-état M de a_s. Pour toute course de a_s partant de $\{s\}$ et arrivant en M, nous appelons *degré de dépliage* le nombre de d-états qu'elle contient. Le *degré de dépliage* de M est le minimum des degrés de dépliage parmi toutes les courses arrivant en M. Notons que tout d-état a un degré de dépliage, en effet par construction de a_s il existe nécessairement une course menant de a_{m_s} à M.

Proposition 6.22. *Soit un littéral d'indice $n+q$ apparaissant dans un d-état de degré de dépliage $k \geq 1$, avec $k \in \mathbb{Z}$. Alors $-2\mu \leq q \leq 2\mu + k$.*

Preuve. DESCENTE est la seule règle à introduire de nouveaux littéraux avec les opérations base et ind. De ces deux opérations, seule ind peut introduire des littéraux dont l'indice contient n. Nous obtenons donc, par récurrence sur k et avec la proposition précédente : $\min(-q_2 + \min(\mathrm{E}_{\mathrm{i}+q}(s)), 1 + \min(\mathrm{E}_{\mathrm{n}+q}(s))) \leq q$ et $q \leq k + \max(-q_2 + \max(\mathrm{E}_{\mathrm{i}+q}(s)), \max(\mathrm{E}_{\mathrm{n}+q}(s)))$ (pour la borne inférieure, nous utilisons le fait que $k \geq 1$). □

Littéraux avec indice de la forme $rn + q$, $r \neq 1$, $q \in \mathbb{Z}$.

Proposition 6.23. *Soit un littéral d'indice $rn + q$, $r \neq 1$, $q \in \mathbb{Z}$, apparaissant dans* $\mathrm{ind}(s)$*, alors* $\mathrm{E}_{qn}(\mathrm{ind}(s)) \subseteq \mathrm{E}_{qn}(s)$ *et* $\mathrm{E}_{\mathrm{n}+q}(\mathrm{ind}(s)) \subseteq \{r + q' \mid q' \in \mathrm{E}_{\mathrm{n}+q}(s)\}$.

Preuve. Tout littéral d'indice $rn + q$ provient d'un littéral d'indice $r(n-1) + q$, c.-à-d. $rn + q - r$, apparaissant dans $\mathrm{ind}'(s)$. Comme ce dernier littéral contient n il apparaît nécessairement en dehors d'une itération. Donc soit c'est un littéral qui provient du remplacement d'une itération $\Pi_{i=q_1}^{n-q_2} m$ par $m[n - q_2/i] \amalg \Pi_{i=q_1}^{n-q_2-1} m$, soit c'est un littéral qui était déjà dans s. Dans le premier cas ce littéral ne peut qu'être issu de l'application de la substitution $[n - q_2/i]$ à un littéral dont l'indice a la forme $i + q'$, $q' \in \mathbb{Z}$, ce qui est impossible car $r \neq 1$. Ne reste que le deuxième cas : comme l'indice $rn + q - r$ apparaissait déjà dans s, nous avons $r \in \mathrm{E}_{qn}(s)$ et $q - r \in \mathrm{E}_{\mathrm{n}+q}(s)$. Donc $q \in \{r + q' \mid q' \in \mathrm{E}_{\mathrm{n}+q}(s)\}$. □

Proposition 6.24. *Soit un littéral d'indice $rn+q$ apparaissant dans un d-état de degré de dépliage $k \geq 1$, avec $k \in \mathbb{Z}$. Alors $-\mu \leq r \leq \mu$ et $kr - \mu \leq q \leq kr + \mu$.*

Preuve. Conséquence de la proposition précédente, par récurrence sur k. □

Preuve principale.

Nous notons $\#s$ la taille de s en nombre de symboles. Les entiers sont codés sous forme de suites soit unaires soit binaires. Pour rester le plus général possible nous utiliserons la notation suivante: $\#_{\mathrm{int}} s$ représente le plus grand entier pouvant apparaître dans s; si les entiers sont codés en binaire $\#_{\mathrm{int}} s = 2^{\#s}$, s'ils sont codés en unaire $\#_{\mathrm{int}} s = \#s$. Nous avons $\mu \leq \#_{\mathrm{int}} s$.

Lemma 6.25. *Il existe $q_0 \in \mathbb{Z}$ tel que $q_0 \in O(\#_{\mathrm{int}} s)$ et tout littéral d'indice $n + q$, $q \in \mathbb{Z}$, apparaissant dans un d-état quelconque est pur si $q > q_0$.*

Preuve. Soit un tel littéral appartenant à un d-état de degré de dépliage $k \geq 1$. Ce littéral est pur ssi, pour toute proposition indexée de signe opposé et d'indice e, et tout environnement ρ satisfaisant $\rho(n) - q_2 \geq q_1 - 1$, $\rho(n + q) \neq \rho(e)$. Nous détaillons chaque cas pour e :

- Si e a la forme $n + q'$, $q' \in \mathbb{Z}$, alors nous ne pouvons avoir $\rho(n + q) = \rho(e)$ que si $q = q'$, ce qui est impossible car nous pourrions alors appliquer CONTRADICTION : contradiction avec le fait que nous considérons un d-état.

- Si e a la forme $rn + q'$, $r, q' \in \mathbb{Z}$, $r > 1$, alors nous définissons k_0 tel que si $k > k_0$, alors : $r\rho(n) + q' > \rho(n) + q$ pour tout environnement ρ, c.-à-d. $(r-1)\rho(n) + q' - q > 0$. D'après l'hypothèse que $r > 1$, les propositions 6.24 et 6.22 et le fait que $\rho(n) - q_2 \geq q_1 - 1$, nous savons que :

$$\begin{aligned} r &\geq 2 \\ \rho(n) &\geq q_1 + q_2 - 1 \geq -2\mu - 1 \\ q' &\geq kr - \mu \geq 2k - \mu \\ q &\leq 2\mu + k \end{aligned}$$

Nous avons donc $(r-1)\rho(n)+q'-q \geq -5\mu-1+k$. Nous pouvons donc prendre k_0 tel que $-5\mu-1+k_0 \geq 0$, p.ex. $k_0 \stackrel{\text{def}}{=} 5\mu+1$. Soit $q_0 \stackrel{\text{def}}{=} 2\mu+k_0$. D'après la proposition 6.22, il est facile de voir que si $q > q_0$ alors $k > k_0$. De plus, vue la forme de q_0 (et de k_0), nous avons bien $q_0 \in O(\#_{\text{int}}s)$.

- Si e a la forme $rn+q'$, $r, q' \in \mathbb{Z}$, $r < 0$, alors nous définissons k_0 tel que si $k > k_0$, alors : $r\rho(n)+q' < \rho(n)+q$ pour tout environnement ρ, c.-à-d. $(1-r)\rho(n)+q-q' > 0$. D'après l'hypothèse que $r < 0$, les propositions 6.24 et 6.22 et le fait que $\rho(n)-q_2 \geq q_1-1$, nous savons que :
$$\begin{aligned} 1-r &\geq 2 \\ \rho(n) &\geq q_1+q_2-1 \geq -2\mu-1 \\ q &\geq -2\mu \\ q' &\leq kr+\mu \end{aligned}$$
Nous avons donc $(1-r)\rho(n)+q-q' \geq 2(-2\mu-1)-kr-3\mu = -7\mu-kr-2$. Et, comme $r \leq 1$, $(1-r)\rho(n)+q-q' \geq -7\mu+k-2$. Nous pouvons donc prendre k_0 tel que $-7\mu+k_0-2 \geq 0$, p.ex. $k_0 \stackrel{\text{def}}{=} 7\mu+2$. Soit $q_0 \stackrel{\text{def}}{=} 2\mu+k_0$. D'après la proposition 6.22, il est facile de voir que si $q > q_0$ alors $k > k_0$. De plus, vue la forme de q_0 (et de k_0), nous avons bien $q_0 \in O(\#_{\text{int}}s)$.

- Si e est un entier alors nous définissons q_0 tel que si $q > q_0$, $\rho(n)+q > e$ pour tout environnement ρ, c.-à-d. $\rho(n)+q-e > 0$. D'après la proposition 6.20 et le fait que $\rho(n)-q_2 \geq q_1-1$, nous savons que $\rho(n) \geq -2\mu-1$ et $e \leq \mu$. Nous posons donc directement $q_0 \stackrel{\text{def}}{=} 3\mu+1$. Une fois encore, nous avons bien $q_0 \in O(\#_{\text{int}}s)$.

- Si e est de la forme $i+q'$, alors nous définissons q_0 tel que si $q > q_0$, alors $\rho(n)+q > \rho \circledcirc \{i \mapsto I\}(i+q')$ pour tout environnement ρ et tout $I \in [q_1; \rho(n)-q_2]$. Cette inéquation équivaut à $q > I+q'-\rho(n)$. Comme $I \leq \rho(n)-q_2$, elle est vraie si $q > -q_2+q'$. C'est bien le cas en particulier si $q > \mu$. Nous posons donc $q_0 \stackrel{\text{def}}{=} \mu$. Une fois encore, $q_0 \in O(\#_{\text{int}}s)$.

Au final nous prenons pour q_0 le maximum des q_0 présentés dans tous les cas. Nous avons donc toujours la propriété $q_0 \in O(\#_{\text{int}}s)$. □

Lemma 6.26. *Il existe $q_0 \in \mathbb{Z}$ tels que $q_0 \in O((\#_{\text{int}}s)^2)$ et tout littéral d'indice $rn+q$, $r, q \in \mathbb{Z}$, $r > 1$, apparaissant dans un d-état quelconque est pur si $q > q_0$.*

Preuve. Soit un tel littéral appartenant à un d-état de degré de dépliage $k \geq 1$. Ce littéral est pur si, pour toute proposition indexée de signe opposé et d'indice e, et tout environnement ρ satisfaisant $\rho(n)-q_2 \geq q_1-1$, $\rho(rn+q) \neq \rho(e)$. Nous détaillons chaque cas pour e :

- Si e a la forme $n+q'$, $q' \in \mathbb{Z}$ alors nous reprenons le k_0 du cas correspondant dans la preuve du lemme 6.25. Ainsi si $k > k_0$, le littéral est bien pur. Soit $q_0 \stackrel{\text{def}}{=} k_0\mu+\mu$. D'après la proposition 6.24, il est facile de voir que si $q > q_0$ alors $k > k_0$. De plus, vue la forme de q_0 et de k_0, nous avons bien $q_0 \in O((\#_{\text{int}}s)^2)$.

- Si e a la forme $r'n+q'$, $r', q' \in \mathbb{Z}$, $r' = r$. Alors le littéral est pur ssi $q \neq q'$, ce qui est nécessairement le cas : autrement, CONTRADICTION aurait été appliquée.

- Si e a la forme $r'n+q'$, $r, q \in \mathbb{Z}$, $r' > r$. Nous définissons k_0 tel que si $k > k_0$, alors : $r\rho(n)+q < r'\rho(n)+q'$ pour tout environnement ρ, c.-à-d. $(r'-r)\rho(n)+q'-q > 0$. D'après l'hypothèse que $r' > r$, la proposition 6.24 et le fait que $\rho(n)-q_2 \geq q_1-1$, nous savons que :
$$\begin{aligned} r'-r &\geq 1 \\ \rho(n) &\geq q_1+q_2-1 \geq -2\mu-1 \\ q' &\geq kr'-\mu \\ q &\leq kr+\mu \end{aligned}$$
Nous avons donc $(r'-r)\rho(n)+q'-q \geq k(r'-r)-4\mu-1 \geq k-4\mu-1$. Nous pouvons donc prendre k_0 tel que $k-4\mu-1 > 0$, p.ex. $k_0 \stackrel{\text{def}}{=} 4\mu+1$. Ainsi si $k > k_0$ le littéral est bien pur. Comme dans le premier cas, nous posons ensuite $q_0 \stackrel{\text{def}}{=} k_0\mu+\mu$, ce qui permet d'obtenir le résultat avec la proposition 6.24. De plus, vue la forme de q_0 (et de k_0), nous avons bien $q_0 \in O((\#_{\text{int}}s)^2)$.

- Si e a la forme $r'n+q'$, $r,q \in \mathbb{Z}$, $r' < r$, nous obtenons : $r\rho(n)+q > r'\rho(n)+q'$ pour tout environnement ρ en prenant le même k_0 que dans le cas précédent.

- Si e est un entier alors nous définissons q_0 tel que si $q \geq q_0$, alors $r\rho(n)+q > e$ pour tout environnement ρ. D'après la proposition 6.20 et le fait que $\rho(n) - q_2 \geq q_1 - 1$, nous avons :

$$\begin{aligned} \rho(n) &\geq q_1 + q_2 - 1 \geq -2\mu - 1 \\ r &\geq 2 \\ e &\leq \mu \end{aligned}$$

  Donc $r\rho(n)+q-e \geq -2(2\mu+1)+q-\mu = -5\mu-2+q$. Il suffit donc de prendre q_0 tel que $-5\mu-2+q>0$, p.ex. $q_0 \stackrel{\text{def}}{=} 5\mu+2$.

- Si e est de la forme $i+q'$, alors nous définissons q_0 tel que si $q > q_0$, alors $r\rho(n)+q \geq \rho \circledcirc \{i \mapsto I\}(i+q')$ pour tout environnement ρ et tout $I \in [q_1; \rho(n)-q_2]$. Cette inéquation équivaut à $q > I+q'-r\rho(n)$. Comme $I \leq \rho(n)-q_2$, elle est vraie si $q > -q_2+q'+(1-r)\rho(n)$. C'est bien le cas en particulier si $q > 2\mu+(1-\mu)(-2\mu-1)$. Nous posons donc $q_0 \stackrel{\text{def}}{=} 2\mu+(1-\mu)(-2\mu-1)$, d'où le résultat. Une fois encore, $q_0 \in O((\#_{\text{int}}s)^2)$.

Au final nous prenons pour q_0 le maximum des q_0 présentés dans tous les cas. Nous avons donc toujours la propriété $q_0 \in O((\#_{\text{int}}s)^2)$. □

Lemma 6.27. *Il existe $q_0 \in \mathbb{Z}$ tels que $q_0 \in O((\#_{\text{int}}s)^2)$ et tout littéral d'indice $rn+q$, $r,q \in \mathbb{Z}$, $r<0$, apparaissant dans un d-état quelconque est pur si $q < q_0$.*

Preuve. Soit un tel littéral appartenant à un d-état de degré de dépliage $k \geq 1$. Ce littéral est pur si, pour toute proposition indexée de signe opposé et d'indice e, et tout environnement ρ satisfaisant $\rho(n)-q_2 \geq q_1-1$, $\rho(rn+q) \neq \rho(e)$. Nous détaillons chaque cas pour e :

- Si e a la forme $n+q'$, $q' \in \mathbb{Z}$ alors nous définissons q_0 tel que si $q > q_0$, $r\rho(n)+q < \rho(n)+q'$ pour tout environnement ρ, c.-à-d. $(1-r)\rho(n)+q'-q > 0$. D'après l'hypothèse que $r<0$, les propositions 6.22 et 6.24, et le fait que $\rho(n)-q_2 \geq q_1-1$, nous savons que :

$$\begin{aligned} 1-r &\geq 1 \\ \rho(n) &\geq q_1+q_2-1 \geq -2\mu-1 \\ q' &\geq -2\mu \\ q &\leq kr+\mu \end{aligned}$$

  Ainsi, $(1-r)\rho(n)+q'-q \geq -kr-5\mu-1 \geq k-5\mu-1$ (car $r \leq -1$). Donc il suffit que $k > 5\mu+1$. Nous posons donc $k_0 \stackrel{\text{def}}{=} 5\mu+1$. Puis nous posons $q_0 \stackrel{\text{def}}{=} -k_0\mu-\mu$. D'après la proposition 6.24, il est facile de voir que si $q < q_0$ alors $k > k_0$. De plus, vue la forme de q_0 et de k_0, nous avons bien $q_0 \in O((\#_{\text{int}}s)^2)$.

- Si e a la forme $r'n+q'$, $r',q \in \mathbb{Z}$, $r'=r$. Alors le littéral est pur ssi $q \neq q'$, ce qui est nécessairement le cas : autrement, CONTRADICTION aurait été appliquée.

- Si e a la forme $r'n+q'$, $r,q \in \mathbb{Z}$, alors nous obtenons le résultat, que nous ayons $r' > r$ ou $r' < r$, en reprenant le même k_0 que dans le cas correspondant de la preuve précédente. Puis nous posons $q_0 \stackrel{\text{def}}{=} -k_0\mu-\mu$, ce qui permet de conclure comme dans le premier cas.

- Si e est un entier alors nous définissons q_0 tel que si $q \geq q_0$, alors $r\rho(n)+q < e$ pour tout environnement ρ, c.-à-d. $q < e-r\rho(n)$. D'après l'hypothèse que $r<0$, la proposition 6.20 et le fait que $\rho(n)-q_2 \geq q_1-1$, nous avons :

$$\begin{aligned} \rho(n) &\geq q_1+q_2-1 \geq -2\mu-1 \\ r &\leq -1 \\ e &\geq -\mu \end{aligned}$$

  Donc $e-r\rho(n) \geq -3\mu-1$. Il suffit donc de prendre q_0 tel que $-3\mu-1 > q$, p.ex. $q_0 \stackrel{\text{def}}{=} -3\mu-1$.

- Si e est de la forme $i + q'$, alors nous définissons q_0 tel que si $q > q_0$, alors $r\rho(n) + q < \rho \circledcirc \{i \mapsto I\}(i + q') + q'$ pour tout environnement ρ et tout $I \in [q_1; \rho(n) - q_2]$. Cette inéquation équivaut à $q < I + q' - r\rho(n)$. Comme $I \geq q_1$ et $r \leq -1$, elle est vraie si $q < q_1 + q' + \rho(n)$. C'est bien le cas en particulier si $q < -2\mu + 1$. Nous posons donc $q_0 \stackrel{\text{def}}{=} -2\mu + 1$ ce qui permet de conclure. Une fois encore, $q_0 \in O((\#_{\text{int}} s)^2)$.

Au final nous prenons pour q_0 le minimum des q_0 présentés dans tous les cas. Nous avons donc toujours la propriété $q_0 \in O((\#_{\text{int}} s)^2)$. □

Lemma 6.28. *Il existe un ensemble fini contenant tous les littéraux apparaissant dans un d-état quelconque. Cet ensemble a $O((\#_{\text{int}} s)^2)$ éléments.*

Preuve. Un littéral est soit positif ou négatif, il contient un symbole propositionnel et un indice donc il y a $2n_{\text{symb}}n_{\text{idx}}$ littéraux possibles, où n_{symb} (resp. n_{idx}) est le nombre de symboles propositionnels (resp. d'indices). Il y a évidemment une quantité finie de symboles propositionnels dans s, et, de plus, $n_{\text{symb}} \leq \#s$. Par ailleurs il y a quatre formes d'indices :

- Les indices entiers, dont nous avons vu au proposition 6.20 qu'ils étaient en quantité finie.

- Les indices de la forme $n + q$. La proposition 6.22 assure que q a une borne inférieure, et le lemme 6.25 assure que q a une borne supérieure (si q dépasse la borne alors le littéral est éliminé par PURE). Donc q appartient à un ensemble fini.

- Les indices de la forme $rn + q$, $r \geq 1$. La proposition 6.24 assure que r appartient à un ensemble fini et donne une borne inférieure pour q (en prenant $k = 1$). De plus le lemme 6.26 donne une borne supérieure pour q.

- Les indices de la forme $rn + q$, $r < 0$. La proposition 6.24 assure que r appartient à un ensemble fini et donne une borne *supérieure* pour q (en prenant $k = 1$). De plus le lemme 6.26 donne une borne *inférieure* pour q.

D'autre part, toutes les bornes mises en jeu sont dans $O((\#_{\text{int}} s)^2)$ donc l'ensemble de tous les littéraux possibles a une taille dans $O((\#_{\text{int}} s)^2)$. □

Theorem 6.29. *(i) a_s contient au plus $O(2^{\#_{\text{int}} s})$ états. (ii) Chaque état contient au plus $O((\#_{\text{int}} s)^2)$ motifs.*

Preuve. Nous nous restreignons d'abord aux d-états. Comme DESCENTE n'est appliquée que si aucune autre règle ne peut s'appliquer, tout d-état ne contient que des itérations et des littéraux. Il est facile de voir que chaque itération apparaissant dans un état de a_s apparaît nécessairement dans s, donc il y en a au maximum $O(\#s)$. D'après le lemme 6.28, il y a au plus $O((\#_{\text{int}} s)^2)$ littéraux dans un d-état, ce qui est toujours supérieur à $O(\#s)$. Donc, au total, l'ensemble de tous les littéraux et itérations possibles a pour taille $O((\#_{\text{int}} s)^2)$. Comme un d-état est un *sous-ensemble* de cet ensemble, nous avons donc $O((\#_{\text{int}} s)^2)$ d-états possibles, c.-à-d. $O(2^{\#_{\text{int}} s})$. Nous obtenons donc (i) et (ii) pour les d-états.

Considérons deux d-états pour lesquels il existe un chemin les reliant et ne contenant pas d'autre d-état. Alors le nombre d'états dans ce chemin est linéaire par rapport au nombre de motifs présents dans le premier état (en effet les règles autres que DESCENTE ne font que décomposer les motifs). Nous venons de voir que ce nombre est en $O((\#_{\text{int}} s)^2)$. Par ailleurs il y a au pire $k.(k-1)$ tels chemins, où k est le nombre de d-états de a_s. Comme $k = O(2^{\#_{\text{int}} s})$, le nombre d'états entre tous les d-états n'augmente pas l'ordre de grandeur du nombre de d-états. Il peut rester des états entre l'état initial et l'état final, entre l'état initial et un d-état, et entre un d-état et un état final. Tous ces cas sont traités de la même manière. Au final chaque état de a_s appartient à un de ces chemins et l'ordre de grandeur du nombre d'états dans a_s est donc le même que celui du nombre de d-états. □

Corollary 6.30. SCHAUT *termine.*

6.4.2 Conséquences

Tout d'abord, comme le problème du vide est décidable pour les automates, nous retrouvons le fait que l'insatisfaisabilité des schémas réguliers est décidable. Et nous avons donc à nouveau la complétude des schémas réguliers pour la réfutation.

Nous obtenons de plus une sorte de "propriété du modèle borné" :

Theorem 6.31. *Tout schéma régulier satisfaisable s a un modèle dont le paramètre est en $O(2^{\#_{\text{int}}s})$.*

Preuve. En effet si l'automate n'est pas vide, il existe une course acceptante. Et dans ce cas il existe une course acceptante qui passe au plus une fois par chaque état. Comme nous avons au plus $O(2^{\#_{\text{int}}s})$ d-états, une telle course passe au plus $O(2^{\#_{\text{int}}s})$ fois par une transition étiquetée par s. Donc la valeur donnée au paramètre est au plus $O(2^{\#_{\text{int}}s})$. □

En précisant la borne de sorte à éliminer le $O()$, nous pouvons ainsi obtenir une autre procédure de décision, bien que celle-ci serait probablement très inefficace.

Évidemment, la borne sur le nombre d'états de a_s jette les base pour une étude de complexité. Notons d'abord que les schémas réguliers incluent les formules de la logique propositionnelle. Nous savons donc d'ores et déjà que la satisfaisabilité des schémas réguliers est NP-dur. Mais SchAut permet de donner une borne supérieure. Nous commençons par le lemme suivant :

Lemma 6.32. *Déterminer s'il existe une interprétation telle que deux littéraux ont le même indice peut être fait en temps polynomial par rapport à la taille des indices.*

Ce calcul est utile pour l'élimination des littéraux purs.

Preuve. Soient deux indices $qn + r$ et $q'n + r'$. Si $q = q'$ alors il suffit de vérifier que $r = r'$. Si $q \neq q'$ alors il y a égalité ssi $n = \frac{r'-r}{q-q'}$. Il suffit donc de s'assurer que $q - q'$ divise $r - r'$ et que, dans ce cas, $\frac{r'-r}{q-q'} \geq q_2 + q_1 - 1$. Toutes ces opérations peuvent être effectuées en temps linéaire par rapport à la taille des entiers apparaissant dans les indices. □

Theorem 6.33. SchAut *termine en $O(2^{\#_{\text{int}}s})$.*

Preuve. Le théorème 6.29 (ii) fournit la propriété la plus importante : le nombre d'états de a_s est en $O(2^{\#_{\text{int}}s})$. Il reste à nous assurer que toutes les étapes de la construction de a_s sont de complexité inférieure.

Nous montrons d'abord que l'application de chaque règle est polynomiale par rapport à la taille de l'état réécrit, elle-même en $O((\#_{\text{int}}s)^2)$ d'après le théorème 6.29 (i). Par conséquent la complexité de chaque règle sera inférieure à $O(2^{\#_{\text{int}}s})$. Chaque état provient de l'application d'une règle, il y a donc autant d'applications de règles que d'états. Nous pourrons donc conclure que le calcul de la réécriture est en $O(2^{\#_{\text{int}}s})$. Montrons donc que l'application de chaque règle est polynomiale par rapport à la taille de l'état réécrit. Les règles $\wedge$ et $\vee$ ne font que décomposer un motif selon sa structure, ce qui peut être fait en temps constant. En revanche il faut trouver, dans l'ensemble de motifs, celui sur lequel appliquer la règle, ce qui peut être largement fait en temps linéaire. contradiction recherche dans l'ensemble une paire de littéraux complémentaires, ce qui peut être fait en temps quadratique. Le calcul de base et ind se fait clairement en temps linéaire. Il en va donc de même pour descente.

Enfin il reste pure : d'après le lemme précédent, nous savons que déterminer si deux littéraux peuvent avoir le même indice peut être fait en temps polynomial. Ainsi si nous parcourons tous les indices deux à deux alors l'ensemble de l'opération reste polynomiale. Cependant il faut aussi tester, pour un littéral donné, si celui-ci peut apparaître dans une itération. Une façon simple de tester ceci est de traduire ce problème en une formule de l'arithmétique linéaire et de s'assurer de la validité de cette formule avec une procédure de décision dédiée. Cependant, comme l'arithmétique linéaire est dans 2-EXPTIME [FR74], il faut analyser précisément la formule pour assurer que nous n'atteignons pas cette borne dans ce cas. Nous utilisons donc une autre approche : nous modifions SchAut de sorte à ce que nous utilisons une méthode incomplète pour la détection de littéraux purs : certains littéraux purs ne seront pas détectés comme tels, mais la terminaison est préservée et la complexité facile à déterminer. Nous partons de la constatation que si l'indice d'un littéral est supérieur à $n - q_2 + \max(\mathrm{E}_{\mathrm{i}+q}(s))$ ou inférieur à $q_1 + \min(\mathrm{E}_{\mathrm{i}+q}(s))$, alors nous sommes certain que le littéral n'apparaîtrait pas dans une itération. Dans ce cas nous pouvons effectivement ne tester que les littéraux en dehors des itérations. En opérant ainsi, certains littéraux

purs ne sont pas éliminés, ce qui est contraire à la définition de SchAut. Cependant il est facile de voir que cette légère modification préserve le théorème principal de SchAut. En revanche il faut aussi s'assurer que le nombre total de littéraux possibles reste en $O((\#_{\text{int}} s)^2)$ (ce qui n'est pas acquis car cette modification fait que nous n'éliminons pas des littéraux qui devraient normalement être éliminés). En examinant les preuves des lemmes 6.25, 6.26 et 6.27, nous voyons que les bornes dont nous montrons l'existence dans ces lemmes sont justement calculées en considérant que les littéraux purs ne peuvent pas apparaître dans les itérations. Ainsi les bornes restent les mêmes et la complexité est préservée. Enfin il est facile de voir qu'il est vérifiable en temps polynomial que l'indice d'un littéral est supérieur à $n - q_2 + \max(\mathrm{E}_{\mathrm{i}+q}(s))$ ou inférieur à $q_1 + \min(\mathrm{E}_{\mathrm{i}+q}(s))$.

Nous étudions maintenant la réécriture elle-même : il faut rechercher un état sur lequel aucune règle n'a encore été appliquée. Ceci se fait très naturellement via une pile ou une file f.i.f.o. Dans les deux cas il est facile d'avoir une insertion et une consultation en temps constant (plus précisément logarithmique par rapport à la taille de la pile si nous tenons compte de l'arithmétique sur les pointeurs). Ces opérations sont donc sans incidence sur la complexité globale.

Enfin la dernière opération coûteuse est la détection de cycle, c.-à-d. lorsque l'une des règles ajoute un état, il faut s'assurer que cet état n'est pas déjà présent. Nous pouvons détecter cela de façon linéaire par rapport au nombre d'états. En revanche, l'égalité entre états n'est pas linéaire car ces états sont des *ensembles* donc l'égalité est plus complexe que pour de simples listes. Mais il est facile d'imposer une forme normale sur leur représentation (typiquement en fixant un ordre sur les motifs) de sorte à pouvoir utiliser l'égalité entre listes. Maintenir de telles formes normales se fait aisément en temps logarithmique par rapport à la taille de l'état (p.ex. avec des arbres binaires équilibrés). □

Corollary 6.34. *Le problème de la satisfaisabilité pour les schémas réguliers est dans 2-EXPTIME. Si les entiers apparaissant dans le schéma en entrée sont codés en unaire, alors elle est dans EXPTIME.*

Theorem 6.35. *La borne sur* SchAut *est aussi une borne inférieure : il existe une suite de schémas réguliers $(s_k)_{k\in\mathbb{N}}$ t.q.* SchAut *termine en $O(2^{\#_{\text{int}} s_k})$.*

C'est un problème ouvert que de savoir si cette borne inférieure est propre à SchAut ou bien valide pour toute procédure sur les schémas réguliers.

Preuve. Soit $k \in \mathbb{N}$. Nous construisons un schéma s_k tel que a_{s_k} contient nécessairement $O(2^{\#_{\text{int}} s_k})$ états, d'où le résultat. Plus précisément, nous considérons tous les nombres encodés sur k bits $(\mathrm{A}_0, \mathrm{A}_1, \ldots, \mathrm{A}_{2^k-1})$ et nous nous arrangeons pour avoir autant d'états. Nous aurons donc 2^k états. De plus, k apparaît dans s_k, nous verrons donc que $k = \theta(\#_{\text{int}} s_k)$, d'où le résultat.

L'encodage binaire de chacun des A_0, A_1, etc. est représenté par k propositions indexées : A_0 est représenté par $\mathrm{a}_1, \ldots, \mathrm{a}_k$, A_1 par $\mathrm{a}_{k+1}, \ldots, \mathrm{a}_{k+k}$, et de façon générale A_i est représenté par $\mathrm{a}_{ik+1}, \ldots, \mathrm{a}_{ik+k}$. Nous utilisons une proposition indexée d_i pour indiquer si i est le *d*ernier bit d'un de ces nombres (c.-à-d. que nous ayons $\mathrm{d}_k, \mathrm{d}_{2k}, \mathrm{d}_{3k}, \ldots$), ce qui est exprimé par :

$$\text{dernier} \stackrel{\text{def}}{=} \mathrm{d}_0 \wedge \left(\bigwedge_{i=0}^{n} \mathrm{d}_i \Rightarrow \mathrm{d}_{i+k} \right)$$

Par ailleurs, nous ordonnons les A_i de sorte à ce que $\mathrm{A}_{i+1} = \mathrm{A}_i + 1$ pour tout $i \in \mathbb{N}$. À cette fin, nous utilisons une retenue r_i :

$$\text{incr} \stackrel{\text{def}}{=} \left(\bigwedge_{i=0}^{n} \mathrm{d}_i \Rightarrow \mathrm{r}_{i+1} \right) \wedge (\mathrm{a}_{i+k} \Leftrightarrow \mathrm{r}_i \oplus \mathrm{a}_i) \wedge (\neg \mathrm{d}_{i+1} \Rightarrow (\mathrm{r}_{i+1} \Leftrightarrow (\mathrm{r}_i \wedge \mathrm{a}_i)))$$

Finalement nous imposons que la suite des A_i commence à 0, ce qui est fait en utilisant une proposition indexée c_i (comme *c*ommence) :

$$\text{commence} \stackrel{\text{def}}{=} \mathrm{c}_1 \wedge \left(\bigwedge_{i=0}^{n} \mathrm{c}_i \wedge \neg \mathrm{d}_i \right) \Rightarrow \mathrm{c}_{i+1} \wedge \neg \mathrm{a}_i$$

Nous posons enfin :

$$s_k \stackrel{\text{def}}{=} \text{commence} \wedge \text{incr} \wedge \text{dernier} \,\&\, n \geq 0$$

Comme k apparaît explicitement dans s_k et est le plus grand entier apparaissant dans le schéma, nous avons $k = \theta(\#_{\text{int}} s_k)$. Donc $k = \theta(2^{s_k})$. Enfin il est facile de voir que SchAut va nécessairement générer 2^k, donc $2^{\#_{\text{int}} s_k}$, états. □

Notons finalement que si nous ne considérons que des schémas tels que $\mathrm{E}_{\mathrm{i}+q}(s)$ soit inclus dans une intervalle *fixé*, alors la complexité est une simple exponentielle, *quel que soit l'encodage des entiers.* C'est intéressant parce qu'il est possible d'augmenter la taille d'un schéma sans augmenter la taille de $\mathrm{E}_{\mathrm{i}+q}(s)$. En pratique même, pour la plupart des schémas, $\mathrm{E}_{\mathrm{i}+q}(s) = \{0, 1\}$ ou $\mathrm{E}_{\mathrm{i}+q}(s) = \{-1, 0, 1\}$. Une piste de travail consisterait à étudier une sous-classe des schémas réguliers pour laquelle on aurait cette propriété. Nous conjecturons que le problème de la satisfaisabilité pour de telles classes pourrait être dans NP.

Annexe

6.A Preuve du théorème principal de SchAut

Nous démontrons maintenant le théorème 6.19 :

Theorem 6.19. *Soit un schéma régulier s de contrainte $n - q_2 \geq q_1 - 1$. $s[k + q_1 + q_2 - 1/n]$ est satisfaisable ssi k est accepté par a_s.*

Les états d'un automate généré par SchAut sont interprétés comme la conjonction de leurs éléments. Nous pouvons voir le lemme suivant comme étant à SchAut ce que le lemme 4.3 est à Stab.

Lemma 6.36. *Soient un schéma régulier s de contrainte $n - q_2 \geq q_1 - 1$ et $k \in \mathbb{Z}$ t.q. $k - q_2 \geq q_1 - 1$. Alors, pour tout état M de a_s, $\langle M \rangle_{\{n \mapsto k\}}$ est satisfaisable ssi :*

- *si M est un état propositionnel : a_s contient une transition (M, ϵ, M') pour un certain M' t.q. $\langle M' \rangle_{\{n \mapsto k\}}$ est satisfaisable;*
- *si M est un d-état et $k = q_1 + q_2 - 1$: M^0 est satisfaisable;*
- *si M est un d-état et $k > q_1 + q_2 - 1$: $\langle M^{\mathrm{s}} \rangle_{\{n \mapsto k-1\}}$ est satisfaisable;*
- *ou M appartient à $\{\emptyset, \{\top\}\}$.*

Preuve. D'après la proposition 6.18, soit M appartient à $\{\emptyset, \{\top\}, \{\bot\}\}$, soit M est un c-état ou un état propositionnel ou un d-état.

Si M appartient à $\{\emptyset, \{\top\}\}$ alors $\langle M \rangle_{\{n \mapsto k\}}$ est trivialement satisfaisable. Si M est $\{\bot\}$ ou un c-état alors M est insatisfaisable.

Supposons maintenant que M est un état propositionnel. Si c'est un $\wedge$-état, alors il existe un seul M' t.q. (M, ϵ, M'). M' ne diffère de M que par le fait qu'une conjonction a été décomposée, donc comme les états sont interprétés comme la conjonction de leurs éléments, la satisfaisabilité de M' est équivalente à celle de M. Si M est un $\vee$-état, alors nous avons le choix entre deux transitions, correspondant chacune à une des possibilités exprimées par la disjonction. Donc l'une au moins mène à un état satisfaisable si M est satisfaisable, et réciproquement. Si M est un p-état, le résultat est une conséquence du lemme 5.28.

Enfin supposons que M est un d-état. Si $k = q_1 + q_2 - 1$ alors le résultat est une conséquence directe de la définition de *base* (définition 6.14). Supposons que $k > q_1 + q_2 - 1$. Par définition de *induction* (définition 6.14), M^{s} est égal à M dont les itérations ont été déroulées une fois et dans lequel $n + 1$ a été substitué à n. Donc $\langle M^{\mathrm{s}} \rangle_{\{n \mapsto n-1\}}$ n'est que M dont les itérations ont été déroulées une fois. Ainsi, comme $k > q_1 + q_2 - 1$, $\langle M^{\mathrm{s}} \rangle_{\{n \mapsto k-1\}}$ et $\langle M \rangle_{\{n \mapsto k\}}$ sont égaux. D'où le résultat. □

Lemma 6.37. *Soient un schéma régulier s, un état M de a_s.*

(i) Si M est satisfaisable alors il existe une course de a_s partant de M, constituée uniquement d'ϵ-transitions et arrivant dans l'état final ou dans un d-état.

(ii) Si, de plus, M ne contient pas d'occurrence de n alors il existe une course de a_s partant de M, constituée uniquement d'ϵ-transitions et arrivant dans l'état final.

(iii) Tout d-état ne contient que des itérations ou des littéraux.

Preuve. (iii) est triviale car, par définition de SchAut, nous n'appliquons descente que si aucune autre règle ne s'applique et il est facile de voir que si nous avons autre chose que des littéraux ou des itérations, alors une autre règle peut s'appliquer.

(i) Nous construisons la course récursivement.

Si aucune ϵ-transition ne part de M, alors soit c'est un d-état, soit aucune transition ne part de M. Dans le deuxième cas, nous avons nécessairement $M \in \{\{\bot\}, \emptyset\}$, mais comme M est satisfaisable nous ne pouvons pas avoir $\{\bot\}$, et M ne peut être que l'état final.

Si une ϵ-transition part de M, alors M ne peut être qu'un état propositionnel, donc d'après le lemme précédent, il existe un M' t.q. M' est satisfaisable et a_s contient (M, ϵ, M'). Nous définissons M' comme l'état suivant dans la course.

Enfin, cette construction termine car le nombre de symboles dans un état diminue strictement à chaque ϵ-transition.

(ii) Nous reprenons la construction de (i) et montrons que dans le cas où aucune ϵ-transition ne part de M, M ne peut être que $\emptyset$. En effet comme aucune ϵ-transition ne part de M, M ne peut contenir que des itérations et des littéraux non purs. Comme M ne contient aucune occurrence de n, il ne peut pas contenir d'itération. Comme M est satisfaisable, il ne peut pas contenir deux littéraux complémentaires, donc tout littéral apparaissant dans M est pur. M ne contient donc aucun littéral, c'est donc soit $\{\top\}$, soit $\emptyset$. Dans le premier cas il existe une ϵ-transition partant de M, ce qui n'est pas possible, M ne peut donc être que $\emptyset$, c.-à-d. l'état final. □

Lemma 6.38. *Soient un schéma régulier s de contrainte $n - q_2 \geq q_1 - 1$ et $k \in \mathbb{N}$. Alors, pour tout état M de a_s, il existe une course de a_s partant de M et acceptant k ssi $\langle M \rangle_{\{n \mapsto k+q_2+q_1-1\}}$ est satisfaisable.*

Preuve. Nous démontrons la première implication par récurrence sur la longueur de la course. Si cette longueur est nulle, alors $M = \emptyset$ et est trivialement satisfaisable (quel que soit la valeur de k). Si cette longueur est strictement positive, alors M a un successeur dans la course. Nous appliquons alors le lemme 6.36 : si M est un état propositionnel, alors $\langle M \rangle_{\{n \mapsto k+q_2+q_1-1\}}$ est satisfaisable ssi $\langle M' \rangle_{\{n \mapsto k+q_2+q_1-1\}}$ est satisfaisable et nous concluons avec l'hypothèse de récurrence. Nous concluons de façon similaire dans les autres cas possibles pour M.

Prouvons maintenant la réciproque : nous supposons $\langle M \rangle_{\{n \mapsto k+q_2+q_1-1\}}$ satisfaisable et nous construisons une course partant de M et acceptant k, par récurrence sur k. D'après le lemme 6.37 (i), il existe une course constituée uniquement d'ϵ-transitions et arrivant dans l'état final ou un d-état. Si la course arrive dans l'état final, le résultat est prouvé (car nous avons les transitions $(\emptyset, 0, \emptyset)$ et $(\emptyset, \mathrm{s}, \emptyset)$). Si la course arrive dans un d-état M', alors nous raisonnons par récurrence sur k. Supposons $k = 0$. Dans ce cas nous ajoutons à la course M'^0. Comme M est satisfaisable, M'^0 l'est aussi (d'après 6.36 et par récurrence sur la longueur de la course). De plus M'^0 ne contient pas d'occurrence de n (par définition de *base*), nous pouvons donc appliquer le lemme 6.37 (ii) ce qui fournit une course jusqu'à l'état final depuis M'^0. Nous avons donc une course depuis M jusqu'à l'état final ne contenant que des ϵ-transitions et une transition étiquetée par 0. Ainsi 0 est accepté à partir de M. Supposons maintenant $k > 0$. Nous ajoutons alors M'^{s} à la course. Comme $\langle M \rangle_{\{n \mapsto k+q_2+q_1-1\}}$ est satisfaisable, $\langle M'^{\mathrm{s}} \rangle_{\{n \mapsto k+q_2+q_1-2\}}$ l'est aussi. Nous pouvons donc construire une course partant de M'^{s} jusqu'à un d-état M'' ou un état final. Si nous arrivons à un état final nous pouvons conclure. Sinon nous savons que $\langle M'' \rangle_{\{n \mapsto k+q_2+q_1-2\}}$ est satisfaisable. Nous pouvons donc appliquer l'hypothèse de récurrence à M''. Comme entre M et M'' nous n'avons que des ϵ-transitions et une transition étiquetée par s, $k-1$ est accepté par cette course. □

CHAPTER 7

Implémentation : RegStab

Ce chapitre est consacré à la présentation de RegStab : il s'agit de l'implémentation de la stratégie de Stab dédiée aux schémas réguliers (stratégie appelée τ en 6.2). Le contenu de ce chapitre a été présenté dans [ACP10e]. On pourra trouver des détails d'ordre plus pratique dans le manuel de RegStab [Ara10]. RegStab est disponible à l'adresse `http://regstab.forge.ocamlcore.org`. En section 7.1, nous présentons le système du point de vue utilisateur : syntaxe, exemples, performances. En section 7.2, nous donnons plus de détails sur l'implémentation.

7.1 Le système

RegStab prend en entrée la description d'un schéma régulier et calcule, de façon complètement automatisée, la satisfaisabilité du schéma, c.-à-d. il affiche UNSATISFIABLE ssi le schéma est insatisfaisable. La procédure sous-jacente à RegStab est Stab avec la stratégie présentée en 6.2. D'après le théorème 6.10 nous avons donc la garantie que Stab terminera *quelle que soit son entrée*. De plus, bien que la procédure ne soit pas basée sur SchAut, il est facile d'adapter la preuve du théorème 6.33 pour démontrer que RegStab a aussi une complexité en double exponentielle. En fait RegStab est "historiquement" basé sur Stab (Stab a été développé avant SchAut), d'où son nom, mais l'implémentation est en fait plus proche de SchAut.

RegStab s'utilise simplement en ligne de commande, soit en entrant un schéma sur `stdin` soit en lisant un fichier d'extension `*.stab`. La grammaire est donnée en figure 7.1.[1] L'exemple récurrent $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$ s'écrit p.ex. :

```
P_0 /\ /\i=0..n-1 (P_i -> P_i+1) /\ ~P_n | n>=0
```

Quand RegStab a fini de s'exécuter, il affiche le statut du schéma, c.-à-d. SATISFIABLE ou UNSATISFIABLE, comme nous pouvons le voir dans les figures 7.2 et 7.3. La syntaxe permet aussi de définir des fonctions de façon à simplifier l'écriture de "longs" schémas : ce qui permet, p.ex. d'écrire l'additionneur de l'exemple 6.4 de façon lisible, c.f. figure 7.5. Par ailleurs, des options permettent d'afficher de plus amples détails sur l'exécution : nombre de cycles détectés, nombre de règles appliquées, profondeur maximale de dépliage des itérations, etc. La figure 7.6 montre le résultat avec l'additionneur. De nombreux exemples sont fournis avec le logiciel. On retrouvera en particulier ceux de la section 6.1. La figure 7.7 présente les temps d'exécution de RegStab sur ces benchmarks.

[1] Pour des raisons d'efficacité et de simplicité d'implémentation, les littéraux ne peuvent pas prendre pour indice des expressions de la forme $qn + r$ où $q \neq 1$. Les schémas en entrée de RegStab sont donc en fait une sous-classe (légèrement) stricte des schémas réguliers. De plus la contrainte est restreinte à un atome et doit entraîner $n \geq q_1 + q_2 - 1$ comme pour SchAut (chapitre 6.3).

sentence ::= *schema*
| *schema* `|` *constraint*
schema ::= *indexed-prop* `_` *linear-expression*
| `~` *indexed-prop* `_` *linear-expression*
| *schema* `/\` *schema*
| *schema* `\/` *schema*
| *schema* `->` *schema*
| *schema* `(+)` *schema*
| *schema* `<->` *schema*
| `(` *schema* `)`
| `/\` *var* `=` *integer* `..` *linear-expression* *no-iteration*
| `\/` *var* `=` *integer* `..` *linear-expression* *no-iteration*
constraint ::= *var* `<=` *integer*
| *var* `>=` *integer*
| *var* `<` *integer*
| *var* `>` *integer*
| *var* `=` *integer*
linear-expression ::= *var*
| *integer*
| *var* `+` *integer*
| *var* `-` *integer*
var ::= `a...z` $\{$`a...z|0...9|'`$\}^*$
indexed-prop ::= `A...Z` $\{$`A...Z|a...z|0...9|'`$\}^*$
integer ::= $\{$`0...9`$\}^+$

Figure 7.1: Syntaxe de RegStab.

7.2 L'implémentation

RegStab applique la stratégie donnée en 6.2, c.-à-d. :

1. nous appliquons toutes les règles sauf les règles d'itération (c.-à-d. les règles propositionnelles, la règle de contrainte, la règle de contradiction et la règle de bouclage avec ses extensions) tant que nous le pouvons;

2. puis nous appliquons les règles d'itérations, uniquement sur les itérations de longueur maximale.

Quelques restrictions supplémentaires sont appliquées pour des raisons d'efficacité. Intuitivement, l'opération la plus coûteuse est la détection de cycle : nous devons stocker les étiquettes de tous les nœuds rencontrés pour pouvoir ultérieurement détecter un cycle. Heureusement, d'après la preuve du théorème 6.2, nous savons que nous pouvons nous restreindre aux nœuds d'alignement. Par conséquent le stockage d'un nœud est effectué juste avant d'appliquer les règles d'itérations (c.-à-d. juste avant l'étape 2). Par suite la recherche d'un nœud dans la base peut aussi n'être effectuée qu'aux nœuds d'alignement. Ainsi la règle de bouclage sera appliquée une fois seulement que les règles propositionnelles, de contrainte et de contradiction auront été appliquées.

De plus, si nous appliquons les extensions de l'égalité à un décalage près à la lettre, nous sommes censés garder les littéraux purs apparaissant dans les nœuds et ne les supprimer que lorsque nous appliquons la règle de bouclage. Un inconvénient évident est que si un nœud a des littéraux purs, ceux-ci

```
~aravantinos> regstab
Taking input from stdin.
P_0 /\ /\i=0..n-1 (P_i -> P_i+1) /\ ~P_n | n>=0
UNSATISFIABLE
~aravantinos>
```

Figure 7.2: Exemple récurrent, donné via `stdin`.

```
~aravantinos> cat exemple.stab
P_0 /\ /\i=0..n-1 (P_i -> P_i+1) /\ ~P_n | n>=0
~aravantinos> regstab exemple.stab
UNSATISFIABLE
~aravantinos>
```

Figure 7.3: Exemple récurrent, donné via le fichier `exemple.stab`.

seront conservés et propagés dans tous les fils, petit-fils, etc. de ce nœud. Cela ne pose pas de problème pour la satisfaisabilité ni pour la terminaison, mais cela induit des calculs redondants lorsque nous voudrons détecter les littéraux purs des fils de ce nœud. Ainsi nous pouvons en fait supprimer directement les littéraux purs (ceci correspond à l'utilisation habituelle des littéraux purs : nous les éliminons pour diminuer l'espace de recherche). Évidemment nous pouvons en faire autant avec les contraintes redondantes. Nous introduisons donc dans la stratégie une étape où nous éliminons les littéraux purs d'un nœud. En revanche nous savons que, pour la terminaison, les littéraux purs n'ont besoin d'être éliminés qu'au moment de la règle de bouclage. Par conséquent plutôt que d'éliminer les littéraux purs après chaque application d'une règle (ce qui peut être très coûteux car il faut à chaque fois parcourir tous les littéraux du nœud), nous les éliminons juste avant la recherche de cycle.

Enfin, nous remarquons facilement que la règle de contrainte n'est appliquée qu'une fois : au tout début de la procédure, il est donc inutile de l'insérer dans la boucle.

Au final nous obtenons la nouvelle stratégie suivante :

1. nous appliquons la règle de contrainte une fois;
2. nous appliquons les règles propositionnelles et de contradiction jusqu'à saturation;
3. nous éliminons les littéraux purs;
4. nous appliquons la règle de bouclage (c.-à-d. nous recherchons le nœud dans la base et nous fermons la branche le cas échéant);
5. nous stockons le nœud dans la base;
6. nous appliquons les règles d'itérations;
7. et nous retournons en 2.

Nous nous intéressons maintenant plus particulièrement à certains sous problèmes qui sont cruciaux pour l'efficacité.

7.2.1 Règle de contradiction

La détection d'une contradiction est plus complexe que dans le cas propositionnel : il faut tenir compte des contraintes du nœud. Ainsi par exemple $\neg p_n \wedge p_1$ n'est contradictoire que si $n = 1$. Nous devons donc tenir compte de l'*arithmétique*. Ceci nous empêche donc d'utiliser des structures classiques comme des tables de hachage ou des arbres binaires qui auraient permis une recherche en temps constant. Cependant nous pouvons quand même améliorer la situation : la satisfaisabilité en arithmétique linéaire, bien que décidable, n'en est pas moins complexe (elle est dans NP si nous n'avons aucun quantificateur, et dans 2-EXPTIME si nous en avons [FR74]). Nous allons donc effectuer des restrictions sur la façon d'ajouter des contraintes aux nœuds, afin de simplifier le raisonnement arithmétique.

La principale restriction est que nous n'appliquons pas la règle de contradiction telle qu'elle est présentée dans Stab : en effet, nous calculons la contrainte générée (trivial), puis nous vérifions qu'elle

sentence ::= ...
| `let` *definition* `:=` *schema* `in` *sentence*
schema ::= ...
| *function-call*
definition ::= *indexed-prop* `(` *parameters* `)`
| *indexed-prop*
parameters ::= *indexed-prop*
| *var*
| *indexed-prop*`,` *parameters*
| *var*`,` *parameters*
function-call ::= *indexed-prop* `(` *arguments* `)`
| *indexed-prop* `()`
arguments ::= *linear-expression*
| *indexed-prop*
| *indexed-prop* `,` *arguments*
| *linear-expression* `,` *arguments*

Figure 7.4: Syntaxe pour l'utilisation de définitions dans RegStab.

```
~aravantinos> cat adder.stab
// Conjecture : 0 est element neutre de l'additionneur

let Sum(i)     := S_i <-> (A_i (+) B_i (+) C_i) in
let Carry(i)   := C_i+1 <->
  (A_i /\ B_i \/ C_i /\ A_i \/ C_i /\ B_i) in
let Adder      := /\i=1..n (Sum(i) /\ Carry(i)) /\ ~C_1 in
let NullB      := /\i=1..n ~B_i in
let Conclusion := \/i=1..n (A_i (+) S_i) in

Adder() /\ NullB() /\ Conclusion()
~aravantinos>
```

Figure 7.5: Exemple de l'additionneur.

n'est pas contradictoire. Si elle l'est nous fermons la branche, sinon nous continuons, mais *nous n'ajoutons pas la contrainte au nœud*. Ceci a pour inconvénient de nous faire "rater" des inférences potentiellement utiles. Mais cela a pour avantage que chaque nœud ne contient qu'une contrainte et cette contrainte a l'une des formes suivantes : $\top$, $\bot$, $n = q$, $n \geq q$, $n \leq q$. En effet, supposons que le schéma est aligné sur $[q_1; n - q_2]$, dans ce cas :

- la contrainte initiale est soit $\top$, soit $n = q$, soit $n \geq q$ où $q \geq q_1 + q_2 - 1$;
- comme nous n'ajoutons pas les contraintes générées par la règle de contradiction, les seules contraintes ajoutées sont de la forme $n + q_1 - q \geq q_2$ où $n + q_1 - q \leq q_2$.

Il est facile de voir que dans ces conditions nous ne pouvons obtenir que des contraintes de la forme ci-dessus. Ceci permet de simplifier de façon drastique le traitement des contraintes, en particulier pour détecter les contradictions et pour détecter les littéraux purs. Il s'agit donc d'un mal pour un bien mais le bien acquis est beaucoup plus systématique : ne pas ajouter ces contraintes permet de gagner un ordre de grandeur sur la complexité de l'arithmétique linéaire, alors que les garder ne garantit aucun gain (c.-à-d. que nous *pouvons* avoir un gain mais celui-ci n'est absolument pas assuré). Notons enfin que ces contraintes ne sont pas nécessaires à la terminaison : seules les contraintes introduites par les règles d'itérations le sont vraiment (par analyse de la preuve du théorème 6.33).

```
~aravantinos > regstab --verbose adder.stab

Conjecture:
  ((((/\i=1..n ((S_i <-> ((A_i (+) B_i) (+) C_i)) /\ (C_i+1 <->
  (((A_i /\ B_i) \/ (C_i /\ A_i)) \/ (C_i /\ B_i))))) /\ ~C_1)
  /\ (/\i=1..n ~B_i)) /\ (\/i=1..n (A_i (+) S_i))


Applications of tableau rules:
/\:  67
\/:  84
(+): 38
<->: 32
->:  0
Iterated /\: 12
Iterated \/:  3
-------
Total propositional rules: 221
Total iterated rules:      15

Number of closed leaves:  137
Number of looping leaves: 30
Number of lemmas:         4

Maximum number of unfoldings: 3
(if you are surprised by this number, notice that the tableau
is constructed depth-first)

UNSATISFIABLE

~aravantinos >
```

Figure 7.6: Option `--verbose` avec l'additionneur.

Une fois cette simplification effectuée, déterminer si la règle de contradiction génère la contrainte $\bot$ devient trivial : cette contrainte a toujours la forme $e \neq f$. Si e et f sont des entiers, le calcul est évident de toute manière. Si e et f ont la forme $n + q_1$ et $n + q_2$ alors il suffit de vérifier que $q_1 = q_2$, ce qui est immédiat. Enfin si e a la forme $n + q$ et f est un entier, alors il est facile de voir que parmi toutes les formes possibles pour la contrainte du nœud, seule $n = q'$ peut générer une contradiction. Dans ce cas il suffit de vérifier que $q' + q = f$. Nous obtenons ainsi une vérification en temps constant[2], à comparer au temps qu'aurait mis l'utilisation d'une procédure pour l'arithmétique linéaire.

7.2.2 Élimination des littéraux purs

Tout comme pour la détection de contradictions, la détection d'un littéral pur est plus complexe que dans le cas propositionnel : il faut tenir compte des contraintes du nœud. Par exemple $\neg p_2$ n'est pas pur en général dans $\bigwedge_{i=1}^{n} p_i$. En revanche, il l'est si $n < 2$. Heureusement, dans le cas très particulier des schémas réguliers et en tenant compte de la restriction opérée dans la section précédente, nous pouvons simplifier ce calcul.

D'après la définition 5.26, un littéral l est pur dans un schéma s ssi pour tout environnement ρ, $\langle l \rangle_\rho$ est pur au sens propositionnel dans $\langle s \rangle_\rho$. Nous pouvons ramener cette définition au niveau des littéraux : un littéral l est pur dans un schéma s ssi pour tout littéral l' apparaissant dans s de signe opposé à l et pour tout environnement ρ : $\langle l \rangle_\rho \neq \langle l' \rangle_\rho$ (ou $\langle l \rangle_\rho \neq \langle l' \rangle_{\rho \circledcirc \{i \mapsto I\}}$, pour un certain $I \in \mathbb{Z}$, si l' contient

[2]en théorie cette vérification est en fait en temps logarithmique par rapport aux entiers concernés; en pratique, comme les entiers machine sont bornés, les processeurs effectuent ces calculs en temps constant

Additionneur à propagation de retenue	
$x + 0 = x$	$0.002s$
commutativité	$0.016s$
associativité	$2.219s$
$3 + 4 = 7$	$0.197s$
Additionneur à retenue anticipée	
$x + 0 = x$	$0.002s$
commutativité	$0.008s$
associativité	$0.375s$
équivalence entre deux différentes définitions du même circuit	$0.008s$
équivalence avec l'additionneur à propagation de retenue	$0.011s$
Comparaison entre vecteurs de bits	
$x \leq 0$	$0.001s$
symétrie de $\leq$ (c.-à-d. $x \leq y \wedge x \geq y \Rightarrow x = y$)	$0.001s$
totalité de $\leq$ (i.e. $x > y \vee x \leq y$)	$0.001s$
transitivité de $\leq$	$0.001s$
$1 \leq 2$	$0.002s$
Arithmétique de Presburger entre vecteurs de bits	
$x + y \geq x$	$0.002s$
$x_1 \leq x_2 \leq x_3 \Rightarrow x_1 + y \leq x_2 + y \leq x_3 + y$	$3.651s$
$x_1 \leq x_2 \wedge y_1 \leq y_2 \Rightarrow x_1 + y_1 \leq x_2 + y_2$	$0.057s$
$x_1 \leq x_2 \leq x_3 \wedge y_1 \leq y_2 \leq y_3 \Rightarrow x_1 + y_1 \leq x_2 + y_2 \leq x_3 + y_3$	$55.190s$
$1 \leq x + y \leq 5 \wedge x \geq 3 \wedge y \geq 4$	$12.733s$
idem mais avec les itérations factorisées	$8.696s$
Autre	
inclusion d'automates	$0.020s$
$\bigvee_{i=1}^{n} p_i \wedge \bigwedge_{i=1}^{n} \neg p_i$	$0.001s$
$p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$	$0.001s$

Figure 7.7: Temps d'exécution de RegStab sur quelques exemples.

une variable liée i). Soient, donc, deux littéraux p_e et $\neg p_f$. Nous distinguons trois cas :

- e et f sont des entiers. Dans ce cas il est facile de voir que $\forall \rho \cdot \langle l \rangle_\rho \neq \langle l' \rangle_\rho$ est équivalent à $e \neq f$, donc dans ce cas le calcul est immédiat.

- e et f sont tous deux de la forme $n + q_1$ et $n + q_2$. Dans ce cas, il suffit de vérifier que $q_1 \neq q_2$, ce qui est tout aussi immédiat.

- e est de la forme $n + q$ et f est un entier. Dans ce cas nous utilisons de façon cruciale le fait que les contraintes ont été restreintes, en effet la seule façon d'avoir $n + q \neq f$ est d'avoir soit $n + q < f$, soit $n + q > f$, ce qui est trivial à détecter vu la forme des contraintes évoquée plus haut.

- e est un entier et f est de la forme $i + q$. Dans ce cas la seule façon d'avoir $e \neq f$, est d'avoir $I + q < f$ ou $I + q > f$ pour tout I compris entre les deux bornes de l'itération q_1 et $n - q_2$. Le seul moyen d'avoir $I + q > f$ quel que soit I est d'avoir $q_1 + q > f$, ce qui est facile à vérifier. Le seul moyen d'avoir $I + q < f$ pour tout I est que la contrainte entraîne $n - k_2 < f$, ce qui se vérifie très simplement en utilisant le même type de raisonnement vu jusque là. En fin de compte, comparer des expressions avec variables liées revient à faire la comparaison aux bornes de l'itération liante.

- le cas où e est de la forme $n + q_2$ et f est de la forme $i + q_1$ est traité de façon similaire.

Dans tous les cas il devient très simple de déterminer si deux littéraux peuvent être égaux ou pas. Notons au final qu'il faut encore effectuer ce calcul pour tous les paires de littéraux d'un nœud, ce qui reste très coûteux.

7.2.3 Détection de cycles

La tâche la plus lourde est évidemment la détection de cycle. Comme les littéraux purs sont directement éliminés et que nous avons restreint les contraintes à une forme très simple, en particulier nous avons automatiquement supprimé les contraintes redondantes, nous n'avons plus qu'à gérer l'égalité à un décalage près. Le fait que nous n'ayons pas une égalité stricte empêche d'utiliser des structures de données classiques pour la base de nœuds déjà vus. Mais nous pouvons définir une forme normale sur les nœuds pour laquelle l'égalité à un décalage près revient à l'égalité. À cette fin, nous définissons le *décalage absolu* :

Definition 7.1. *Soit une expression e de la forme $n+q$ où $q \in \mathbb{Z}$ et n est une variable. Nous appelons q le* décalage absolu *de e, noté $\delta(e)$. Si e est un entier alors $\delta(e)$ n'est pas défini.*

Le décalage absolu *d'une contrainte c, noté $\delta(c)$, est le décalage absolu de la première expression arithmétique rencontrée en lisant c de gauche à droite. Il est indéfini si c ne contient pas d'expression arithmétique.*

Le décalage absolu *d'un schéma s, noté $\delta(s)$, est défini par :*

$$\delta(p_e) \stackrel{def}{=} \delta(\neg p_e) \stackrel{def}{=} \delta(e)$$

$$\delta(m_1 \amalg m_2) \stackrel{def}{=} \begin{cases} \delta(m_1) & \text{si } \delta(m_1) \text{ est défini} \\ \delta(m_2) & \text{sinon} \end{cases}$$

$$\delta(m \,\&\, c) \stackrel{def}{=} \begin{cases} \delta(m) & \text{si } \delta(m) \text{ est défini} \\ \delta(c) & \text{sinon} \end{cases}$$

$$\delta(\Pi_{i=e}^{f} m) \stackrel{def}{=} \delta(f)$$

où $\amalg \in \{\wedge, \vee\}$ et $\Pi \in \{\bigwedge, \bigvee\}$.

Par exemple, $\delta(p_n \wedge \bigwedge_{i=1}^{n+1} p_i) = 0$ et $\delta(p_{n-1} \wedge \bigwedge_{i=1}^{n} p_i) = -1$. Intuitivement, le décalage absolu d'un schéma correspond simplement au décalage de la première expression contenant n rencontrée dans le schéma. La notion de "première" est complètement arbitraire, elle permet juste d'avoir une expression de référence pour définir le *représentant* d'un schéma :

Definition 7.2. *Pour un schéma s, nous appelons $s[n - \delta(s)/n]$ le* représentant *de s.*

Par exemple le représentant de $p_n \wedge \bigwedge_{i=1}^{n+1} p_i$ est lui-même, celui de $p_{n-1} \wedge \bigwedge_{i=1}^{n} p_i$ est $p_n \wedge \bigwedge_{i=1}^{n+1} p_i$. Intuitivement, le représentant de s est s décalé de sorte à ce que la première expression contenant n soit simplement n, sans décalage. Les définitions de décalage absolu et de représentant s'étendent naturellement aux ensembles de motifs. Il est alors facile de voir que si deux schémas sont égaux à un décalage près, alors leurs représentants sont égaux. Nous avons donc bien la forme normale voulue : au lieu de stocker un schéma dans la base, nous allons stocker son représentant. Puis quand nous testerons l'appartenance d'un schéma à la base (à un décalage près), nous testerons en fait l'appartenance de son représentant. Il reste un problème qui est que si les représentants de deux schémas sont égaux alors nous n'avons pas nécessairement l'égalité à un décalage près de ces schémas : nous avons en fait l'implication inverse de celle que nous voulions. Par exemple $p_n \wedge \bigwedge_{i=1}^{n+1} p_i \not\rightrightarrows_n p_{n-1} \wedge \bigwedge_{i=1}^{n} p_i$, pourtant leurs représentants valent tous deux $p_n \wedge \bigwedge_{i=1}^{n+1} p_i$.

De façon plus concrète cela signifie que nous pourrions avoir la situation suivante. Nous rencontrons $p_{n-1} \wedge \bigwedge_{i=1}^{n} p_i$ dans une branche de l'arbre. Nous l'ajoutons à la base. Par ailleurs, *dans une branche parallèle*, nous rencontrons $p_n \wedge \bigwedge_{i=1}^{n+1} p_i$. Comme les représentants de $p_n \wedge \bigwedge_{i=1}^{n+1} p_i$ et $p_{n-1} \wedge \bigwedge_{i=1}^{n} p_i$ sont les mêmes, nous ne développons pas la branche de $p_n \wedge \bigwedge_{i=1}^{n+1} p_i$. Pourtant, $p_n \wedge \bigwedge_{i=1}^{n+1} p_i$ *ne boucle pas sur* $p_{n-1} \wedge \bigwedge_{i=1}^{n} p_i$. Dans ce cas la méthode ne serait pas correcte. En fait nous avons le résultat suivant (trivial) : $s_1 \rightrightarrows_n s_2$ ssi le représentant de s_1 est égal au représentant de s_2 *et* $\delta(s_1) < \delta(s_2)$. Il est alors très facile d'effectuer la recherche de représentant puis de vérifier que le décalage est inférieur.

Nous pouvons enfin utiliser une structure classique pour stocker les représentants : table de hachage, arbre binaire, ou comme c'est le cas dans RegStab des "tries" [Fre60]. Pour pouvoir utiliser cette structure nous devons représenter les nœuds par des listes et non pas des ensembles. Ceci se fait très simplement en ordonnant les ensembles (avec un ordre arbitraire). Au final nous avons donc une structure de données pour laquelle la recherche est en temps constant par rapport à la taille de l'ensemble et linéaire par rapport à la taille du nœud recherché.

CHAPTER 8

DPLL*

Nous définissons maintenant un autre système de preuve, appelé DPLL*. Comme son nom l'indique il s'appuie sur DPLL (2.3.2, [DLL62]), au contraire de STAB qui était, lui, basé sur les tableaux propositionnels. Nous avons défini DPLL* (pour la première fois dans [ACP09a]) dans le but de pallier la difficulté qu'a STAB à gérer les *itérations imbriquées*. L'absence d'itérations imbriquées est justement une des contraintes les plus importantes dans la définition des schémas réguliers. Dans ce chapitre nous commençons par préciser cette assertion : nous donnons en 8.1 un exemple contenant des itérations imbriqués et ne terminant pas avec STAB. Nous définissons ensuite DPLL* en 8.2, comme nous le verrons le système est beaucoup moins intuitif que STAB, mais il permet de prouver l'exemple de 8.1 comme nous le verrons en 8.3. Nous prouvons enfin la correction et la complétude pour la satisfaisabilité de DPLL* en 8.4 et 8.5. Le contenu de ce chapitre provient de [ACP09a], et surtout dans sa forme actuelle de [ACP10a, ACP10b].

8.1 Motivating Example

Soit le schéma (trivialement insatisfaisable) :

$$s \stackrel{\text{def}}{=} \left(\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} p_i \Rightarrow q_j \right) \wedge \bigwedge_{i=1}^{n} \neg q_i \wedge \bigvee_{i=1}^{n} p_i \,\&\, n \geq 1$$

Nous montrons que STAB ne termine pas sur cet exemple. On applique toutes les règles propositionnelles et la règle de contrainte :

$$\left\{ \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} p_i \Rightarrow q_j, \bigwedge_{i=1}^{n} \neg q_i, \bigvee_{i=1}^{n} p_i, n \geq 1 \right\}$$

Puis nous appliquons toutes les règles d'itération :

$$\left\{ \left(\bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n} p_i \Rightarrow q_j \right) \wedge \bigvee_{j=1}^{n} p_n \Rightarrow q_j, \left(\bigwedge_{i=1}^{n-1} \neg q_i \right) \wedge \neg q_n, \left(\bigvee_{i=1}^{n-1} p_i \right) \vee p_n, n \geq 1 \right\}$$

Puis à nouveau les règles propositionnelles ($\wedge$ uniquement):

$$\left\{ \bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n} p_i \Rightarrow q_j, \bigvee_{j=1}^{n} p_n \Rightarrow q_j, \bigwedge_{i=1}^{n-1} \neg q_i, \neg q_n, \left(\bigvee_{i=1}^{n-1} p_i \right) \vee p_n, n \geq 1 \right\}$$

Nous pouvons à nouveau appliquer ITERATED $\vee$:

$$\left\{ \bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n} p_i \Rightarrow q_j, \bigvee_{j=1}^{n-1} p_n \Rightarrow q_j \vee p_n \Rightarrow q_n, \bigwedge_{i=1}^{n-1} \neg q_i, \neg q_n, \bigvee_{i=1}^{n-1} p_i \vee p_n, n \geq 1 \right\}$$

Deux applications de $\vee$ donnent lieu à quatre branches différentes. Nous nous focalisons sur celle contenant $p_n \Rightarrow q_n$ et $\bigvee_{i=1}^{n-1} p_i$:

$$\left\{ \bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n} p_i \Rightarrow q_j, p_n \Rightarrow q_n, \bigwedge_{i=1}^{n-1} \neg q_i, \neg q_n, \bigvee_{i=1}^{n-1} p_i, n \geq 1 \right\}$$

Une nouvelle application de $\vee$ permet de décomposer l'implication $p_n \Rightarrow q_n$. Nous choisissons la branche contenant $\neg p_n$:

$$\left\{ \bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n} p_i \Rightarrow q_j, \neg p_n, \bigwedge_{i=1}^{n-1} \neg q_i, \neg q_n, \bigvee_{i=1}^{n-1} p_i, n \geq 1 \right\}$$

Comme d'habitude nous espérons trouver un cycle, nous pouvons donc utiliser l'extension du littéral pur. Comme $\neg p_n$ est pur dans cet ensemble nous pouvons alors l'éliminer :

$$\left\{ \bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n} p_i \Rightarrow q_j, \bigwedge_{i=1}^{n-1} \neg q_i, \neg q_n, \bigvee_{i=1}^{n-1} p_i, n \geq 1 \right\}$$

Le schéma que nous obtenons alors est *presque* $s[n-1/n]$. Les seuls éléments qui empêchent cet égalité sont l'itération $\bigvee_{j=1}^{n} p_i \Rightarrow q_j$ dont la borne supérieure est n et pas $n-1$, et le littéral $\neg q_n$ qui n'est pas pur car il apparaît implicitement dans l'itération $\bigvee_{j=1}^{n} p_i \Rightarrow q_j$. Il est facile de voir que si nous continuons à appliquer les règles, toutes les itérations *sauf* $\bigvee_{j=1}^{n} p_i \Rightarrow q_j$ seront dépliées indéfiniment. De plus les littéraux $\neg q_n$, $\neg q_{n-1}$, etc. ne seront jamais purs car l'itération $\bigvee_{j=1}^{n} p_i \Rightarrow q_j$ sera toujours présente. Ainsi aucun cycle ne sera jamais détecté dans cette branche (du moins en utilisant l'égalité à un décalage près avec ses extensions[1]).

Le problème provient du fait que $\bigvee_{j=1}^{n} p_n \Rightarrow q_j$ ne se verra *jamais* appliquer une règle d'itération car ces dernières ne s'appliquent que si la *racine* du schéma est une itération, mais pas sur des itérations situées *en profondeur* du schéma. Ainsi toutes les occurrences de n seront décalées, sauf celles situées en profondeur, ce qui nous empêche de conclure. L'idée derrière DPLL* est donc d'avoir un système capable de modifier un schéma *à une profondeur arbitraire*.

8.2 System Presentation

Tout comme STAB, DPLL* est définie à l'aide de tableaux, bien que ceux-ci n'aient plus de rapport avec les tableaux sémantiques. Cette fois, chaque nœud est étiqueté non par des ensembles mais par la paire d'un schéma et d'un ensemble de littéraux (comparer avec DPLL 2.3.2, où les nœuds sont étiquetés par la paire d'un formule et d'un ensemble de littéraux). Nous commençons par donner l'intuition générale de DPLL* en faisant l'analogie entre les tableaux propositionnels et STAB, et entre DPLL et DPLL*.

À chaque branchement, les tableaux propositionnels séparent l'ensemble de toutes les interprétations selon la structure de la formule : par exemple si nous avons la formule $\phi_1 \vee \phi_2$, la branche de gauche permettra de considérer les modèles de ϕ_1 et la branche de droite les modèles de ϕ_2 (notons que ces deux ensembles ne sont pas disjoints en général). Dans chaque branche nous pouvons effectuer les simplifications correspondantes : dans la branche de gauche, nous supprimons ϕ_2 car nous ne nous intéressons qu'aux modèles qui valideront ϕ_1, et réciproquement dans la branche droite. Le tableau ainsi construit permet donc de considérer les différentes interprétations. L'idée étant que lorsque nous avons

[1] En effet, le nœud contient $\neg q_n$, et il est facile de voir que $\neg q_n \wedge \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} p_i \Rightarrow q_j \equiv \neg q_n \wedge \bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n-1} p_i \Rightarrow q_j$ (le dernier rang de l'itération imbriquée est "inutile"). Et $\neg q_n$ est pur dans $\bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n-1} p_i \Rightarrow q_j$, donc tout modèle de $\neg q_n \wedge \bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n-1} p_i \Rightarrow q_j$ est un modèle de $\bigwedge_{i=1}^{n-1} \bigvee_{j=1}^{n-1} p_i \Rightarrow q_j$. Donc tout modèle du dernier nœud obtenu est un modèle de $s[n-1/n]$, donc il boucle bien sur le nœud original par rapport à l'ordre standard.

suffisamment restreint l'ensemble d'interprétations dans une branche la formule en devient suffisamment simplifiée pour que nous puissions conclure de façon triviale si elle est satisfaisable ou non.

Le but de DPLL est aussi de séparer l'ensemble des interprétations mais pas en se basant sur la structure de la formule : DPLL distingue les interprétations selon la valeur qu'elles donnent à une variable propositionnelle. Ainsi le branchement permet de distinguer toutes les interprétations qui assignent la valeur false à une variable, et celles qui lui assignent la valeur true. Tout comme pour les tableaux, cette restriction dans chaque branche permet de simplifier la formule considérée.

Tout comme les tableaux propositionnels, STAB permet de séparer les interprétations suivant la structure de la formule. Cependant, en plus de donner une valeur aux variables propositionnelles, une interprétation des schémas donne aussi une valeur entière aux paramètres. C'est pourquoi les nœuds d'un tableau généré par STAB contiennent aussi des contraintes : en plus de séparer les interprétations selon qu'elles satisfont une formule ou une autre, nous les séparons aussi selon le fait que les valeurs données aux paramètres satisfont ou non une contrainte. D'ailleurs, STAB comporte deux types de branchement : celui de la règle propositionnelle $\vee$ et ceux des règles d'itérations qui séparent l'espace de recherche selon que la contrainte $e \leq f$ est vraie ou fausse.

DPLL* est à STAB ce que DPLL est aux tableaux propositionnels, et à DPLL ce que STAB est aux tableaux propositionnels : l'ensemble des interprétations est maintenant séparé selon la valeur donnée aux variables propositionnelles (et non selon la structure de la formule) et selon les contraintes qui sont satisfaites par leurs paramètres. Tout comme pour STAB nous avons donc deux types de branchements (s est un schéma et L un ensemble de littéraux) :

PROPOSITIONAL SPLITTING :

$$\frac{(s, L)}{(s, L \cup \{p_e\}) \mid (s, L \cup \{\neg p_e\})}$$

si ni p_e ni $\neg p_e$ ne peuvent apparaître dans $(\bigwedge_{l \in L} l) \,\&\, c_s$.

CONSTRAINT SPLITTING ($\wedge$) :

$$\frac{(m_s[\bigwedge_{i|\delta} m] \,\&\, c_s, L)}{(m_s[\bigwedge_{i|\delta} m] \,\&\, c_s \wedge \exists i \delta, L) \mid (m_s[\top] \,\&\, c_s \wedge \forall i \neg \delta, L)}$$

CONSTRAINT SPLITTING ($\vee$) :

$$\frac{(m_s[\bigvee_{i|\delta} m] \,\&\, c_s, L)}{(m_s[\bigvee_{i|\delta} m] \,\&\, c_s \wedge \exists i \delta, L) \mid (m_s[\bot] \,\&\, c_s \wedge \forall i \neg \delta, L)}$$

La notation $m[m_1]$ en prémisse (ou, dans la suite, en membre gauche d'une règle de réécriture) signifie que le motif m_1 apparaît dans m; la même notation $m[m_2]$ en conclusion (ou en membre droit d'une règle de réécriture) représente le motif m dans lequel m_1 a été remplacé par m_2. L'application de ces règles est soumise à plusieurs conditions que nous verrons lors de leur définition formelle.

Dans STAB les règles qui séparaient les interprétations selon les contraintes, effectuaient le "dépliage" des itérations. Ici ce dépliage est effectué par une règle de réécriture indépendante :

UNFOLDING :

$$\Pi_{i|\delta}\, m \rightarrow m[e/i] \amalg \Pi_{i|\delta \wedge i \neq e}\, m$$

où e est une expression arithmétique choisie arbitrairement ne contenant pas i et telle que context$(s) \wedge \delta[e/i]$ est valide (p.ex. au chapitre 9, nous choisirons toujours la borne supérieure d'une itération encadrée) et s est le schéma réécrit.

Une fois que nous avons ajouté un littéral à l'ensemble de littéraux, nous voulons, comme pour DPLL, le remplacer par $\top$. Cependant dans le cas des schémas, ce n'est pas aussi simple que pour les formules propositionnelles. Encore une fois parce que les littéraux peuvent apparaître implicitement, p.ex. p_n peut apparaître implicitement dans $\bigwedge_{i=1}^{n} p_i$, donc si nous voulons lui substituer $\top$, il faut vérifier que $n \geq 1$, et le cas échéant dérouler l'itération pour pouvoir remplacer p_n, et uniquement lui, par $\top$. Ce mécanisme s'effectue via deux règles. La première réécrit une occurrence implicite potentielle de p_n, la réécriture ajoute à cette occurrence une formule arithmétique garantissant qu'elle *n'est pas* une occurrence implicite de p_n :

EXPANSION (positif) :

$$p_{e_1,\ldots,e_k} \rightarrow \bigwedge_{i|(e_1 \neq f_1 \vee \ldots \vee e_k \neq f_k) \wedge i = 0} p_{e_1,\ldots,e_k} \quad \text{si } p_{f_1,\ldots,f_k} \in L$$

Expansion (négatif) :

$$p_{e_1,\ldots,e_k} \rightarrow \bigvee_{i|(e_1\neq f_1\vee\ldots\vee e_k\neq f_k)\wedge i=0} p_{e_1,\ldots,e_k} \quad \text{si } \neg p_{f_1,\ldots,f_k} \in L$$

où i est une variable n'apparaissant pas dans s. Cette règle ne préserve pas la forme normale négative des schémas (nous pouvons ajouter une itération entre une négation et un atome). Mais il est facile de rétablir cette forme en appliquant, immédiatement après Expansion, les règles $\neg \bigwedge_{i|\delta} p_{e_1,\ldots,e_k} \rightarrow \bigvee_{i|\delta} \neg p_{e_1,\ldots,e_k}$ et $\neg \bigvee_{i|\delta} p_{e_1,\ldots,e_k} \rightarrow \bigwedge_{i|\delta} \neg p_{e_1,\ldots,e_k}$, ce que nous ferons toujours implicitement.

Notons que l'itération introduite est très particulière car i vaut toujours 0. En fait, au contraire des itérations habituelles qui peuvent être considérées comme des boucles "pour", cette itération doit plutôt être interprétée comme un "si ... alors ... sinon ..." : si la condition $e_1 \neq f_1 \vee \ldots \vee e_k \neq f_k$ n'est pas vérifiée alors l'itération est vide (donc le littéral $p_{e_1,\ldots,e_k}$ est supprimé), sinon l'itération est préservée et son contenu aussi (c.-à-d. le littéral $p_{e_1,\ldots,e_k}$). La deuxième règle assure ce comportement :

Deep splitting ($\wedge$) :

$$\bigwedge_{i|\delta} (m[\amalg_{i'|\delta'}\ m']) \rightarrow \bigwedge_{i|\delta\wedge\exists i'\delta'} (m[\amalg_{i'|\delta'}\ m']) \amalg \bigwedge_{i|\delta\wedge\forall i'\neg\delta'} (m[\top])$$

Deep splitting ($\vee$) :

$$\bigvee_{i|\delta} (m[\amalg_{i'|\delta'}\ m']) \rightarrow \bigvee_{i|\delta\wedge\exists i'\delta'} (m[\amalg_{i'|\delta'}\ m']) \amalg \bigvee_{i|\delta\wedge\forall i'\neg\delta'} (m[\bot])$$

Cette règle va bien au delà de la simple gestion du "si ... alors ... sinon ...". Il est facile de voir qu'elle est très similaire à Constraint Splitting. On peut considérer qu'elle joue le même rôle mais à l'intérieur des formules. Évidemment, elle ne sert pas à séparer les interprétations. Mais en décomposant la formule en, d'une part, la sous formule obtenue quand l'itération interne est vide, et d'autre part, la sous formule obtenue quand elle n'est pas vide, cette règle permet des simplifications souvent pertinentes. De plus nous voyons que l'information "remonte" le long des itérations : initialement, la condition assurant que l'itération est vide est codée dans l'itération elle-même; après application de la règle, cette information est codée dans l'itération englobante, au prix toutefois d'une duplication de cette itération. Ainsi la condition finira par arriver à la racine du schéma et pourra alors donner lieu à une application utile de Constraint Splitting.

Enfin, nous regroupons toutes les règles "algébriques", c.-à-d. les règles qui appliquent des simplifications triviales du schéma. Ce sont elles qui permettent de conclure en réécrivant un schéma trivialement insatisfaisable en $\bot$:

$$\neg\top \rightarrow \bot \quad m\wedge\top \rightarrow m \quad m\wedge\bot \rightarrow \bot \quad \bigwedge_{i|\delta} \top \rightarrow \top \quad m\wedge m \rightarrow m$$

$$\neg\bot \rightarrow \top \quad m\vee\bot \rightarrow m \quad m\vee\top \rightarrow \top \quad \bigwedge_{i|\delta} \bot \rightarrow \bot \quad m\vee m \rightarrow m$$

$$\text{si context}(s)\wedge\exists i\delta \text{ est insatisfaisable :} \quad \bigwedge_{i|\delta} m \rightarrow \top \qquad \bigvee_{i|\delta} m \rightarrow \bot$$

$$\text{si } m \text{ ne contient pas } i \text{ et context}(s) \Rightarrow \exists i\delta \text{ est valide :} \quad \amalg_{i|\delta}\, m \rightarrow m$$

Nous définissons maintenant formellement Dpll*. Toute règle de Dpll* prend en prémisse un schéma syntaxiquement encadré (définition 3.23)[2]. Nous verrons dans la preuve du lemme 8.5 que cette restriction sur les prémisses est nécessaire pour s'assurer qu'il est toujours possible d'appliquer une règle, c.-à-d. que le système ne peut pas rester "bloqué" artificiellement.

Dpll* *ne préserve pas* l'encadrement syntaxique. Nous devons en fait utiliser les résultats de la section 3.C : un schéma syntaxiquement encadré est un schéma quasi-syntaxiquement et linéairement encadré. La propriété d'encadrement linéaire est préservée par Dpll* (proposition 8.3), mais pas celle d'encadrement quasi-syntaxique. Ceci peut être facilement rectifié en utilisant, si nécessaire, la procédure

[2]Notons que, d'après les résultats de la section 3.C, il suffit en fait qu'il soit *linéairement* encadré. En effet nous pouvons transformer tout schéma en un schéma quasi-syntaxiquement encadré par application de l'algorithme R_{syn}, et donc en schéma syntaxiquement encadré si le schéma est linéaire.

R_{syn} donnée en 3.44 entre deux applications d'une règle de DPLL* pour ramener les schémas introduits par le système à des schémas syntaxiquement encadrés.

DPLL* utilise la notion de *contexte* : soit un schéma s dont toutes les variables liées sont différentes (et différentes des paramètres), nous appelons *contexte arithmétique* de s, noté context(s), la formule arithmétique $c_s \wedge \delta_1 \wedge \ldots \wedge \delta_k$ où $\delta_1, \ldots, \delta_k$ sont les domaines de toutes les itérations apparaissant dans s. Nous supposons donc que toutes les variables libres sont différentes[3], ce n'est pas restrictif car tout schéma peut être transformé en un schéma ayant cette propriété par α-conversion. Le contexte arithmétique permet par exemple de savoir si une itération est vide, même lorsqu'elle est située "en profondeur".

Definition 8.1 (DPLL*)**.** *Chaque nœud d'un tableau de* DPLL* *est étiqueté par une paire d'un schéma et d'un ensemble fini de littéraux. Nous notons $s(\nu)$ (resp. $L(\nu)$) le schéma (resp. l'ensemble de littéraux) étiquetant un nœud ν. Les règles sont les suivantes :*

- PROPOSITIONAL SPLITTING *:*

$$\frac{(s, L)}{(s, L \cup \{p_e\}) \mid (s, L \cup \{\neg p_e\})}$$

si p_e ou $\neg p_e$ apparaît toujours dans s et ni p_e ni $\neg p_e$ ne peuvent apparaître dans $(\bigwedge_{l \in L} l) \,\&\, c_s$.

- CONSTRAINT SPLITTING *:*

$$\frac{(m_s[\Pi_{i|\delta}\, m] \,\&\, c_s, L)}{(m_s[\Pi_{i|\delta}\, m] \,\&\, c_s \wedge \exists i\delta, L) \mid (m_s[\epsilon] \,\&\, c_s \wedge \forall i \neg\delta, L)}$$

où $(\Pi, \epsilon) \in \{(\bigwedge, \top), (\bigvee, \bot)\}$, et si $c_s \wedge \forall i \neg\delta$ est satisfaisable et toutes les variables libres de δ sont des paramètres, sauf i.

- REWRITING *:*

$$\frac{(s, L)}{(s', L)}$$

où $c_{s'} = c_s$ et $m_{s'}$ est obtenu en réécrivant m_s selon les règles de réécriture suivantes (appliquées à une profondeur quelconque) :

- ALGEBRAIC SIMPLIFICATIONS *:*

$$\neg\top \to \bot \quad m \wedge \top \to m \quad m \wedge \bot \to \bot \qquad \bigwedge_{i|\delta} \top \to \top \qquad m \wedge m \to m$$

$$\neg\bot \to \top \quad m \vee \bot \to m \quad m \vee \top \to \top \qquad \bigwedge_{i|\delta} \bot \to \bot \qquad m \vee m \to m$$

si context(s) $\wedge\, \exists i\delta$ *est insatisfaisable :* $\bigwedge_{i|\delta} m \to \top \qquad \bigvee_{i|\delta} m \to \bot$

si m ne contient pas i et context(s) $\Rightarrow \exists i\delta$ *est valide :* $\Pi_{i|\delta}\, m \to m$

- UNFOLDING *:*

$$\Pi_{i|\delta}\, m \to m[e/i] \,\sqcap\, \Pi_{i|\delta \wedge i \neq e}\, m$$

où e est une expression arithmétique choisie arbitrairement ne contenant pas i et telle que context(s) $\wedge\, \delta[e/i]$ *est valide.*

- DEEP SPLITTING *:*

$$\Pi_{i|\delta}\,(m[\,\Pi_{i'|\delta'}\; m']) \to \Pi_{i|\delta \wedge \exists i'\delta'}\,(m[\,\Pi_{i'|\delta'}\; m']) \,\sqcap\, \Pi_{i|\delta \wedge \forall i' \neg\delta'}\;(m[\epsilon])$$

où $(\Pi, \epsilon) \in \{(\bigwedge, \top), (\bigvee, \bot)\}$, et si context($s$)$\wedge\forall i' \neg\delta'$ *est satisfaisable et i apparaît linéairement dans δ'.*

[3]Les règles de DPLL* préservent cette propriété, sauf DEEP SPLITTING. Mais il est facile de corriger cela en supposant que les variables de l'une des itérations dupliquées sont renommées de sorte à ce que la propriété soit vérifiée. On ne le fait pas par souci de lisibilité.

- EXPANSION :

$$p_{e_1,\ldots,e_k} \rightarrow \Pi_{i|(e_1\neq f_1\vee\ldots\vee e_k\neq f_k)\wedge i=0}\ p_{e_1,\ldots,e_k}$$

où $f_1,\ldots,f_k$ sont les indices d'un atome $p_{f_1,\ldots,f_k}$ apparaissant dans L, $\Pi = \bigwedge$ si $p_{f_1,\ldots,f_k} \in L$ ou $\Pi = \bigvee$ si $\neg p_{f_1,\ldots,f_k} \in L$ et si context$(s) \wedge e_1 = f_1 \wedge \ldots \wedge e_k = f_k$ *est satisfaisable. i est une variable n'apparaissant pas dans s.*

Une feuille ν est close ssi $m_{s(\nu)} = \bot$ ou $c_{s(\nu)}$ est insatisfaisable.

Pour la sémantique (et pour le bouclage) nous associons à chaque nœud ν le schéma suivant :

$$sL(\nu) \stackrel{def}{=} m_{s(\nu)} \wedge \left(\bigwedge_{l\in L(\nu)} l\right) \,\&\, c_{s(\nu)}$$

Comme la définition de schéma n'est pas purement syntaxique – il faut s'assurer que toutes les itérations ont des domaines encadrés – nous devons vérifier que tous les objets obtenus par application des règles de DPLL* sont bien des schémas.[4]

Proposition 8.2. *Soit un nœud ν d'un tableau construit par* DPLL**, $s(\nu)$ est bien un schéma.*

Preuve. Il suffit de vérifier que si une itération est encadrée avant application d'une règle, alors elle l'est toujours après. Les seules règles à modifier les itérations sont UNFOLDING et DEEP SPLITTING. Ces deux règles ne font que restreindre un domaine déjà existant. Donc si ce domaine était encadré avant, il l'est toujours après. Par ailleurs les seules itérations introduites par le système sont celle obtenues par EXPANSION, elles sont trivialement encadrées car elles imposent $i = 0$. □

De plus les règles DPLL* ne prend en entrée que des schémas syntaxiquement encadrés[5]. Comme nous l'avons dit en introduction, cette propriété n'est pas préservée par DPLL* : la règle DEEP SPLITTING ajoute des quantificateurs ce qui est contraire à la définition de schéma syntaxiquement encadré. De même, la règle UNFOLDING ajoute un atome contenant i à une itération et la règle EXPANSION ajoute une itération dont le domaine contient des disjonctions. En revanche la propriété d'encadrement linéaire est préservée, comme le montre la proposition suivante. Ceci nous permet d'utiliser l'algorithme $\mathrm{R}_{\mathrm{syn}}$ après application des règles DEEP SPLITTING et UNFOLDING afin de se ramener à un schéma syntaxiquement encadré comme requis par la définition de DPLL*.

Proposition 8.3. *Soit un nœud ν d'un tableau construit par* DPLL**, $s(\nu)$ est linéairement encadré.*

Preuve. ν est soit le premier nœud, auquel cas le résultat est trivial, soit un nœud obtenu par application d'une règle de DPLL*. Dans ce cas nous inspectons chaque règle : PROPOSITIONAL SPLITTING, CONSTRAINT SPLITTING et ALGEBRAIC SIMPLIFICATIONS ne modifient pas le domaine des itérations donc préservent l'encadrement linéaire. En revanche les autres règles modifient bien les itérations ou en introduisent de nouvelles. Toutes ces règles préservent la linéarité : la seule expression introduite par UNFOLDING et contenant i est i, donc la linéarité est préservée; pour DEEP SPLITTING, le résultat est obtenu grâce à la condition d'application de la règle "i apparaît linéairement dans δ'."; et pour EXPANSION, le seul atome contenant i est $i = 0$ qui est bien linéaire par rapport à i.

Cependant, la preuve n'est pas terminée : les itérations introduites par ces trois règles ne sont pas syntaxiquement encadrées, donc il faut appliquer $\mathrm{R}_{\mathrm{syn}}$ pour qu'elles le deviennent. Nous devons nous assurer que le résultat de cette application est toujours linéairement encadré. Il est facile de voir que la seule règle de $\mathrm{R}_{\mathrm{syn}}$ pouvant introduire des expressions non linéaires est FORME NORMALE ARITHMÉTIQUE, *via* l'élimination des quantificateurs. Parmi les règles de DPLL*, seule DEEP SPLITTING peut introduire des quantificateurs, donc il suffit de vérifier que les domaines introduits par cette règle restent linéaires après élimination des quantificateurs. C'est bien le cas car δ' est syntaxiquement encadrée par rapport à i', donc δ' a la forme $e \leq i' \wedge i' \leq f \wedge \phi$ où ϕ est une formule sans disjonction ni quantificateur et ne contenant pas i'. Par conséquent $\exists i'\delta'$ et $\forall i'\neg\delta'$ seront réécrits en $e \leq f \wedge \phi$ et $e > f \vee \neg\phi$: ces deux formules ne contiennent plus de quantificateurs et sont bien linéaires par rapport à i. □

Corollary 8.4. *La règle* LINÉARISATION *de* $\mathrm{R}_{\mathrm{syn}}$ *n'est jamais appliquée dans* DPLL*.

[4] Pour rappel, il est nécessaire d'avoir des domaines encadrés afin d'assurer que toute instance d'un schéma est finie.

[5] Nous verrons dans la preuve du lemme 8.5 l'utilité de cette restriction.

8.3 Example

Afin de se familiariser avec DPLL* nous donnons un exemple de dérivation avant de s'intéresser à la correction et à la complétude.

8.3.1 Sans imbrication

Nous prenons, une fois encore, l'exemple $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$. Un tableau construit avec DPLL* est présenté en figure 8.1 . Pour le bouclage, il faut garder à l'esprit que c'est le schéma suivant

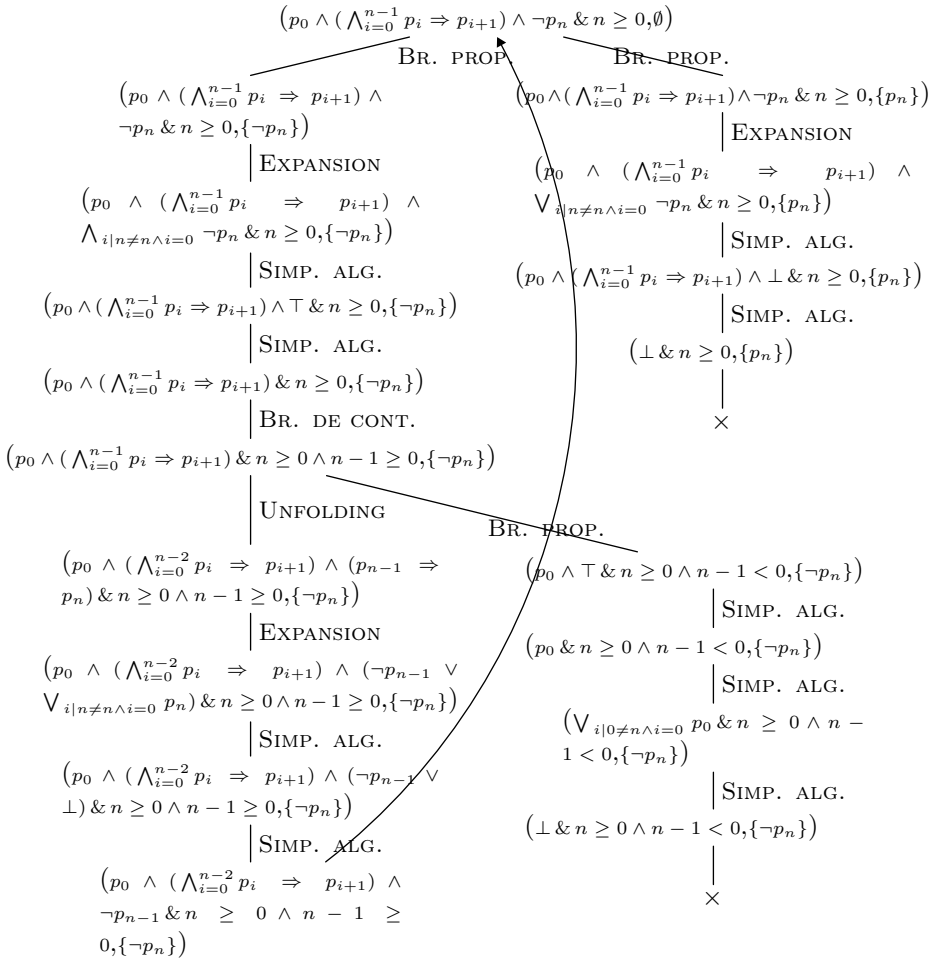


Figure 8.1: Application de DPLL* à $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$.

qui doit boucler :

$$sL(\nu) \stackrel{\text{def}}{=} m_{s(\nu)} \wedge \left(\bigwedge_{l \in L(\nu)} l \right) \,\&\, c_{s(\nu)}$$

et pas seulement $s(\nu)$. Il faut donc tenir compte de l'ensemble de littéraux, c.-à-d., pour la feuille qui boucle dans la figure 8.1, de $\{\neg p_n\}$. Or, dans cette feuille, $\neg p_n$ est pur, donc, avec l'extension du littéral pur, nous pouvons ne pas considérer ce littéral. Enfin nous utilisons aussi l'extension de la contrainte équivalente : nous avons $n \geq 0 \wedge n-1 \geq 0 \equiv n-1 \geq 0$, or $n-1 \geq 0$ est égale à la contrainte du schéma d'origine à un décalage près, ce qui permet de conclure.

8.3.2 Avec imbrication

Nous reprenons l'exemple de la section 8.1, dont nous avons vu qu'il ne terminait pas avec Stab : $\left(\bigwedge_{i=1}^{n}\bigvee_{j=1}^{n} p_i \Rightarrow q_j\right) \wedge \bigwedge_{i=1}^{n} \neg q_i \wedge \bigvee_{i=1}^{n} p_i$. Un tableau construit avec Dpll* est présenté en figures 8.2 , 8.3 et 8.4 . La première figure détaille les règles une par une, les suivantes ne donnent que la structure générale. Notons, dans le dernier arbre que le bouclage est rendu possible grâce à la règle $m \wedge m \to m$ de Algebraic simplifications : cette règle peut sembler superflue de prime abord, mais elle est utile pour *aider* le bouclage.

8.4 Soundness

Tout d'abord, l'application de chaque règle génère un schéma équivalent :

Lemma 8.5. *Soient :*

- *une interprétation* $\mathfrak{I}$*,*
- *une feuille* ν *d'un tableau* t*,*
- *et un tableau* t' *obtenu par application d'une règle de* Dpll* *en* ν*.*

Alors $\mathfrak{I} \models \nu$ *ssi il existe un fils* ν' *de* ν *dans* t' *t.q.* $\mathfrak{I} \models \nu'$*.*

On rappelle que $\mathfrak{I} \models \nu$ ssi $\mathfrak{I} \models sL(\nu)$, c.-à-d. $\mathfrak{I} \models m_{s(\nu)} \wedge \left(\bigwedge_{l \in L(\nu)} l\right) \,\&\, c_{s(\nu)}$.

Preuve. Par inspection des règles.

Supposons que t' est obtenu par application de Propositional Splitting sur ν et soit $\mathfrak{M}$ un modèle de ν. Pour toute interprétation $\mathfrak{I}$ et tout littéral p_e nous avons soit $\mathfrak{I} \models p_e$, soit $\mathfrak{I} \models \neg p_e$. Donc nous avons soit $\mathfrak{M} \models sL(\nu) \wedge p_e$, soit $\mathfrak{M} \models sL(\nu) \wedge \neg p_e$, c.-à-d. $\mathfrak{M}$ est un modèle soit du nœud de gauche, soit du nœud de droite. Réciproquement si $\mathfrak{M}$ est un modèle de l'un des deux fils alors $\mathfrak{M} \models sL(\nu) \wedge p_e$ ou $\mathfrak{M} \models sL(\nu) \wedge \neg p_e$, donc $\mathfrak{M} \models sL(\nu)$, c.-à-d. $\mathfrak{M} \models \nu$.

Supposons maintenant que t' est obtenu par application de Constraint Splitting et soit $\mathfrak{M}$ un modèle de ν. Pour toute interprétation $\mathfrak{I}$ et toute formule ϕ de l'arithmétique linéaire, nous avons soit $\mathfrak{I}_{\text{arith}} \models \phi$ soit $\mathfrak{I}_{\text{arith}} \models \neg\phi$. Donc nous avons soit $\mathfrak{M} \models sL(\nu) \,\&\, \exists i\delta$, soit $\mathfrak{M} \models sL(\nu) \,\&\, \forall i \neg\delta$, c.-à-d. $\mathfrak{M}$ est un modèle soit du nœud de gauche, soit du nœud de droite. Réciproquement si $\mathfrak{M}$ est un modèle de l'un des deux fils alors $\mathfrak{M} \models sL(\nu) \,\&\, \exists i\delta$ ou $\mathfrak{M} \models sL(\nu) \,\&\, \forall i \neg\delta$, donc $\mathfrak{M} \models sL(\nu)$, c.-à-d. $\mathfrak{M} \models \nu$.

Pour Algebraic simplifications, Unfolding et Deep splitting chaque membre gauche est équivalent à son membre droit (et même plus précisément : si une règle réécrit un motif m en un motif m', alors $\langle m \rangle_\rho \equiv \langle m' \rangle_\rho$ pour tout environnement ρ).

Pour Expansion, soit un modèle $\mathfrak{M}$ de ν. On a donc $\mathfrak{M} \models sL(\nu)$ et par conséquent $\mathfrak{M} \models l$ pour tout littéral l de $L(\nu)$. C'est donc le cas de $p_{f_1,\ldots,f_k}$. Soit un atome de la forme $p_{g_1,\ldots,g_k}$ apparaissant dans $\langle s \rangle_{\mathfrak{M}_{\text{arith}}}$. D'après la définition 3.8, c'est qu'il existe un atome $p_{e_1,\ldots,e_k}$ de s et un environnement ρ tels que $p_{g_1,\ldots,g_k} = p_{\rho(e_1),\ldots,\rho(e_k)}$.

- si $\rho(e_1) = \rho(f_1)$, $\ldots$, $\rho(e_k) = \rho(f_k)$ alors, comme $p_{f_1,\ldots,f_k} \in L(\nu)$, $\mathfrak{M} \models p_{g_1,\ldots,g_k}$, d'une part, et $\langle \bigwedge_{i|(e_1 \neq f_1 \vee \ldots \vee e_k \neq f_k) \wedge i=0} p_{e_1,\ldots,e_k} \rangle_\rho = \top$ d'autre part (car le domaine n'est satisfaisable pour aucune valeur de i). Donc $\mathfrak{M} \models \langle \bigwedge_{i|(e_1 \neq f_1 \vee \ldots \vee e_k \neq f_k) \wedge i=0} p_{e_1,\ldots,e_k} \rangle_\rho$. Ainsi les deux membres sont tous deux interprétés à $\top$.
- si $\rho(e_1) \neq \rho(f_1)$ ou $\ldots$ ou $\rho(e_k) \neq \rho(f_k)$, alors le domaine est satisfaisable et la seule valeur possible pour i est 0 (car le domaine implique $i = 0$) donc $\langle \bigwedge_{i|(e_1 \neq f_1 \vee \ldots \vee e_k \neq f_k) \wedge i=0} p_{e_1,\ldots,e_k} \rangle_\rho = \langle p_{e_1,\ldots,e_k} \rangle_{\rho \oslash \{i \mapsto 0\}}$. Comme i n'apparaît pas dans s, $\langle p_{e_1,\ldots,e_k} \rangle_{\rho \oslash \{i \mapsto 0\}} = \langle p_{e_1,\ldots,e_k} \rangle_\rho = p_{g_1,\ldots,g_k}$. Donc les deux membres de la règle ont tous deux la même interprétation par $\mathfrak{M}$.

Dans tous les cas la valeur de $p_{g_1,\ldots,g_k}$ est préservée par la réécriture, donc $\mathfrak{M}$ est toujours un modèle après application de la règle. Comme $p_{f_1,\ldots,f_k}$ appartient aussi à l'ensemble de littéraux étiquetant le fils de ν, le raisonnement est le même pour prouver la réciproque.

Enfin, notons que toute forme normale d'un schéma par R_{syn} est équivalente à ce schéma (proposition 3.43), donc l'application de ce système entre deux règles préserve la correction. □

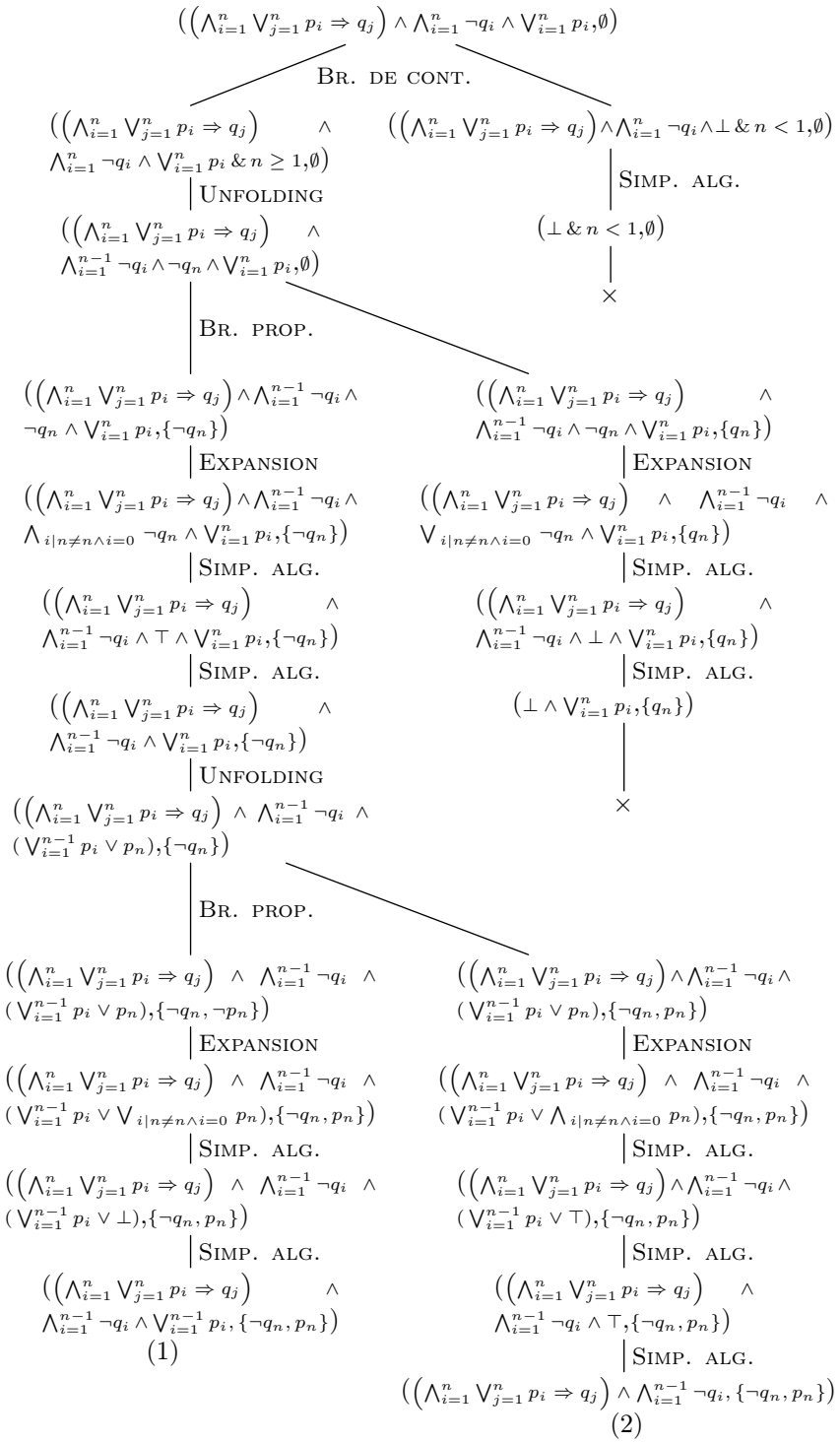


Figure 8.2: Application de Dpll* à $\left(\bigwedge_{i=1}^{n}\bigvee_{j=1}^{n} p_i \Rightarrow q_j\right) \wedge \bigwedge_{i=1}^{n} \neg q_i \wedge \bigvee_{i=1}^{n} p_i$.

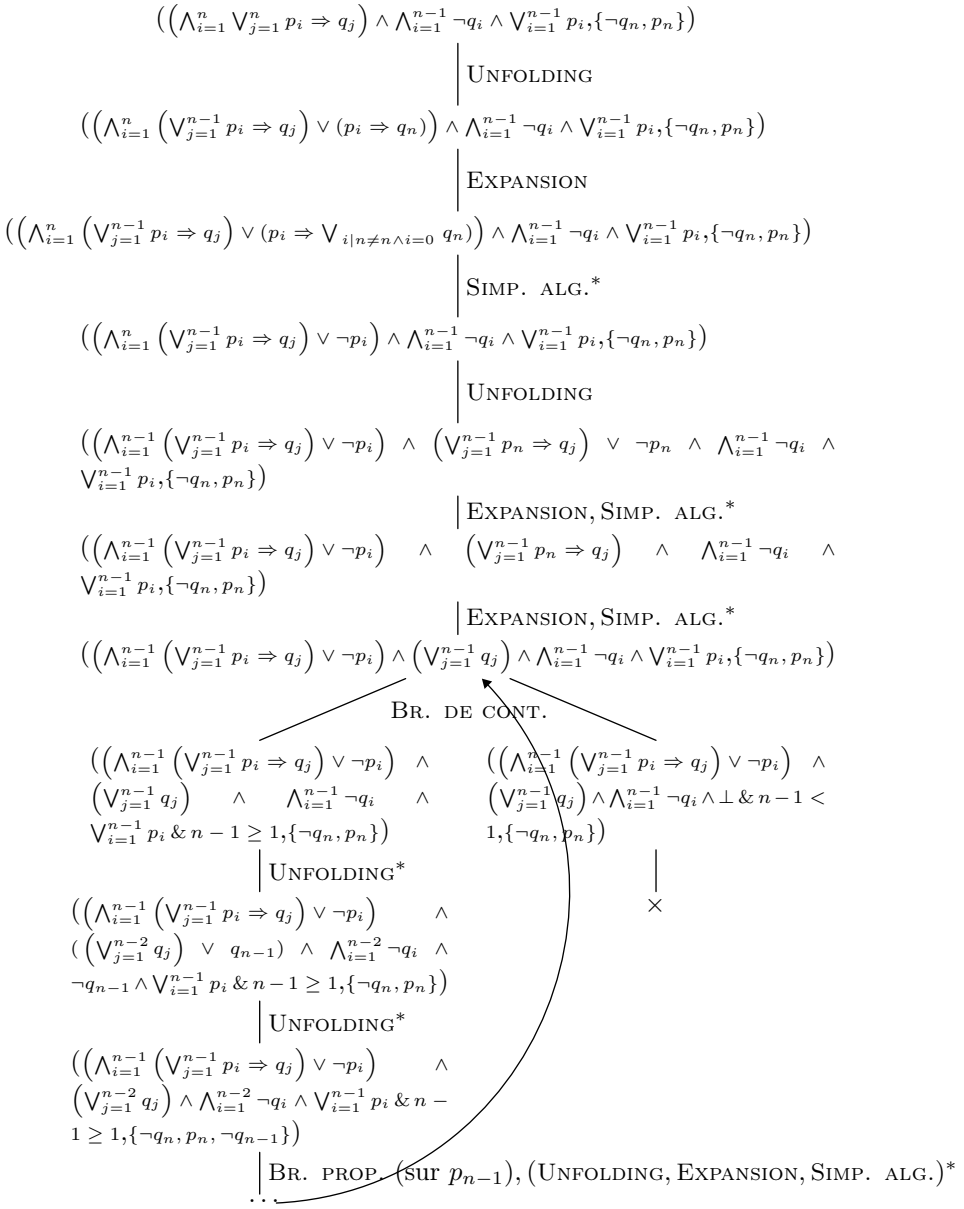


Figure 8.3: Application de DPLL* au nœud (1) de la figure 8.2 .

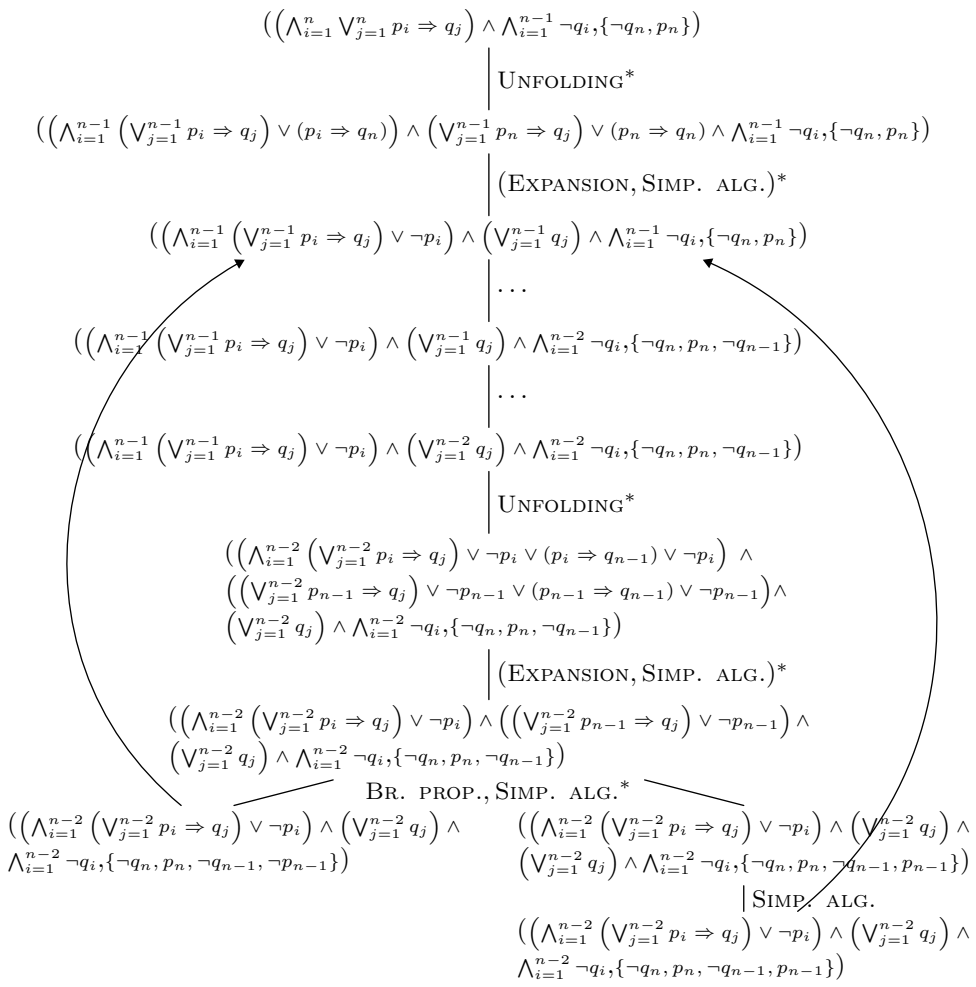


Figure 8.4: Application de Dpll* au nœud (2) de la figure 8.2.

Le principe de nombreuses procédures de preuve consiste à "simplifier" successivement une formule de façon à obtenir une formule dont la satisfaisabilité sera triviale. Il est donc essentiel que lorsqu'une feuille est irréductible, nous sachions de façon simple si elle est satisfaisable ou non. Dans le cas ou une telle feuille est close, il est clair qu'elle est insatisfaisable. Si ce n'est pas le cas, alors elle est satisfaisable :

Lemma 8.6. *Si une feuille ν d'un tableau t est irréductible et non close alors t est satisfaisable.*

La preuve de ce lemme utilise de façon essentielle le fait que les itérations sont syntaxiquement encadrées. Ceci permet en effet d'utiliser la proposition 3.24 pour extraire un rang (typiquement le dernier) d'une itération dont nous savons qu'elle est non vide.

Preuve. Nous montrons d'abord que $s(\nu)$ ne contient pas d'itération. En effet supposons qu'il existe une itération. Sans perte de généralité nous pouvons supposer que cette itération n'est pas contenue dans une autre itération. Aucune instance de cette itération ne peut être vide. En effet s'il existait un environnement pour lequel l'itération est vide alors Constraint Splitting s'appliquerait, ce qui est impossible par irréductibilité. Donc, quel que soit l'environnement, l'itération n'est pas vide. De plus toute itération est syntaxiquement encadrée. Donc il existe un rang e satisfaisant le domaine de l'itération quel que soit l'environnement, d'après la proposition 3.24. On peut donc appliquer Unfolding, ce qui est impossible par irréductibilité. Ainsi il est impossible d'avoir une itération.

Nous montrons ensuite que $s(\nu)$ ne peut pas contenir de littéraux. En effet, supposons que $s(\nu)$ contienne un littéral l. Dans ce cas l apparaît toujours dans $s(\nu)$, car $s(\nu)$ ne contient pas d'itération. Donc, par irréductibilité de Propositional Splitting, l a nécessairement été ajouté à $L(\nu)$. Par conséquent, par irréductibilité de Expansion, cette règle a nécessairement été appliquée en prenant $p_{e_1,\dots,e_k} = l$ (car l apparaît dans le schéma) et $p_{f_1,\dots,f_k} = l$ (car $l \in L(\nu)$). L'itération introduite alors est insatisfaisable et a donc été éliminée, par irréductibilité de Algebraic simplifications, qui l'a remplacée par $\top$ ou $\bot$ (note : nous venons précisément de décrire la façon dont DPLL* simule DPLL). Ainsi l ne peut pas apparaître dans $s(\nu)$.

$s(\nu)$ ne contient donc ni itération, ni littéral. Il est donc construit uniquement à partir de $\top$, $\bot$, $\neg$, $\vee$ et $\wedge$. Par irréductibilité de Algebraic simplifications, un tel schéma est nécessairement réduit à $\bot$ ou $\top$. Comme la feuille n'est pas close, ce ne peut être $\bot$. On a donc $s(\nu) = \top$ qui est satisfaisable. Enfin $L(\nu)$ ne peut pas contenir deux littéraux opposés car la condition d'application de Propositional Splitting assure qu'un littéral n'est ajouté que si ni lui-même ni sa négation ne peuvent déjà y apparaître. Ainsi $sL(\nu)$ est bien satisfaisable. Il en va donc de même de ν et donc de t. □

Theorem 8.7 (Correction)**.** *Soit t' un tableau obtenu par application d'une règle de* DPLL* *à un tableau t. Si t' contient une feuille irréductible non close alors t est satisfaisable.*

Preuve. C'est une conséquence immédiate des lemmes 8.5 et 8.6. □

8.5 Completeness for Satisfiability

Nous prouvons la complétude pour la satisfaisabilité de DPLL* comme nous l'avons fait pour Stab : étant donné un modèle, nous donnons une mesure associée à ce modèle et au schéma étiquetant un nœud de sorte que cette mesure diminue strictement à chaque application de règle. Cette mesure étant bien fondée, nous garantissons ainsi la terminaison en cas de satisfaisabilité du schéma.

Intuitivement, nous analysons une dérivation de DPLL* en nous focalisant uniquement sur une branche dont tous les nœuds satisfont un modèle $\mathfrak{M}$ donné (d'après le lemme 8.5, il existe toujours une telle branche). L'intuition derrière la mesure est alors la suivante : toutes les itérations vont être progressivement dépliées. Ceci ne peut avoir lieu indéfiniment car nous nous focalisons sur la branche dont les nœuds satisfont $\mathfrak{M}$: dans une interprétation donnée les itérations ont une longueur fixée (ou plutôt dans l'instance associée à une interprétation). Concrètement, il arrivera un nœud dont toutes les itérations seront vides. Celles-ci seront alors éliminées par Constraint Splitting ou Deep splitting. Une fois ceci fait pour toutes les itérations, il ne reste plus qu'une formule propositionnelle. Cette formule correspond à l'instance du schéma initial par rapport à $\mathfrak{M}_{\text{arith}}$ (dans les faits, il ne s'agit pas précisément de cette instance car des littéraux peuvent avoir été évalués entre temps et donc éliminés). Enfin Algebraic simplifications va simplifier la formule obtenue par rapport à la valeur des atomes dans $\mathfrak{M}$ (en tout cas ceux qui pour lesquels la formule n'a pas encore été simplifiée), jusqu'à obtenir $\top$.

Comme le lecteur pourra le constater, la mesure définie ci-après n'est pas simple (particulièrement $\mu^2_{\mathfrak{M}}$). Avant de la définir formellement nous présentons brièvement les problèmes rencontrés qui justifient une définition aussi complexe. D'après les explications précédentes, la mesure d'un schéma qui contient des itérations doit être plus grande que celle du même schéma dans lequel les itérations ont été dépliées. De plus, plus la longueur de ces itérations est grande dans $\mathfrak{M}$, plus la mesure du schéma doit être grande : ceci pour exprimer le fait que toutes les itérations doivent être dépliées. C'est typiquement le cas après une application de Unfolding : l'itération, après réécriture, contient un rang de moins que l'itération avant réécriture. En revanche Deep splitting ne diminue pas la longueur des itérations puisqu'il divise une itération en deux itérations de longueurs égales. Nous pouvons facilement contourner ce problème en élevant au carré la longueur des itérations. Mais un problème plus important apparaît avec Expansion : cette règle *ajoute* des itérations qui n'existaient pas avant application de la règle. Dans ce cas, la mesure esquissée jusqu'à maintenant ne peut pas diminuer. Une façon classique de résoudre ce problème est de définir une autre mesure spécifiquement pour Expansion (p.ex. le nombre d'applications possibles de la règle) et de la combiner avec la mesure précédente en considérant, p.ex., l'extension lexicographique de ces deux mesures. Mais ceci ne marche pas si nous prenons pour mesure le nombre d'applications possibles de Expansion, car Unfolding duplique le motif m et multiplie donc les possibles applications de Expansion : nous ne pouvons pas ordonner ces deux mesures. En fait nous n'avons pas trouvé de solution permettant un traitement de ces deux règles. Nous avons donc dû gérer plusieurs règles en même temps. Nous aurions peut-être pu y parvenir via des compositions de fonctions classiques (additions, multiplications, exponentielles), mais il est plus simple de résoudre ce problème en définissant directement des fonctions récursives dédiées : μ_{it} et μ_{l}. Notons enfin que les mêmes problèmes se posent avec Deep splitting et avec l'application des règles du système $\mathrm{R}_{\mathrm{syn}}$ de la section 3.44 (en effet, plusieurs règles introduisent des schémas qui ne sont pas syntaxiquement encadrés; il faut donc, après chaque application de ces règles, ramener la conclusion à un schéma syntaxiquement encadré).

Nous donnons maintenant la définition formelle de notre mesure. Pour ce faire, nous définissons d'abord les fonctions suivantes :

$$\mu_{\mathrm{it}}(x,0) \stackrel{\mathrm{def}}{=} (x+2)^2 \qquad \mu_{\mathrm{l}}(0) \stackrel{\mathrm{def}}{=} 1$$
$$\mu_{\mathrm{it}}(x,k+1) \stackrel{\mathrm{def}}{=} (\mu_{\mathrm{it}}(x,k)+x+2)^2 \quad \mu_{\mathrm{l}}(k+1) \stackrel{\mathrm{def}}{=} \mu_{\mathrm{it}}(\mu_{\mathrm{l}}(k),0)+1$$

L'intérêt de ces deux fonctions réside dans le fait qu'elles vérifient les propriétés suivantes, utiles pour démontrer que la mesure diminue. Toute autre paire de fonctions satisfaisant ces propriétés suffirait à assurer la complétude.

Proposition 8.8.

1. $\forall x,k \in \mathbb{N}, \mu_{\mathrm{it}}(x,k) \geq 4$
2. $\forall x,k \in \mathbb{N}, \mu_{\mathrm{it}}(x,k) \geq x$
3. $\forall x,y,k \in \mathbb{N}, x < y \Rightarrow \mu_{\mathrm{it}}(x,k) < \mu_{\mathrm{it}}(y,k)$
4. $\forall k_1,k_2 \in \mathbb{N}, k_1 < k_2 \Rightarrow \mu_{\mathrm{l}}(k_1) < \mu_{\mathrm{l}}(k_2)$
5. $\forall x,k_1,k_2 \in \mathbb{N}, k_1 < k_2 \Rightarrow \mu_{\mathrm{it}}(x,k_1) < \mu_{\mathrm{it}}(x,k_2)$
6. $\forall x,y,k \in \mathbb{N}, y \geq 1 \Rightarrow \mu_{\mathrm{it}}(x,k)+y < \mu_{\mathrm{it}}(x+y,k)$
7. $\forall x,y,k \in \mathbb{N}, \mu_{\mathrm{it}}(x,k)+\mu_{\mathrm{it}}(y,k) < \mu_{\mathrm{it}}(x+y,k+1)$
8. $\forall k \in \mathbb{N}, \mu_{\mathrm{it}}(\mu_{\mathrm{l}}(k),0) < \mu_{\mathrm{l}}(k+1)$
9. $\forall k,q,x_1,\ldots,x_q \in \mathbb{N}, \sum_{i=1}^{q} \mu_{\mathrm{it}}(x_i,k) < \mu_{\mathrm{it}}(\sum_{i=1}^{q} x_i, k+q)$

Preuve.

1. Trivial.
2. Trivial.

3. Par récurrence sur k : si $k = 0$ alors $\mu_{\text{it}}(x,k) = x^2 + 2x + 4 < y^2 + 2y + 4 = \mu_{\text{it}}(y,k)$. Si $k > 0$ alors $\mu_{\text{it}}(x,k) = \mu_{\text{it}}(x,k-1) + x + 2^2$. Par hypothèse de récurrence, $\mu_{\text{it}}(x,k-1) < \mu_{\text{it}}(x,k)$, donc $\mu_{\text{it}}(x,k-1) + x + 2^2 < \mu_{\text{it}}(y,k-1) + y + 2^2$, d'où le résultat.

4. Par récurrence sur k_1 : si $k_1 = 0$ alors $\mu_{\text{l}}(k_1) = 1$. De plus, comme $k_2 > k_1 = 0$, $\mu_{\text{l}}(k_2) = \mu_{\text{it}}(\mu_{\text{l}}(k_2 - 1), 0) + 1$. D'après la première assertion, nous avons donc $\mu_{\text{l}}(k_2) \geq 5$, c.-à-d. $\mu_{\text{l}}(k_2) > \mu_{\text{l}}(k_1) = 1$. Si $k_1 > 0$ alors $\mu_{\text{l}}(k_1) = \mu_{\text{it}}(\mu_{\text{l}}(k_1 - 1), 0) + 1$ et comme $k_2 > k_1$, donc $k_2 > 0$: $\mu_{\text{l}}(k_2) = \mu_{\text{it}}(\mu_{\text{l}}(k_2 - 1), 0) + 1$. Par hypothèse de récurrence $\mu_{\text{l}}(k_1 - 1) < \mu_{\text{l}}(k_2 - 1)$ et d'après l'assertion précédente, μ_{it} est croissante, d'où le résultat.

5. Il suffit de montrer que $\mu_{\text{it}}(x,k) < \mu_{\text{it}}(x,k+1)$, ce qui est bien le cas vu la définition de $\mu_{\text{it}}(x,k+1)$.

6. Par récurrence sur k. Si $k = 0$, alors $\mu_{\text{it}}(x,k) + y < \mu_{\text{it}}(x+y,k)$ est équivalent à $2x + y < 2xy + y^2$. Cette dernière inéquation est valide car $y \geq 1$ par hypothèse. Si $k \geq 0$ alors $\mu_{\text{it}}(x+y,k) = (\mu_{\text{it}}(x+y,k) + x + y + 2)^2$. Par hypothèse de récurrence, cette expression est strictement supérieure à $(\mu_{\text{it}}(x,k) + x + 2y + 2)^2$, et donc à $(\mu_{\text{it}}(x,k) + x + 2)^2 + 4y^2 + 2y(\mu_{\text{it}}(x,k) + x + 2)$, et finalement à $(\mu_{\text{it}}(x,k) + x + 2)^2$.

7. Par récurrence sur k. Si $k = 0$, alors $\mu_{\text{it}}(x,k) + \mu_{\text{it}}(y,k) = (x+2)^2 + (y+2)^2$ et $\mu_{\text{it}}(x+y,k+1) = (\mu_{\text{it}}(x+y,k) + x + y + 2)^2 \geq (4 + x + y + 2)^2$ d'après 1. Il est ensuite facile de voir que cette expression est supérieure strictement à $(x+2)^2 + (y+2)^2$.

   Ensuite si $k > 0$ alors nous développons complètement $\mu_{\text{it}}(x,k) + \mu_{\text{it}}(y,k)$: $(\mu_{\text{it}}(x,k-1) + x + 2)^2 + (\mu_{\text{it}}(y,k-1) + y + 2)^2 = \mu_{\text{it}}(x,k-1)^2 + x^2 + 4 + 4x + 4\mu_{\text{it}}(x,k-1) + 2x\mu_{\text{it}}(x,k-1) + \mu_{\text{it}}(y,k-1)^2 + y^2 + 4 + 4y + 4\mu_{\text{it}}(y,k-1) + 2y\mu_{\text{it}}(y,k-1)$.

   Par ailleurs : $\mu_{\text{it}}(x+y,k+1) = (\mu_{\text{it}}(x+y,k) + x + y + 2)^2$, et donc par hypothèse de récurrence : $\mu_{\text{it}}(x+y,k+1) > (\mu_{\text{it}}(x,k-1) + \mu_{\text{it}}(y,k-1) + x + y + 2)^2$. Nous développons cette dernière expression : $\mu_{\text{it}}(x,k-1)^2 + \mu_{\text{it}}(y,k-1)^2 + x^2 + y^2 + 4 + 2xy + 4x + 4y + 2\mu_{\text{it}}(x,k-1)\mu_{\text{it}}(y,k-1) + 2x\mu_{\text{it}}(x,k-1) + 2x\mu_{\text{it}}(y,k-1) + 2y\mu_{\text{it}}(x,k-1) + 2y\mu_{\text{it}}(y,k-1) + 4\mu_{\text{it}}(x,k-1) + 4\mu_{\text{it}}(y,k-1)$.

   En comparant avec l'expression obtenue à partir de $\mu_{\text{it}}(x,k) + \mu_{\text{it}}(y,k)$, l'inégalité revient à : $4 < 2xy + 2\mu_{\text{it}}(x,k-1)\mu_{\text{it}}(y,k-1) + 2y\mu_{\text{it}}(x,k) + 2x\mu_{\text{it}}(y,k)$. D'après le premier résultat, $\mu_{\text{it}}(x,k-1) \geq 4$ et $\mu_{\text{it}}(x,k-1) \geq 4$, donc l'expression de droite est supérieure à 32 et donc à 4.

8. Conséquence triviale de la définition de $\mu_{\text{l}}(k+1)$.

9. Par récurrence sur q; conséquence des résultats 5 et 7.

□

Nous pouvons maintenant définir la mesure. Soit une interprétation $\mathfrak{I}$ et un nœud ν d'un tableau t. On définit alors $\mu_{\mathfrak{I}}(\nu,t) \stackrel{\text{def}}{=} (\mu^1_{\mathfrak{I}}(\nu), \mu^2_{\mathfrak{I}}(s(\nu)), \mu^3_{\mathfrak{I}}(\nu))$, ordonné en utilisant l'extension lexicographique de l'ordre habituel sur les nombres naturels et où $\mu^1_{\mathfrak{I}}(\nu)$, $\mu^2_{\mathfrak{I}}(s(\nu))$, $\mu^3_{\mathfrak{I}}(\nu)$ sont définis par :

1. $\mu^1_{\mathfrak{I}}(\nu)$ est le nombre d'atomes qui apparaissent dans $\langle s(\nu)\rangle_{\mathfrak{I}_{\text{arith}}}$ mais pas dans $\langle \bigwedge_{l \in L(\nu)} l\rangle_{\mathfrak{I}_{\text{arith}}}$;

2. $\mu^2_{\mathfrak{I}}(s(\nu))$ est défini par induction sur la structure de $m_{S(\nu)}$:

   - $\mu^2_{\mathfrak{I}}(\top) \stackrel{\text{def}}{=} \mu^2_{\mathfrak{I}}(\bot) \stackrel{\text{def}}{=} 1$
   - $\mu^2_{\mathfrak{I}}(\neg m) \stackrel{\text{def}}{=} \mu^2_{\mathfrak{I}}(m) + 1$
   - $\mu^2_{\mathfrak{I}}(m_1 \amalg m_2) \stackrel{\text{def}}{=} \mu^2_{\mathfrak{I}}(m_1) + \mu^2_{\mathfrak{I}}(m_2)$, où $\amalg \in \{\wedge, \vee\}$
   - $\mu^2_{\mathfrak{I}}(\amalg_{i|\delta}\, m) \stackrel{\text{def}}{=} \mu_{\text{it}}(\sum_{I \in E} \mu^2_{\mathfrak{I}}(m[I/i]), n_{\text{it}} + n_\vee + n_{\text{lin}})$ où :
     - $E \stackrel{\text{def}}{=} \{I \in \mathbb{Z} \mid \mathfrak{I}_{\text{arith}} \circledcirc \{i \mapsto I\} \models \delta\}$ (E est fini car δ encadre i);
     - n_{it} est le nombre d'itérations $\amalg_{i'|\delta'}\, m'$ tel que Deep splitting peut être appliquée à $\amalg_{i|\delta}(m[\amalg_{i'|\delta'}\, m'])$;

- $n_\vee$ est défini comme :

$$\sum\{(\text{nb. de } \vee \text{ dans } \delta') \times (\#\delta' + 1) \mid \Pi_{i'|\delta'}\, m' \text{ apparaît dans } \Pi_{i|\delta}\, m\}$$

  en supposant que tout domaine apparaissant dans $\Pi_{i|\delta}\, m$ est sous forme normale disjonctive (sans quantificateur)[6] et où $\#\delta'$ est la taille de δ'. Le but de cette définition est que $n_\vee$ diminue avec une application de la règle Élimination des disjonctions de $\mathrm{R_{syn}}$ – section 3.44;
- Pour n_{lin}, nous considérons une forme normale de $\Pi_{i|\delta}\, m$ par toutes les règles de $\mathrm{R_{syn}}$, sauf Linéarisation[7]. n_{lin} est alors le nombre d'applications possibles de la règle Linéarisation sur cette forme normale.

- $\mu^2_{\mathfrak{J}}(p_{e_1,\ldots,e_k}) \stackrel{\text{def}}{=} \mu_{\text{l}}(n_{\text{l}})$ où n_{l} est le nombre de littéraux de la forme $p_{f_1,\ldots,f_k} \in L(\nu)$ tels que Expansion peut être appliquée à $p_{e_1,\ldots,e_k}$ avec le littéral $p_{f_1,\ldots,f_k}$.

3. $\mu^3_{\mathfrak{J}}(\nu)$ est le nombre d'itérations pour lesquelles nous pouvons appliquer Constraint Splitting.

Nous montrons maintenant que cette mesure décroît bien à l'application de chaque règle de Dpll* (lemme 8.11). Pour parvenir à ce résultat nous montrons d'abord que $\mathrm{R_{syn}}$ préserve la mesure (lemme 8.10). Le lemme suivant est utile à la démonstration du lemme 8.10.

Lemma 8.9. *Soit une itération $\Pi_{i|\delta}\, m$ t.q. $\delta \equiv \delta_1 \vee \delta_2$. Si δ_1 et δ_2 sont disjoints (définition 3.38) alors :*

$$\sum_{I \in E} \mu^2_{\mathfrak{J}}(m[I/i]) = \sum_{I \in E_1} \mu^2_{\mathfrak{J}}(m[I/i]) + \sum_{I \in E_2} \mu^2_{\mathfrak{J}}(m[I/i])$$

où :

- $E = \{I \in \mathbb{Z} \mid \mathfrak{I}_{\text{arith}} \circledcirc \{i \mapsto I\} \models \delta_1 \vee \delta_2\}$
- $E_1 = \{I \in \mathbb{Z} \mid \mathfrak{I}_{\text{arith}} \circledcirc \{i \mapsto I\} \models \delta_1\}$
- $E_2 = \{I \in \mathbb{Z} \mid \mathfrak{I}_{\text{arith}} \circledcirc \{i \mapsto I\} \models \delta_2\}$

Preuve. En effet pour tout environnement ρ tel que $\rho \models \delta_1 \vee \delta_2$, nous avons soit $\rho \models \delta_1$, auquel cas $\rho \not\models \delta_2$ car $\delta_1 \wedge \delta_2$ est insatisfaisable, soit $\rho \models \delta_2$ auquel cas $\rho \not\models \delta_1$ pour la même raison. Donc E_1 et E_2 sont disjoints. De plus il est clair que $E = E_1 \cup E_2$. D'où le résultat. □

Le lemme suivant assure que la mesure n'augmente pas lorsque nous utilisons la procédure auxiliaire $\mathrm{R_{syn}}$ pour encadrer syntaxiquement un schéma.

Lemma 8.10. *Soit un motif m et un motif m' obtenu à partir de m en appliquant l'une des règles de $\mathrm{R_{syn}}$, alors $\mu^2_{\mathfrak{J}}(m') \leq \mu^2_{\mathfrak{J}}(m)$.*

Preuve. Forme normale arithmétique modifie uniquement le domaine des itérations et ce domaine est équivalent au domaine avant réécriture, donc la mesure ne change pas.

Pour Élimination des disjonctions le domaine du membre gauche de la règle est équivalent à la disjonction des domaines du membre droit (c.-à-d. $(l_1 \wedge \ldots \wedge l_k) \vee \delta \vee \phi \equiv [l_1 \wedge \ldots \wedge l_k] \vee [(\neg l_1 \wedge \delta) \vee \phi] \vee \ldots \vee [(l_1 \wedge \ldots \wedge l_{k-1} \wedge \neg l_k \wedge \delta) \vee \phi]$). De plus tous ces domaines sont deux à deux disjoints. On peut donc appliquer le lemme 8.9, donc nous savons que la somme dans la mesure du membre gauche est égale à la somme des sommes dans toutes les mesures du membre droit. Par ailleurs, n_{it} a la même valeur dans l'itération du membre gauche et dans chaque itération du membre droit (en fait n_{it} peut même décroitre car les nouveaux domaines peuvent changer le contexte et donc rendre Deep splitting inapplicable sur certaines itérations). En revanche comme nous avons appliqué la règle Élimination des disjonctions une fois, il y a une disjonction en moins dans le domaine de chaque nouvelle itération. Ainsi $n_\vee$ diminue. Plus précisément $n_\vee$ diminue de $\#\delta + 1$. D'autre part n_{lin} ne peut que rester constant, puisque sa valeur est calculée après application de la règle Élimination des

[6] Il peut y avoir plusieurs formes normales disjonctives (mais toujours une quantité finie), nous considérons celle qui maximise la mesure.

[7] Une fois encore il peut y avoir plusieurs formes normales, nous considérons celle qui maximise la mesure.

DISJONCTIONS. Donc si $\Pi_{i|(l_1\wedge\ldots\wedge l_k)\vee\delta\vee\phi}\, m$ a pour mesure $\mu_{\text{it}}(\sum_{I\in E}\mu^2_{\mathfrak{I}}(m[I/i]), n_{\text{it}}+n_\vee+n_{\text{lin}})$, alors $(\Pi_{i|l_1\wedge\ldots\wedge l_k}\, m)\,\Pi\,(\Pi_{i|(\neg l_1\wedge\delta)\vee\phi}\, m)\,\Pi\ldots\Pi\,(\Pi_{i|(l_1\wedge\ldots\wedge l_{k-1}\wedge\neg l_k\wedge\delta)\vee\phi}\, m)$ a pour mesure :

$$\begin{aligned}
&\mu_{\text{it}}(\sum_{I\in E_0}\mu^2_{\mathfrak{I}}(m[I/i]), n^0_{\text{it}}+n_\vee-\#\delta-1+n^0_{\text{lin}})\\
+&\mu_{\text{it}}(\sum_{I\in E_1}\mu^2_{\mathfrak{I}}(m[I/i]), n^1_{\text{it}}+n_\vee-\#\delta-1+n^1_{\text{lin}})\\
+&\ldots\\
+&\mu_{\text{it}}(\sum_{I\in E_k}\mu^2_{\mathfrak{I}}(m[I/i]), n^k_{\text{it}}+n_\vee-\#\delta-1+n^k_{\text{lin}})
\end{aligned}$$

où E, E_0, E_1, ... sont définis comme précédemment pour chaque itération, $n^0_{\text{it}}\leq n_{\text{it}}$, $n^1_{\text{it}}\leq n_{\text{it}}$, ..., $n^0_{\text{lin}}\leq n_{\text{lin}}$, $n^1_{\text{lin}}\leq n_{\text{lin}}$, ... et la somme sur E est égale à la somme des sommes sur E_0, E_1, ... Comme le nombre de termes dans cette somme est $k+1$, il est inférieur à $\#\delta+1$. On peut donc conclure avec le résultat 9 de la proposition 8.8.

Pour LINÉARISATION, l'itération est divisée en deux itérations de domaines disjoints, nous pouvons donc à nouveau appliquer 8.9. n_{it} et $n_\vee$ sont, au pire, constant, mais n_{lin} diminue strictement, ce qui permet de conclure en utilisant le résultat 7 de la proposition 8.8. □

Enfin, nous démontrons que les règles de DPLL* diminuent bien strictement la valeur de la mesure :

Lemma 8.11. *Soient:*

- *t un tableau satisfaisable,*
- *$\mathfrak{I}$ un modèle de t,*
- *ν une feuille d'un tableau t, telle que $\mathfrak{I}\models\nu$,*
- *et t' un tableau obtenu par application d'un règle de* DPLL* *à ν.*

Alors pour chaque fils ν' de ν dans t' tel que $\mathfrak{I}\models\nu'$, nous avons $\mu_{\mathfrak{I}}(\nu',t')<\mu_{\mathfrak{I}}(\nu,t)$.

Preuve. Par inspection des règles, nous montrons que :

Règle	$\mu^1_{\mathfrak{I}}$	$\mu^2_{\mathfrak{I}}$	$\mu^3_{\mathfrak{I}}$
PROPOSITIONAL SPLITTING	$<$		
CONSTRAINT SPLITTING	$\leq$	$\leq$	$<$
REWRITING	$\leq$	$<$	

$\leq$ (resp. $<$) signifie que la mesure ne décroit pas (resp. décroit strictement) par application de la règle correspondante.

Considérons d'abord $\mu^1_{\mathfrak{I}}$. Quand PROPOSITIONAL SPLITTING est appliquée, nous ajoutons soit p_e, soit $\neg p_e$ à $L(\nu)$. Donc $\langle p_e\rangle_{\mathfrak{I}_{\text{arith}}}$ ou $\langle\neg p_e\rangle_{\mathfrak{I}_{\text{arith}}}$ apparaît positivement dans $\langle\bigwedge_{l\in L(\nu)} l\rangle_{\mathfrak{I}_{\text{arith}}}$ après application de la règle. On sait que ce littéral n'y apparaissait pas avant d'après les conditions d'application de la règle. Par ailleurs, aussi par ces conditions d'application, soit p_e, soit $\neg p_e$ apparaît toujours dans $S(\nu)$, donc soit $\langle p_e\rangle_{\mathfrak{I}_{\text{arith}}}$, soit $\langle\neg p_e\rangle_{\mathfrak{I}_{\text{arith}}}$ apparaît positivement dans $\langle s(\nu)\rangle_{\mathfrak{I}_{\text{arith}}}$. Il est donc clair que $\mu^1_{\mathfrak{I}}$ décroit strictement. Enfin il est facile de voir que les autres règles n'ajoutent pas d'atome à $\langle s(\nu)\rangle_{\mathfrak{I}_{\text{arith}}}$, au pire elles dupliquent des atomes existants, donc la mesure n'augmente pas.

Pour $\mu^2_{\mathfrak{I}}$, nous détaillons toutes les règles de réécriture :

- ALGEBRAIC SIMPLIFICATIONS : évident pour chaque règle. Notons que les deux premières propriétés de la proposition 8.8 sont utilisées dans les règles mettant en jeu des itérations.
- UNFOLDING : c'est une conséquence du résultat 6 de la proposition 8.8 (nous pouvons appliquer ce résultat car $\mu^2_{\mathfrak{I}}(m)\geq 1$ pour tout motif m). Une fois que la règle est appliquée, il faut encore appliquer R_{syn} pour ramener le motif résultant à un motif syntaxiquement encadré. D'après le lemme 8.10 nous savons que cette opération n'augmente pas la mesure.

- Deep splitting : c'est une conséquence du résultat 7 de la proposition 8.8. Il est clair que n_{it} décroit dans chaque itération, ce qui permet d'appliquer le résultat.[8] Ensuite il faut appliquer R_{syn} pour que le schéma soit syntaxiquement aligné, ce qui ne peut pas augmenter la mesure comme nous l'avons déjà vu.

- Expansion : c'est une conséquence du résultat 8 de la proposition 8.8.[9] Une fois encore il faut appliquer R_{syn} pour que le schéma soit syntaxiquement aligné, ce qui ne peut pas augmenter la mesure.

Enfin Constraint Splitting ne fait pas croître $\mu^2_{\mathfrak{I}}$ parce qu'elle ne modifie pas le motif, elle ne modifie pas le domaine des itérations (donc $n_\vee$ et n_{lin} sont inchangés) et la contrainte ajoutée dans chaque branche ne peut que restreindre les possibilités d'application de Deep splitting, donc n_{it} n'augmente pas non plus.

Enfin il est évident qu'une application de Constraint Splitting diminue strictement $\mu^3_{\mathfrak{I}}$: le nombre d'itérations auxquelles peut s'appliquer Constraint Splitting diminue strictement dans une branche comme dans l'autre. □

À partir de maintenant la preuve est exactement la même que pour Stab (section 4).

Proposition 8.12. *Si une feuille est satisfaisable et irréductible alors elle n'est pas close.*

Preuve. Si elle était close soit elle contiendrait $\bot$, soit l'ensemble de ses contraintes seraient insatisfaisable. Dans les deux cas la feuille elle-même serait insatisfaisable. □

Theorem 8.13 (Complétude pour la satisfaisabilité)**.** *Soient :*

- t_0 *un tableau satisfaisable,*
- $\mathfrak{I}$ *un modèle de* t_0*,*
- *et* $(t_i)_{i\in I}$ *une dérivation équitable.*

Il existe $k \in I$ *et une feuille* ν_k *de* t_k *telle que* ν_k *est irréductible et non close.*

Preuve. D'après le lemme 8.5, pour tout $i \in I$, t_i contient une feuille ϕ_i telle que $\mathfrak{I} \models \phi_i$. Soit k tel que $\mu_{\mathfrak{I}}(\nu_k, t_k)$ soit minimale (k existe car la mesure est bien fondée). Il est clair d'après le lemme précédent qu'aucune règle ne s'applique à ν_k dans la dérivation (autrement cela entrerait en contradiction avec l'hypothèse de minimalité). Comme la dérivation est équitable, soit ν_k est irréductible, soit il existe une autre feuille qui est irréductible et non close. Si ν_k est irréductible, alors elle ne peut être close d'après la proposition précédente. □

[8] n_{it} a été défini précisément pour pouvoir gérer ce cas.
[9] μ_{l} et n_{it} ont été définis précisément pour pouvoir gérer ce cas.

CHAPTER 9

Regularly Nested Schemata

Nous introduisons maintenant une nouvelle classe de schémas qui est complète pour la réfutation, et donc décidable : les *schémas réguliers imbriqués*. Comme leur nom l'indique, ils sont très proches des schémas réguliers, mais autorisent l'imbrication d'itérations. Pour démontrer la décidabilité, nous prouvons que DPLL* termine sur ces schémas. Cette propriété n'est pas satisfaite par la procédure STAB introduite au chapitre 4 ce qui démontre l'intérêt de la procédure de preuve DPLL* introduite au chapitre précédent.

Ce chapitre est essentiellement consacré à la démonstration de terminaison de DPLL* sur les schémas réguliers imbriqués, qui a été présentée dans [ACP10a, ACP10b].

9.1 Definition

Definition 9.1 (Schéma régulier imbriqué)**.** *Un schéma est* régulier imbriqué *ssi il n'a qu'un paramètre n, est monadique, à translation bornée (déf. 5.18) et est aligné sur $[q_1; n - q_2]$ (déf. 6.1), pour des entiers $q_1, q_2 \in \mathbb{Z}$ fixés.*

En dehors de la contrainte que le schéma doit être plat, nous retrouvons toutes les propriétés qui définissent un schéma régulier, sauf pour la propriété de translation bornée qui est plus forte que celle de propagation bornée (déf. 6.1) : cette dernière autorise des expressions contenant des multiplications par une constante en dehors des itérations. Mais tout schéma régulier *et à translation bornée* est régulier imbriqué. Par exemple le littéral p_{5n} est un schéma régulier mais il n'est pas régulier imbriqué.

Example 9.2. *Le schéma $\left(\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} p_i \Rightarrow q_j\right) \wedge \bigwedge_{i=1}^{n} \neg q_i \wedge \bigvee_{i=1}^{n} p_i \,\&\, n \geq 1$ est régulier imbriqué. Pour rappel nous avons vu au chapitre 8.1 que* STAB *ne terminait pas lorsqu'il prenait ce schéma en entrée. Le schéma formalisant le principe des pigeonniers (exemple 3.2) n'est pas régulier imbriqué car il n'est pas monadique (il n'est pas aligné non plus mais ce problème est facile à contourner).*

Peu d'exemples concrets sont *intrinsèquement* réguliers imbriqués, c.-à-d réguliers imbriqués mais pas réguliers, et pour cause : les schémas réguliers imbriqués peuvent être transformés en schémas réguliers équivalents [ACP10d][1] (ce qui nous permet d'obtenir une autre preuve, indirecte, de complétude). Toutefois cette transformation est exponentielle et nous pensons que la technique de preuve (directe) utilisée dans ce chapitre pour démontrer la complétude des schémas réguliers imbriqués est plus générale que la méthode par traduction. Nous pourrions ainsi, à terme, obtenir des résultats de décidabilité sur des classes plus expressives, p.ex. contenant plusieurs variables dans les bornes des itérations. Mais comme nous allons le voir la preuve directe est fastidieuse, il faudrait donc que le gain en expressivité soit vraiment important pour justifier cet effort.

En fait, l'intérêt de ce chapitre réside principalement dans l'originalité de la preuve qui, comme expliqué par la suite, se fait par induction sur la structure du schéma d'entrée. Plus précisément, nous

[1] Nous ne présentons pas ce travail ici pour ne pas alourdir la thèse. Nous l'évoquons rapidement en section 9.A.

montrons que chaque sous-schéma va engendrer un ensemble fini de schémas, et démontrons le résultat pour le schéma lui-même par induction. Pour pouvoir mettre ce raisonnement en place, il faut introduire un certain nombre de restrictions sur la façon dont nous appliquons DPLL* (sections 9.2 et 9.3.1). Ce sont ces restrictions qui rendent la preuve fastidieuse.

9.2 Spécialisation de DPLL* aux schémas réguliers imbriqués

Nous donnons tout d'abord une stratégie pour DPLL* dédiée aux schémas réguliers et nous démontrons les propriétés qui font son intérêt dans le cadre du résultat que nous souhaitons démontrer.

Tout d'abord, nous caractérisons un type particulier d'application de la règle CONSTRAINT SPLITTING : soit $(m_s[\Pi_{i=1}^n m] \,\&\, c_s, L)$ la prémisse de CONSTRAINT SPLITTING, si $\delta \equiv e \leq i \wedge i \leq f$, où e et f ne contiennent pas i, alors le CONSTRAINT SPLITTING est appelé un CONSTRAINT SPLITTING encadré.

Nous définissons ensuite la stratégie τ comme la boucle suivante :

1. CONSTRAINT SPLITTING encadré est appliquée jusqu'à irréductibilité.
2. Puis toutes les règles, exceptée UNFOLDING, sont appliquées jusqu'à irréductibilité, avec la restriction que EXPANSION ne peut réécrire p_e[2] que si e ne contient pas de variable liée et que PROPOSITIONAL SPLITTING n'est appliquée que si p_e apparaît librement dans s (déf. 3.16).
3. Enfin seul UNFOLDING est appliquée jusqu'à irréductibilité, avec la restriction que l'expression e choisie arbitrairement est la borne supérieure du domaine de l'itération réécrite, à condition que la seule variable libre de cette borne soit n[3] (ainsi la règle UNFOLDING est très proche des règles d'itération de STAB). Puis nous retournons en 1.

Pour la règle de bouclage nous utilisons l'égalité à un décalage près avec l'extension du littéral pur et de la contrainte équivalente.

Example 9.3. *Nous détaillons l'application de la stratégie au schéma :*

$$\left(\bigwedge_{i=1}^{n}\bigvee_{j=1}^{n} p_i \Rightarrow q_j\right) \wedge \bigwedge_{i=1}^{n} \neg q_i \wedge \bigvee_{i=1}^{n} p_i$$

dont nous avons vu en section 8.1 qu'il ne terminait pas avec STAB.

1. *L'application de l'étape 1 est donnée en figure 9.1 . En (1),* CONSTRAINT SPLITTING encadré *ne peut pas s'appliquer une deuxième fois car la contrainte que $c_s \wedge \forall i \neg\delta$ est satisfaisable n'est plus vraie. En (2), elle ne peut pas s'appliquer simplement car il n'y a plus d'itération. Il est facile de généraliser cet exemple pour montrer que l'étape 1 termine quel que soit le schéma (lemme 9.8).*
2. *Le nœud (2) à la fin de l'étape 1 ne contient ni littéraux ni itérations. Par conséquent, seule* ALGEBRAIC SIMPLIFICATIONS *peut s'appliquer et la branche est fermée, voir figure 9.2 . Toutes les itérations du nœud (1) sont alignées donc* CONSTRAINT SPLITTING NON ENCADRÉ *ne peut pas s'appliquer.* ALGEBRAIC SIMPLIFICATIONS *ne peut pas s'appliquer non plus.* DEEP SPLITTING *ne peut pas s'appliquer car aucun domaine d'une itération ne contient de variable liée par une autre itération (ce qui sera généralisé dans la proposition 9.5). Comme l'ensemble de littéraux est vide,* EXPANSION *ne peut pas s'appliquer. Enfin, aucun littéral n'apparaît* librement *dans le schéma (déf. 3.16), par conséquent* PROPOSITIONAL SPLITTING *ne peut pas non plus s'appliquer (si le schéma était, p.ex. $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$, alors la règle pourrait s'appliquer car p_0 apparaît librement). Ainsi,* aucune règle n'est appliquée *et nous passons directement à l'étape 3.*
3. *L'application de l'étape 3 au nœud (1) est donnée en figure 9.3 . Notons que* UNFOLDING *peut s'appliquer à chaque fois car la contrainte ajoutée lors de l'étape 1 assure que les itérations ne sont pas vides. Nous pouvons donc extraire le dernier rang. Si nous ne suivions pas la stratégie,*

[2] Notons que p n'a qu'un indice car les schémas réguliers imbriqués sont monadiques.
[3] Cette dernière restriction sert uniquement à simplifier les preuves mais est inutile à la terminaison.

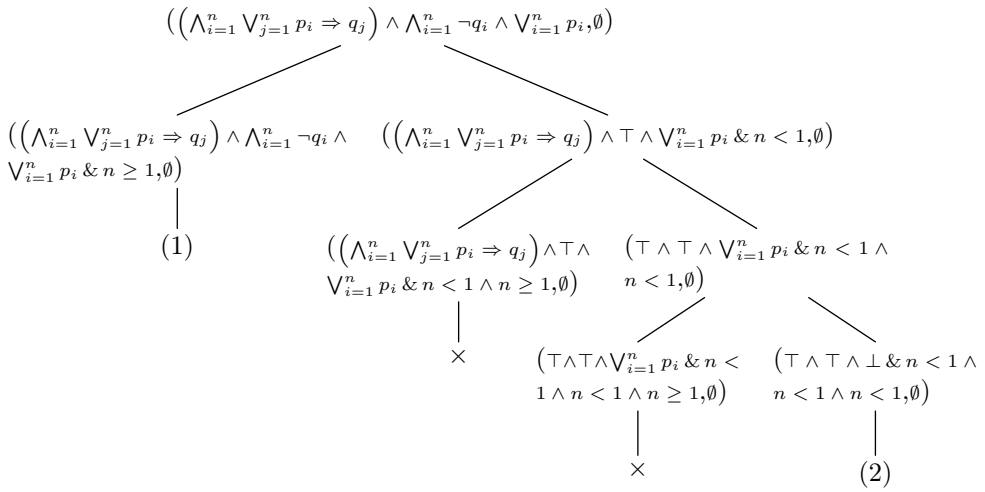


Figure 9.1: Application de l'étape 1 à $\left(\bigwedge_{i=1}^{n}\bigvee_{j=1}^{n} p_i \Rightarrow q_j\right) \wedge \bigwedge_{i=1}^{n} \neg q_i \wedge \bigvee_{i=1}^{n} p_i$ (chaque règle appliquée est un Constraint Splitting encadré).

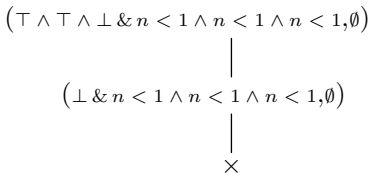


Figure 9.2: Application de l'étape 2 au nœud (2) de la figure 9.1 (chaque règle appliquée est une Algebraic simplifications).

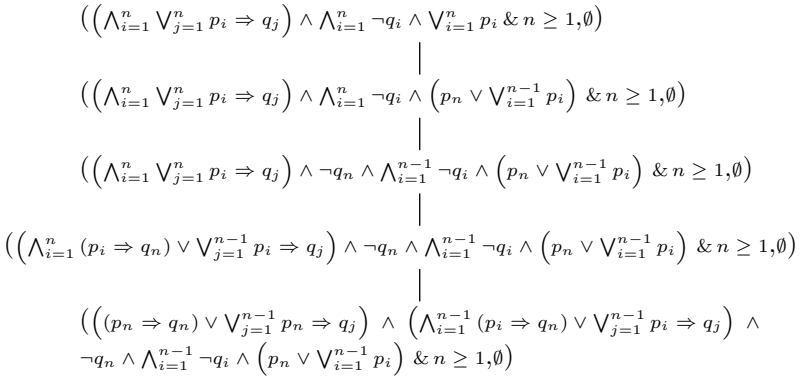


Figure 9.3: Application de l'étape 3 au nœud (1) de la figure 9.1 (chaque règle appliquée est un Unfolding).

nous pourrions aussi extraire le premier rang. Par exemple, en figure 9.3, nous obtiendrions, au deuxième nœud, le motif suivant :

$$\left(\bigwedge_{i=1}^{n}\bigvee_{j=1}^{n} p_i \Rightarrow q_j\right) \wedge \bigwedge_{i=1}^{n} \neg q_i \wedge \left(p_1 \vee \bigvee_{i=2}^{n} p_i\right)$$

au lieu de :

$$\left(\bigwedge_{i=1}^{n}\bigvee_{j=1}^{n} p_i \Rightarrow q_j\right) \wedge \bigwedge_{i=1}^{n} \neg q_i \wedge \left(p_n \vee \bigvee_{i=1}^{n-1} p_i\right)$$

Mais τ impose que ce soit la borne supérieure qui soit utilisée.

Puis nous revenons à l'étape 1 :

1. *Nous ajoutons soit la contrainte $n-1 \geq 1$ (nœud (3)), soit la contrainte $n-1 < 1$ (nœud (4)). Dans le premier cas, et comme au cycle précédent, l'étape 1 est immédiatement terminée car* Constraint Splitting encadré *ne peut plus être appliquée. Dans le deuxième cas, après plusieurs applications de la règle, toutes les itérations sont remplacées par leur élément neutre, comme au cycle précédent. Nous obtenons alors :*

$$((p_n \Rightarrow q_n) \vee \bot) \wedge \top \wedge \neg q_n \wedge \top \wedge (p_n \vee \bot) \,\&\, n \geq 1 \wedge n - 1 < 1$$

2. *L'application de l'étape 2 au nœud (4) est donnée en figure 9.4 (nous avons réécrit la contrainte en $n = 1$ pour des raisons de lisibilité). Comme ce nœud ne contient pas d'itération ni* Constraint Splitting non encadré*, ni* Deep splitting *ne peuvent s'appliquer. Nous pouvons donc appliquer soit* Algebraic simplifications*, soit* Propositional Splitting *(*Expansion *ne peut pas s'appliquer car l'ensemble de littéraux est vide). Pour des raisons de lisibilité nous appliquons* Algebraic simplifications *en premier afin de simplifier la formule. Puis nous appliquons* Propositional Splitting *avec p_n : nous pouvons bien appliquer cette règle car p_n apparaît librement dans le schéma (déf. 3.16). Puis, la seule règle que nous pouvons appliquer est* Expansion *qui ajoute une itération autour de p_n. Notons que ceci est possible car p_n ne contient pas de variable liée (autrement ce n'aurait pas été conforme à la définition de τ). Nous pouvons maintenant appliquer soit* Constraint Splitting non encadré*, soit* Algebraic simplifications*. Il est plus direct d'appliquer* Algebraic simplifications *(d'ailleurs cette remarque est généralisable) qui n'introduit pas de branchement. Nous appliquons cette règle jusqu'à ce que la formule soit complètement simplifiée. Dans la branche la plus à droite, nous obtenons $\bot$, donc la branche est fermée. Dans les branches de gauche, nous faisons un nouveau branchement sur la valeur de q_n. Suivant le même processus, toutes les branches sont fermées.*

   En fait, nous pouvons constater que l'étape 2 correspond essentiellement à l'application de Dpll *: un littéral est sélectionné et suivant sa valeur des simplifications peuvent être opérées. Mais il n'y a aucun travail sur les itérations. Nous ne détaillons pas le nœud (3) pour lequel la même démarche va être appliquée. Nous ne considérons qu'une branche, celle où l'ensemble de littéraux est $\{\neg p_n, \neg q_n\}$, le résultat est donné en figure 9.5 .*

   Hormis pour la première itération, nous retrouvons presque le schéma initial à un décalage près :

   - *La contrainte est différente mais par extension de la contrainte équivalente, nous pouvons considérer que c'est juste $n - 1 \geq 1$ (qui subsume $n \geq 1$).*
   - *Il faut aussi tenir compte du fait que le bouclage ne s'effectue pas seulement sur le schéma mais sur $sL(\nu)$ (d'après la définition 8.1) :*

$$sL(\nu) \stackrel{def}{=} m_{s(\nu)} \wedge \left(\bigwedge_{l \in L(\nu)} l\right) \,\&\, c_{s(\nu)}$$

     c.-à-d. il faut tenir compte des littéraux dans l'ensemble de littéraux. Ici, la conjonction $\neg p_n \wedge \neg q_n$ n'est pas égal à $\top$ à un décalage près ($\top$ car le schéma initial avait $\emptyset$ pour ensemble de littéraux; cet ensemble devient $\top$ dans $sL(\nu)$). Mais p_n et q_n sont purs dans le schéma. Nous pouvons donc considérer que l'ensemble est $\emptyset$, et il y a bien bouclage.

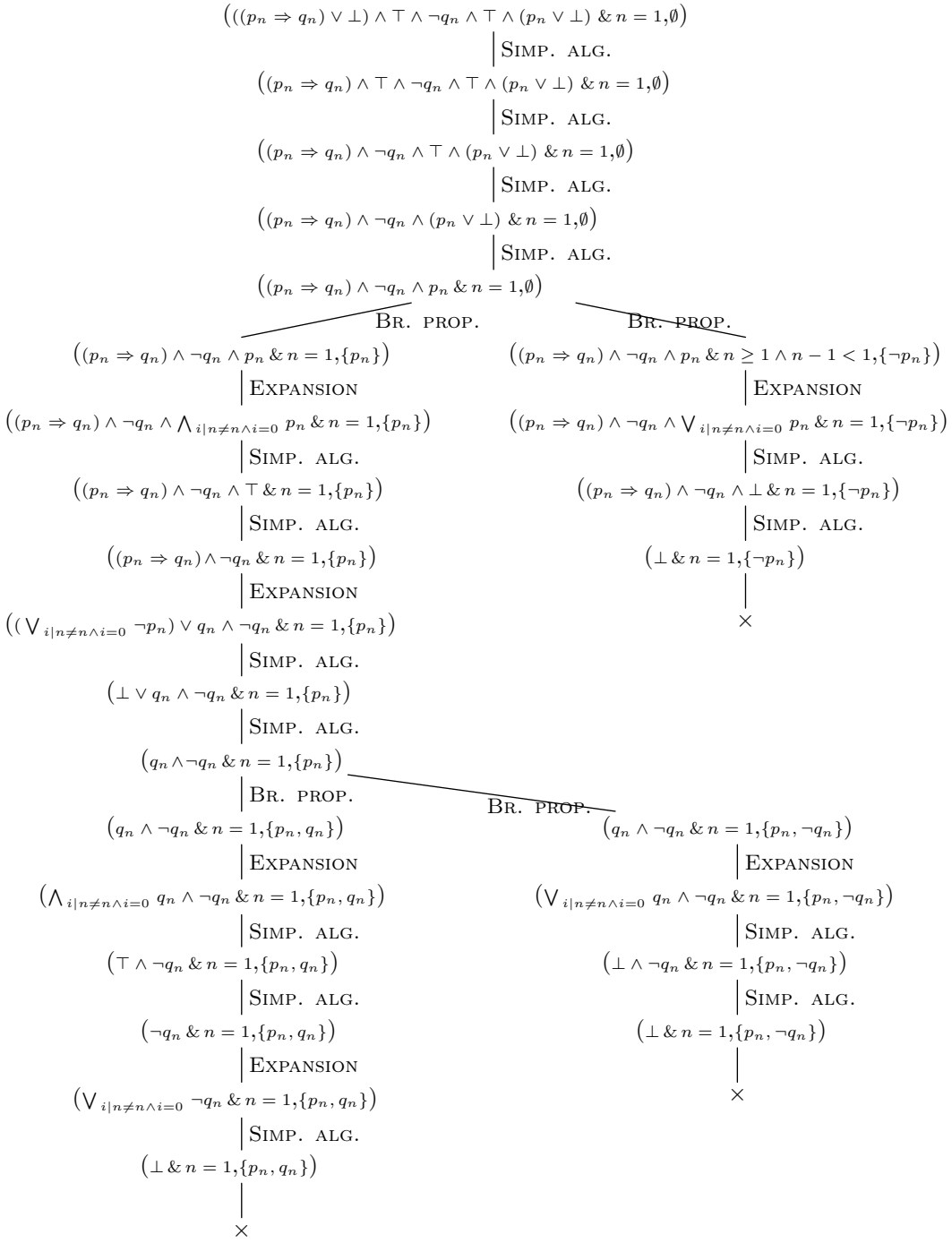


Figure 9.4: Application de l'étape 2 au nœud (4).

$$\Big(\Big((p_n \Rightarrow q_n) \vee \bigvee_{j=1}^{n-1} p_n \Rightarrow q_j\Big) \wedge \Big(\bigwedge_{i=1}^{n-1} (p_i \Rightarrow q_n) \vee \bigvee_{j=1}^{n-1} p_i \Rightarrow q_j\Big) \wedge \neg q_n \wedge \bigwedge_{i=1}^{n-1} \neg q_i \wedge \Big(p_n \vee \bigvee_{i=1}^{n-1} p_i\Big) \;\&\; n \geq 1 \wedge n-1 \geq 1, \{\neg p_n, \neg q_n\}\Big)$$

$$\Big|\ \ldots$$

$$\Big(\Big((\top \vee \bot) \vee \bigvee_{j=1}^{n-1} \top \vee q_j\Big) \wedge \Big(\bigwedge_{i=1}^{n-1} (p_i \vee \bot) \vee \bigvee_{j=1}^{n-1} p_i \Rightarrow q_j\Big) \wedge \top \wedge \bigwedge_{i=1}^{n-1} \neg q_i \wedge \Big(\bot \vee \bigvee_{i=1}^{n-1} p_i\Big) \;\&\; n \geq 1 \wedge n-1 \geq 1, \{\neg p_n, \neg q_n\}\Big)$$

$$\Big|\ \textsc{Algebraic simplifications}^*$$

$$\Big(\Big(\bigwedge_{i=1}^{n-1} p_i \vee \bigvee_{j=1}^{n-1} p_i \Rightarrow q_j\Big) \wedge \bigwedge_{i=1}^{n-1} \neg q_i \wedge \Big(\bigvee_{i=1}^{n-1} p_i\Big) \;\&\; n \geq 1 \wedge n-1 \geq 1, \{\neg p_n, \neg q_n\}\Big)$$

Figure 9.5: Application (partielle) de l'étape 2 au nœud (5).

Le seul problème est donc cette première itération $\bigwedge_{i=1}^{n-1} p_i$. En fait le bouclage aura lieu après encore un cycle et sur ce schéma plutôt que le schéma initial. En réalité, ce schéma correspond à un invariant. Ceci nous permet de mettre en lumière un intérêt des preuves cycliques par rapport au preuves classiques par induction : nous n'avons pas besoin de trouver d'invariant avant la preuve, c'est en fait la preuve elle-même qui va construire l'invariant.

(fin de l'exemple 9.3)

Note. Le but de la définition de τ est principalement d'obtenir les propositions 9.12 et 9.15, qui permettent de "structurer" les dérivations de façon à assurer un bouclage. En particulier, les itérations ne restent pas alignées au fur et à mesure de la procédure; en revanche, elles sont toujours alignées *à la fin de l'étape 2* (ce qui correspond à la proposition 9.12). La proposition 9.15 assure que chaque étape prise indépendamment termine.

L'intuition derrière τ est la suivante (ce que nous verrons en détail dans les démonstrations) :

1. Nous assurons que toutes les itérations vont pouvoir être dépliées;
2. Nous appliquons DPLL (et pas DPLL*) sur les littéraux apparaissant en dehors des itérations : en effet nous appliquons, entre autres, PROPOSITIONAL SPLITTING et EXPANSION ce qui a pour effet de donner une valeur à chaque littéral apparaissant librement dans le motif et de propager cette valeur dans tous les littéraux. Puis ALGEBRAIC SIMPLIFICATIONS s'applique pour simplifier le schéma obtenu.

   Note. Comme nous devons comparer certains littéraux qui ne sont pas tout le temps égaux, p.ex. p_1 et p_n, il est nécessaire d'appliquer CONSTRAINT SPLITTING NON ENCADRÉ pour faire la distinction entre le cas où ces littéraux sont égaux et celui où ils ne le sont pas (p.ex. $n = 1$ et $n \neq 1$).

   Si le schéma ne contenait aucune itération, alors il vaut soit $\top$, soit $\bot$ à la fin de cette étape et le développement de la branche est fini.
3. Nous déroulons les itérations. Il est important d'effectuer cette étape de façon indépendante, pour s'assurer que toutes les itérations sont alignées (proposition 9.12).

Proposition 9.4. *τ préserve la complétude pour la satisfaisabilité.*

Preuve (esquisse). Évidemment, la mesure définie pour prouver la complétude de DPLL* (p.98) décroît toujours et il est facile de vérifier que les restrictions introduites par τ ne peuvent pas mener à une situation où un nœud est irréductible et insatisfaisable sans que la branche soit fermée, c.-à-d. il faut vérifier que le lemme 8.6 est préservé. Il est facile de voir que la preuve est toujours valable. □

Avant de commencer la preuve de terminaison en elle-même, nous remarquons que, avec la stratégie τ, la règle DEEP SPLITTING de DPLL* n'est jamais appliquée :

Proposition 9.5. *Si DPLL* est appliquée avec la stratégie τ sur un schéma régulier imbriqué, alors la règle DEEP SPLITTING n'est jamais appliquée.*

Preuve. En effet cette règle n'est appliquée que si le domaine d'une itération contient une variable liée (par une autre itération). Nous montrons par récurrence sur la longueur de la dérivation que nous ne pouvons jamais avoir cette situation.

C'est évident dans le cas de base car le schéma initial est aligné sur $[q_1; n-q_2]$, donc les seules variables apparaissant dans les domaines sont n et la variable liée par l'itération elle-même.

Nous vérifions ensuite que toutes les règles préservent la propriété que les seules variables apparaissant dans les domaines sont n et la variable liée par l'itération elle-même. C'est trivial pour toutes les règles (car elles n'affectent pas les domaines) exceptées Unfolding et Expansion. Dans le premier cas, l'expression e peut contenir des variables liées; mais dans le contexte de la stratégie τ, c'est impossible car e est justement restreint de sorte à ne pas contenir d'autre variable que n. Dans le deuxième cas, la stratégie impose que Expansion ne peut réécrire p_e que si n est la seule variable apparaissant dans e, ce qui assure que nous avons bien la propriété. □

9.3 Termination

L'idée de la preuve est la suivante : comme pour les schémas réguliers, nous supposons qu'une branche infinie b est construite; puis nous montrons que l'ensemble $\{sL(\nu) \mid \nu \text{ est un nœud de } b\}$ (c.-à-d. l'ensemble des schémas générés tout au long de la branche) est fini à un décalage près (avec les extensions du littéral pur et de la contrainte équivalente); ainsi la règle de bouclage est nécessairement applicable et donc la branche est bloquée, d'où une contradiction avec le fait qu'elle est infinie.

Par définition de sL, l'ensemble $\{sL(\nu) \mid \nu \text{ est un nœud de } b\}$ est égal à :

$$\left\{ m_{s(\nu)} \wedge \left(\bigwedge_{l \in L(\nu)} l \right) \,\&\, c_{s(\nu)} \,\middle|\, \nu \text{ est un nœud de } b \right\}$$

La preuve consiste à montrer indépendamment que

$$\{m_{s(\nu)} \mid \nu \text{ est un nœud de } b\}$$

$$\{\textstyle\bigwedge_{l \in L(\nu)} l \mid \nu \text{ est un nœud de } b\}$$

$$\{c_{s(\nu)} \mid \nu \text{ est un nœud de } b\}$$

sont finis à un décalage près (propositions 9.31, 9.26 et 9.30 respectivement) puis à "recoller" ces résultats grâce au théorème 5.20 (plus particulièrement grâce au corollaire 5.25).

Proposition 9.6. *Pour tout nœud ν de b, m_ν contient des itérations.*

Preuve. En effet, si ce n'est pas le cas τ termine trivialement (Dpll* n'est en fait que Dpll dans ce cas), ce qui est contradictoire avec le fait que b est infinie. □

Par conséquent, dans toute la suite, nous ne considérons que des nœuds contenant des itérations.

9.3.1 Nœuds d'alignement

En fait, l'ensemble $\{sL(\nu) \mid \nu \text{ est un nœud de } b\}$ n'est pas fini à un décalage près : nous devons nous restreindre à un certain type de nœuds pour avoir effectivement cette propriété. Tout comme pour les schémas réguliers (définition 6.9) nous appelons ces nœuds des *nœuds d'alignement*. Nous aurons alors effectivement la propriété : $\{sL(\nu) \mid \nu \text{ est un nœud d'alignement de } b\}$ est fini à un décalage près. Comme nous le verrons, toute branche infinie contient nécessairement une infinité de nœuds d'alignement (proposition 9.15). Donc la règle de bouclage s'applique nécessairement et ferme la branche, ce qui entre en contradiction avec le fait qu'elle est infinie.

Dans cette section, nous définissons les nœuds d'alignement et étudions leurs propriétés. Le but est uniquement de préparer le terrain pour la preuve principale (section 9.3.3). À partir de maintenant, s est un schéma régulier imbriqué de paramètre n aligné sur $[q_1; n - q_2]$ (c.-à-d. que toutes ses itérations ont le même domaine $q_1 \leq i \wedge i \leq n - q_2$ ou une formule équivalente à la variable i près, et où $q_1, q_2 \in \mathbb{Z}$, voir déf. 6.1) et t est un tableau de T-Dpll* dont la racine est étiquetée par $(s, \emptyset)$.

Definition 9.7 (Nœud d'alignement)**.** *Un nœud de t est un* nœud d'alignement *ssi il est irréductible par l'étape 2 de la stratégie τ.*

Nous montrons que tout nœud d'alignement est aligné sur $[q_1; n - q_2 - k]$ pour un certain $k \in \mathbb{N}$ (proposition 9.12). Ce résultat est établi au prix de trois lemmes intermédiaires, correspondant à chacune des trois étapes de la stratégie τ. Les résultats de cette section sont bien illustrés par l'exemple 9.3, en gardant à l'esprit que seule une branche infinie vérifie ces propriétés.

Lemma 9.8 (Étape 1)**.** *Soit η un nœud de b obtenu par application de l'étape 1 à un nœud ν (c.-à-d. jusqu'à irréductibilité). Alors :*

(i) Chaque itération de $m_{s(\eta)}$ apparaît dans $m_{s(\nu)}$

(ii) De plus, s'il existe des expressions e et f telles que ν est aligné sur $[e; f]$, alors $c_{s(\eta)} \models e \leq f$.

Note. Un nœud ν est aligné sur $[e; f]$ ssi $s(\nu)$ l'est.

Preuve. Seule Constraint Splitting s'applique dans l'étape 1. Cette règle ne modifie pas le motif de $s(\nu)$ donc (i) est triviale. De plus si ν est aligné sur $[e; f]$, alors, par définition, toutes ses itérations ont le même domaine $e \leq i \wedge i \leq f$, donc Constraint Splitting ne s'applique qu'une fois, générant deux branches : une dans laquelle $c_{s(\nu)} \models \exists i(e \leq i \wedge i \leq f)$, et donc $c_{s(\nu)} \models e \leq f$, l'autre dans laquelle $c_{s(\nu)} \models \forall i(\neg e \leq i \wedge i \leq f)$, et où toutes les itérations seront donc supprimées à l'étape 2, ce qui est contradictoire avec la proposition 9.6. □

Lemma 9.9 (Étape 2)**.** *Soit ν un nœud de t et η un nœud obtenu par application de l'étape 2 à ν, jusqu'à irréductibilité. Si une itération $\Pi_{i|\delta}\, m$ apparaît dans $m_{s(\eta)}$ alors il existe m' tel que $\Pi_{i|\delta}\, m'$ apparaît dans $m_{s(\nu)}$.*

Preuve. Supposons qu'aucune itération de la forme $\Pi_{i|\delta}\, m'$ n'apparaisse dans $m_{s(\nu)}$. C'est donc que, pour l'une des règles appliquées entre ν et η, la prémisse (ou plutôt son instance sur laquelle s'applique la règle) ne contient pas de motif de la forme $\Pi_{i|\delta}\, m'$, mais que la conclusion contient bien un tel motif. La seule règle de Dpll* qui a cette propriété est Expansion (et Deep splitting, qui modifie le domaine, mais comme nous l'avons vu Deep splitting n'est jamais appliquée). Nous montrons donc qu'il est impossible que η contienne une itération $\Pi_{i|\delta}\, m'$ introduite par Expansion.

Supposons qu'il existe une telle itération. Par définition de la règle et de la stratégie τ, δ a nécessairement la forme $\alpha n + q \neq n + q' \wedge i = 0$, où $\alpha \in \{0, 1\}$. Nous avons alors deux possibilités :

- soit $\text{context}(s(m')) \Rightarrow \alpha n + q \neq n + q'$ est valide : dans ce cas, comme η est irréductible par l'étape 2, il l'est en particulier par Algebraic simplifications; donc, comme i n'apparaît pas dans m' (en effet, i est une nouvelle variable), Algebraic simplifications peut s'appliquer et remplacer l'itération par m', ce qui est contradictoire avec le fait que le nœud est irréductible ;
- sinon, $\neg(\alpha n + q \neq n + q')$ est satisfaisable. Supposons que cette formule est même valide, c.-à-d. que $\alpha n + q \neq n + q'$ est insatisfaisable, alors l'itération est vide donc Algebraic simplifications est applicable, contradictoire avec le fait que le nœud est irréductible. Supposons donc que $\neg \alpha n + q \neq n + q'$ est satisfaisable sans être valide. Par conséquent, Constraint Splitting est applicable, contradiction.

En conclusion, il est impossible d'avoir une itération $\Pi_{i|\delta}\, m'$ introduite par Expansion en ν. Par conséquent l'itération $\Pi_{i|\delta}\, m$ provient nécessairement de la réécriture d'un certain $\Pi_{i|\delta}\, m'$ apparaissant dans ν. □

Lemma 9.10 (Étape 3)**.** *Soit ν un nœud de t et η un nœud obtenu par application de l'étape 3 à ν, jusqu'à irréductibilité. S'il existe des expressions f et g telles que ν est aligné sur $[f; g]$ et $c_{s(\eta)} \models f \leq g$, alors η est aligné sur $[f; g - k]$ pour un certain $k \geq 1$.*

Preuve. Soit k l'entier maximal tel que $c_{s(\eta)} \models f \leq g - k + 1$. D'après les hypothèses du lemme, nous avons $k \geq 1$. Comme $c_{s(\eta)} \models f \leq g$, Unfolding (seule règle de l'étape 3) peut être appliquée. D'après la définition de la stratégie τ, e (selon la notation utilisée dans la définition de Unfolding) est nécessairement la borne supérieure de l'itération. Comme ν est aligné sur $[f; g]$, cette borne est g. Donc comme Unfolding est la seule règle appliquée, toutes les itérations sont de la forme $\Pi_{i=f}^{g-k'} m$ où $k' \in \mathbb{N}$ (obtenues en dépliant k' fois une itération du nœud initial). Comme cette itération est issue de l'application de la règle, alors, par définition nous devons avoir $c_{s(\nu)} \models f \leq g - k' + 1$ donc $k' \leq k$. Puisque l'itération est irréductible, nous devons avoir $c_{s(\nu)} \not\models f \leq g - k'$ d'où $k' = k$. □

Lemma 9.11. *Si un nœud d'alignement ν est aligné sur $[e; f]$, alors $c_{s(\nu)} \models e \leq f$.*

Preuve. Notons ν_1 le plus lointain ancêtre de ν tel que seule l'étape 2 s'applique entre ν_1 et ν. Notons ν_0 le plus lointain ancêtre de ν_1 tel que seule l'étape 1 s'applique entre ν_0 et ν. D'après la lemme 9.9, nous savons que ν_1 est aussi aligné sur $[e; f]$. De même d'après la lemme 9.8 (i), nous savons qu'il en va de même pour ν_0. Donc, d'après la lemme 9.8 (ii), $c_{s(\nu_1)} \models e \leq f$. Il est facile de voir que les conséquences logiques d'une contrainte *avant application* d'une règle sont toujours conséquences *après application* (en effet, seule la règle Constraint Splitting modifie la contrainte, et la modification apportée est une *conjonction* avec l'ancienne contrainte). Par conséquent nous avons aussi $c_{s(\nu)} \models e \leq f$. □

Nous obtenons enfin la propriété principale des nœuds d'alignement : tout nœud d'alignement est aligné (!)

Proposition 9.12. *Tout nœud d'alignement ν est aligné sur $[q_1; n - q_2 - k]$ pour un certain $k \in \mathbb{N}$.*

De plus, si ν est l'ancêtre d'un nœud d'alignement η, alors il existe $k' > k$ tel que η est aligné sur $[q_1; n - q_2 - k']$.

Preuve. Nous montrons le premier résultat par récurrence sur le nombre de nœuds d'alignement au dessus de ν. Le deuxième résultat sera démontré en même temps que le cas inductif. Tout d'abord, pour le cas de base, c.-à-d. si le nombre de nœuds d'alignement au dessus de ν est nul, alors le résultat est une conséquence du fait que la racine est alignée sur $[q_1; n-q_2]$ et que les étapes 1 et 2 préservent l'alignement (d'après les lemmes 9.8 (i) et 9.9).

Dans le cas inductif, il existe un ancêtre η de ν qui est un nœud d'alignement. Par hypothèse de récurrence, η est aligné sur $[q_1; n - q_2 - k]$ pour un certain $k \in \mathbb{N}$. D'après le lemme 9.11, nous avons $c_{s(\eta)} \models q_1 \leq n - q_2 - k$. Donc les itérations sont alignées sur $[q_1; n - q_2 - k]$, ainsi le nœud obtenu par application de l'étape 3 à η est aligné sur $[q_1; n - q_2 - k - k']$, $k' > 0$, d'après la proposition 9.10. De plus les étapes 1 et 2 préservent l'alignement (d'après les propositions 9.8 et 9.9), donc ν est aligné sur $[q_1; n - q_2 - k - k']$. □

Un nœud d'alignement aligné sur $[q_1; n - q_2 - k]$ pour un certain $k \in \mathbb{N}$ est appelé un *k-nœud d'alignement*. D'après la proposition précédente, tout nœud d'alignement est un k-nœud d'alignement pour un certain $k \in \mathbb{N}$. Notons que si le nœud ne contient pas d'itération, alors il est aligné sur n'importe quel intervalle, donc c'est un k-nœud d'alignement *pour tout* $k \in \mathbb{N}$. Autrement k est unique.

En revanche pour tout $k \in \mathbb{N}$, il n'existe pas nécessairement de k-nœud d'alignement correspondant. En effet, supposons que l'étape 2 s'applique sur le schéma $p_1 \wedge (\bigwedge_{i=1}^{n} p_i) \wedge p_n \,\&\, n \geq 1$: alors, durant cette étape, la règle Propositional Splitting peut évaluer p_1 à true, c.-à-d. ajouter p_1 à l'ensemble de littéraux; Expansion va alors remplacer p_n par $\bigwedge_{i'|n\neq 1 \wedge i'=0} p_n$; puis Propositional Splitting va faire un branchement selon que $n \neq 1$ ou $n = 1$; dans le premier cas nous aurons donc $n \geq 2$ (car nous avions déjà $n \geq 1$), par conséquent la règle Unfolding peut s'appliquer deux fois et donc transformer le schéma en $p_1 \wedge (\bigwedge_{i=1}^{n-2} p_i) \wedge p_{n-1} \wedge p_n \wedge p_n \,\&\, n \geq 1 \wedge n \neq 1$. Ainsi le nœud d'origine était un 0-nœud d'alignement et le suivant est un 2-nœud d'alignement : il n'y a pas de 1-nœud d'alignement.

Nous pouvons en fait dire plus que la proposition précédente : tout nœud d'alignement est non seulement aligné, mais en plus il est régulier imbriqué.

Corollary 9.13. *Pour tout nœud d'alignement ν, $s(\nu)$ est un schéma régulier imbriqué.*

Preuve.

- La procédure ne peut pas introduire de nouveau paramètre : le seul moyen pour que les règles puissent introduire un nouveau paramètre serait qu'elles suppriment une itération liant une variable alors que le corps de l'itération contient toujours cette variable. Il est facile de voir que ce n'est jamais le cas.
- L'aspect monadique est trivialement préservé (l'arité des propositions indexées n'est pas modifiée par les règles).
- Les schémas restent à translation bornée tout au long de l'application des règles : en effet la seule règle qui peut introduire de nouvelles expressions arithmétiques dans le schéma est Unfolding[4]; or

[4]En réalité, R_{syn} peut aussi introduire de nouvelles expressions via l'élimination des quantificateurs et la règle Linéarisation mais nous avons vu dans la preuve de la proposition 8.3 que ce n'était pas le cas.

l'expression en question est obtenue en remplaçant une variable liée i dans une expression e par une expression de la forme $n - q_2 - k$ (car UNFOLDING s'applique toujours sur un nœud d'alignement et que ceux-ci sont alignés sur $[q_1; n - q_2 - k]$ d'après la proposition 9.12). Comme le schéma est à translation bornée, e a elle-même la forme q ou $i + q$, où $q \in \mathbb{Z}$. Dans un cas comme dans l'autre, $e[n - q_2 - k/i]$ contient au plus une variable : n. Ainsi le schéma est toujours à translation bornée après application de la règle.

- Enfin l'alignement est une conséquence triviale de la proposition 9.12.

□

L'autre propriété importante des nœuds d'alignement est qu'une branche infinie contient nécessairement une infinité de nœuds d'alignement. C'est cette propriété qui justifie que nous puissions restreindre notre raisonnement aux nœuds d'alignement.

Lemma 9.14. *Chacune des étapes 1, 2 et 3 (prise indépendamment) termine.*

Preuve.

- Étape 1 : comme déjà vu dans la preuve du lemme 9.8, CONSTRAINT SPLITTING ne s'applique qu'une fois, d'où le résultat.

- Étape 2 : aucune règle de l'étape 2 ne peut ajouter de littéral dans le schéma, donc PROPOSITIONAL SPLITTING est appliqué une quantité finie de fois. Cette règle ajoute donc une quantité finie d'éléments à l'ensemble de littéraux étiquetant le nœud, suscitant alors (au plus) autant d'applications de EXPANSION sur chaque proposition indexée. Enfin CONSTRAINT SPLITTING NON ENCADRÉ est appliquée autant de fois qu'il y a d'itérations ajoutées par EXPANSION.

- Étape 3 : seule UNFOLDING peut être appliquée et, pour pouvoir l'appliquer, la contrainte du nœud doit assurer que les itérations ne sont pas vides. Or cette contrainte est constante tout au long de l'étape 3 (car, justement, seule UNFOLDING peut être appliquée), en revanche les itérations ne le sont pas puisqu'elles sont toutes déroulées par UNFOLDING. Comme la longueur des itérations décroît à chaque application, la contrainte ne peut pas assurer indéfiniment que toutes les itérations ne sont pas vides.

□

Proposition 9.15. *Toute branche infinie contient une infinité de nœuds d'alignement.*

Preuve. Conséquence du lemme précédent. □

Enfin, la proposition suivante sera utile pour la démonstration de la proposition 9.31. Elle permet en plus de mieux saisir le comportement de la procédure.

Proposition 9.16. *Pour tout nœud d'alignement ν de t, n n'apparaît que dans le domaine des itérations.*

Preuve. Supposons que ν est un k-nœud d'alignement pour un certain $k \in \mathbb{N}$. Il suffit de montrer qu'aucun littéral de $s(\nu)$ ne contient n. Pour ce faire, nous montrons qu'un tel littéral est toujours éliminé par la procédure.

Soit donc l un littéral de $s(\nu)$ contenant n. Nous montrons alors que PROPOSITIONAL SPLITTING va nécessairement ajouter l à $L(\nu)$. Pour ce faire, il faut que l apparaisse toujours dans $s(\nu)$ et que ni l ni son complémentaire ne puisse apparaître dans $L(\nu)$. Nous montrons donc ces deux résultats :

- *l apparaît toujours dans $s(\nu)$.* Contrairement aux apparences, ce n'est pas évident : l peut apparaître dans $s(\nu)$ sans apparaître toujours, en effet l peut apparaître dans une itération (car, dans ce cas, pour tout environnement ρ qui rend l'itération vide, $\langle l \rangle_\rho$ n'apparaît pas dans $\langle s(\nu) \rangle_\rho$). Comme ν est un nœud d'alignement, et d'après le lemme 9.11 : $c_{s(\nu)} \models q_1 \leq n - q_2 - k$. Nous avons donc l'assurance que les itérations ne sont pas vides. Dans les deux cas l apparaît bien toujours dans $s(\nu)$.

- *ni l ni son complémentaire ne peuvent apparaître dans $L(\nu)$.* Supposons que ça ne soit pas le cas. Il existe donc dans $L(\nu)$ un littéral l' tel que, pour au moins un environnement ρ, $\langle l' \rangle_\rho = \langle l \rangle_\rho$. Par conséquent Expansion aurait réécrit l en une itération exprimant l'inégalité entre les indices de l et de l'. Par conséquent Constraint Splitting non encadré aurait été appliqué et aurait soit supprimé l, impossible puisque l apparaît dans $s(\nu)$, soit ajouté une contrainte assurant justement que $\langle l' \rangle_\rho \neq \langle l \rangle_\rho$ quel que soit ρ, impossible puisque c'est justement l'hypothèse initiale sur l'. Dans les deux cas nous obtenons une contradiction.

Par conséquent Propositional Splitting est nécessairement applicable, donc $l \in L(\nu)$. Ainsi Expansion peut éliminer l de $s(\nu)$. □

9.3.2 "Tracing" Dpll*

Rappelons que notre but est de prouver que les ensembles suivants sont finis à un décalage près :

$$\{\textstyle\bigwedge_{l \in L(\nu)} l \mid \nu \text{ est un nœud d'alignement de } b\}$$
$$\{c_{s(\nu)} \mid \nu \text{ est un nœud d'alignement de } b\}$$
$$\{m_{s(\nu)} \mid \nu \text{ est un nœud d'alignement de } b\}$$

Parmi ces trois tâches, la plus difficile est la dernière. En effet elle nécessite une preuve par induction sur la structure du motif initial : nous montrons de cette manière que chaque sous-motif ne peut engendrer qu'une quantité finie à un décalage près de motifs, ce qui permet d'obtenir le résultat pour le motif initial. Pour mener cette preuve à bien il faut définir "les motifs *engendrés* par un sous-motif". Plus précisément, il faut pouvoir *suivre* l'évolution d'un sous-motif à travers les applications des règles de Dpll* : celui-ci peut être déplacé, dupliqué, supprimé, un de ses sous-motifs peut subir les mêmes transformations, etc. Pour suivre ces transformation, nous définissons une version "tracée" de Dpll* : T-Dpll* ("T" pour **T**racé) où chacun des nœuds de l'arbre est associé à un ensemble de positions associées au motif que nous cherchions à tracer (initialement, la position est unique, mais, certaines des règles pouvant dupliquer les motifs, il est possible d'obtenir plus d'une position).

Pour suivre un sous-motif, nous l'identifions par sa position dans le motif :

Definition 9.17 (Position)**.** *Comme d'habitude une* position *est un mot sur les entiers; les* positions *d'un motif sont définis comme suit :*

- *ϵ est une position de tout motif;*
- *si p est une position dans un motif m, alors $1 \cdot p$ est une position dans $\neg m$, $\bigwedge_{i|\delta} m$ et $\bigvee_{i|\delta} m$;*
- *soient deux motifs m_1 et m_2, et $i \in \{1, 2\}$; si p est une position dans m_i, alors $i \cdot p$ est une position dans $m_1 \wedge m_2$ et $m_1 \vee m_2$.*

Pour tout motif m et toute position p dans m, $[m]_p$ est le sous-motif de m en position p, défini de façon habituelle.

Pour rappel (définition 2.30), nous notons $p_1 < p_2$ ssi p_1 est un préfixe (strict) de p_2. De plus, étant donnés une position p_1 et un préfixe p_2 de p_1, nous notons $p_2 \setminus p_1$ le mot tel que $p_2 \cdot (p_2 \setminus p_1) = p_1$. Le mot $p_2 \setminus p_1$ peut être vu comme la position de p_2 *relativement* au motif en position p_1.

Nous pouvons maintenant présenter T-Dpll* qui est défini comme Dpll* mais en accompagnant chaque nœud d'un ensemble de positions. Cet ensemble de positions désigne les sous-motifs que nous souhaitons "tracer" : les règles sont étendues de sorte à modifier cet ensemble de positions. Ainsi les positions *après application des règles* permettent de savoir ce que sont devenus les sous-motifs caractérisés par les positions *avant application*. Nous donnons maintenant la définition formelle de T-Dpll* :

Definition 9.18 (T-Dpll*)**.** *Un tableau de T-*Dpll* *est défini comme un tableau de* Dpll* *dont tout nœud ν, en plus d'être étiqueté par $s(\nu)$ et $L(\nu)$, est étiqueté par un ensemble de positions de $m_{s(\nu)}$ noté $P(\nu)$. Les règles de T-*Dpll* *sont les mêmes que celles de* Dpll* *en ce qui concerne $s(\nu)$ et $L(\nu)$. Il faut juste préciser leur comportement sur $P(\nu)$. Pour cela, nous utilisons la notation $p \to p_1 \dots p_n$, $\nu \in \mathbb{N}$, qui signifie que p est supprimé de $P(\nu)$ et $p_1 \dots p_n$ lui sont ajoutées, les autres positions restant inchangées. Les règles sont les suivantes :*

- *les règles de branchement laissent $P(\nu)$ tel quel;*
- *pour les règles de réécriture autres que* Expansion *: nous notons q la position du sous-motif qui est réécrit. Alors* la règle $p \to \dots$ ne s'applique que si $q < p$. *Selon les règles, ces positions sont modifiées de la façon suivante :*
  - *pour* Algebraic simplifications*, règle par règle :*
    - *Pour toutes les règles dont le membre droit est $\top$ ou $\bot$:*
    $$p \to \emptyset$$
    - *Pour $m \wedge \top \to m$, $m \vee \bot \to m$, et $\Pi_{i=1}^{n} m \to m$:*
    $$p \to q \cdot (1 \setminus (q \setminus p))$$
    si p est la position d'un sous-motif de m
    $$p \to \emptyset$$
    si p est la position de $\top$ ou $\bot$.
    - *Pour $m \wedge m \to m$ et $m \vee m \to m$:*
    $$p \to q \cdot (1 \setminus (q \setminus p))$$
    si p est la position d'un sous-motif de l'occurrence gauche de m
    $$p \to q \cdot (2 \setminus (q \setminus p))$$
    si p est la position d'un sous-motif de l'occurrence droite de m.
  - *pour* Unfolding *:*
  $$p \to q \cdot 1 \cdot (q \setminus p),\ q \cdot 2 \cdot 1 \cdot (q \setminus p)$$
  - *nous ne précisons pas le comportement de* Deep splitting *puisque cette règle n'est jamais appliquée sur les schémas réguliers imbriqués.*
  - Expansion *laisse $P(\nu)$ tel quel.*

De façon informelle, T-Dpll* ne fait que "suivre" le motif (suivant la notation utilisée par toutes les règles), et ses sous-motifs. À la racine, $P(\nu)$ contient en général une unique position p, repérant un sous-motif m. A chaque noeud ν, $P(\nu)$ contient alors l'ensemble des positions qui correspondent à des parties de la formule descendant de m (éventuellement après transformation).

Note. La stratégie τ, la notion de nœud d'alignement, ainsi que les lemmes associés s'étendent naturellement à T-Dpll*.

Definition 9.19 (Décoration)**.** *Soient ν un nœud dans un tableau de* Dpll* *et p une position de $m_{s(\nu)}$. Nous notons t le sous-arbre de racine ν.*

Nous appelons décoration *de t par rapport à p, notée t_p, l'arbre obtenu à partir de t en ajoutant à l'étiquette de ν l'ensemble ne contenant que la position p (c.-à-d. $\{p\}$) et en remplaçant chaque application d'une règle de* Dpll* *par la règle correspondante de T-*Dpll*[5]*.*

Pour un descendant ν' de ν et un motif m', nous notons $[m_{s(\nu)}]_p \rightsquigarrow_{t_p}^{\nu'} m'$ ssi il existe une position $p' \in P(\nu')$ telle que $m' = [m_{s(\nu')}]_{p'}$ dans la décoration de t par rapport à p.

Example 9.20.

- *Supposons que $r_1 \wedge ((r_2 \wedge r_3) \wedge \top)$ soit réécrit en $r_1 \wedge (r_2 \wedge r_3)$ par* Algebraic simplifications*. Dans ce cas le sous-motif auquel s'applique la réécriture est $(r_2 \wedge r_3) \wedge \top$ qui est en position 2, donc $q = 2$. Alors les positions évoluent de la façon suivante :*
  - *ϵ est inchangé car $q \not< \epsilon$. De même les positions 1 et 2 sont inchangées.*

[5]Il est évident qu'un tel arbre existe toujours, puisque les conditions d'application des règles ne sont pas modifiées.

- *Nous avons bien* $2 \cdot 1 > 2$*, par conséquent la (deuxième) réécriture de positions s'applique, c.-à-d.* $p \to q \cdot (1 \setminus (q \setminus p))$*. Comme* $q = 2$ *et* $p = 2 \cdot 1$*,* $q \setminus p = 1$ *et donc* $1 \setminus (q \setminus p) = \epsilon$*. Par conséquent* $2 \cdot 1 \to q$*, c.-à-d.* 2*.*
- *Selon le même raisonnement :* $2 \cdot 1 \cdot 1 \to 2 \cdot 1$ *et* $2 \cdot 1 \cdot 2 \to 2 \cdot 2$*.*
- *De même* $2 \cdot 2$ *peut se réécrire. Comme c'est la position de* $\top$*, elle est en fait supprimée.*

L'évolution des motifs est plus claire graphiquement (les sous-termes soulignés correspondent aux positions de $P(\nu)$*):*

$$\underline{r_1 \wedge ((r_2 \wedge r_3) \wedge \top)} \to \underline{r_1 \wedge (r_2 \wedge r_3)}$$
$$\underline{r_1} \wedge ((r_2 \wedge r_3) \wedge \top) \to \underline{r_1} \wedge (r_2 \wedge r_3)$$
$$r_1 \wedge \underline{((r_2 \wedge r_3) \wedge \top)} \to r_1 \wedge \underline{(r_2 \wedge r_3)}$$
$$r_1 \wedge (\underline{(r_2 \wedge r_3)} \wedge \top) \to r_1 \wedge \underline{(r_2 \wedge r_3)}$$
$$r_1 \wedge ((\underline{r_2} \wedge r_3) \wedge \top) \to r_1 \wedge (\underline{r_2} \wedge r_3)$$
$$r_1 \wedge ((r_2 \wedge \underline{r_3}) \wedge \top) \to r_1 \wedge (r_2 \wedge \underline{r_3})$$
$$r_1 \wedge ((r_2 \wedge r_3) \wedge \underline{\top}) \to r_1 \wedge (r_2 \wedge r_3)$$

- *Supposons que* $r_1 \wedge \bigwedge_{i=1}^{n} r_i$ *soit réécrit en* $r_1 \wedge (r_n \wedge \bigwedge_{i=1}^{n-1} r_i)$ *par* Unfolding*. Dans ce cas le sous-motif auquel s'applique la réécriture est* $\bigwedge_{i=1}^{n} r_i$ *qui est en position* 2*, donc* $q = 2$*. Alors les positions évoluent de la façon suivante :*

$$\underline{r_1 \wedge \textstyle\bigwedge_{i=1}^{n} r_i} \to \underline{r_1 \wedge (r_n \wedge \textstyle\bigwedge_{i=1}^{n-1} r_i)}$$
$$\underline{r_1} \wedge \textstyle\bigwedge_{i=1}^{n} r_i \to \underline{r_1} \wedge (r_n \wedge \textstyle\bigwedge_{i=1}^{n-1} r_i)$$
$$r_1 \wedge \underline{\textstyle\bigwedge_{i=1}^{n} r_i} \to r_1 \wedge \underline{(r_n \wedge \textstyle\bigwedge_{i=1}^{n-1} r_i)}$$
$$r_1 \wedge \textstyle\bigwedge_{i=1}^{n} \underline{r_i} \to r_1 \wedge (\underline{r_n} \wedge \textstyle\bigwedge_{i=1}^{n-1} \underline{r_i})$$

Notons que dans le dernier cas la position génère deux positions distinctes, car la sous-formule correspondante (r_i) *est dupliquée (puis instanciée) par la règle.*

- *Évidemment pour toute autre règle, quel que soit l'exemple, les positions ne changent pas.*

La proposition suivante montre tout l'intérêt de T-Dpll*. Elle nous permet de caractériser l'ensemble des motifs apparaissant à un noeud donné, ce qui est essentiel pour la preuve de terminaison :

Proposition 9.21. *Soient un nœud* ν *et son fils* η *dans un tableau* t *de* T-Dpll**. Soit un motif* m *en position* p *dans* $m_{s(\nu)}$*. Alors, pour tout motif* m' *tel que* $m \rightsquigarrow_{t_p}^{\eta} m'$ *:*

- *soit* $m' = m$*,*
- *soit il existe une substitution* σ *telle que* $m' = m\sigma$*,*
- *soit* $m \to m'$ *suivant l'une des règles de réécritures de* Dpll**.*

Preuve. Si la règle est une réécriture au dessus de p ou dans une position parallèle à p, ou si la règle laisse $P(\nu)$ tel quel, alors nous obtenons le premier cas. Si la règle appliquée est Unfolding et que la position de m' correspond à $q \cdot 2 \cdot 1 \cdot (q \setminus p)$, alors nous obtenons le deuxième cas. Enfin pour toute application de Algebraic simplifications réécrivant m lui-même ou l'un de ses sous-motifs, nous obtenons le troisième cas. □

Le point important est que *même lorsque la règle appliquée est une réécriture au dessus de* p la transformation est une identité du point de vue de $\rightsquigarrow_{t_p}^{\eta}$. Sans T-Dpll*, nous serions incapable de relier deux sous-motifs apparaissant dans des positions distinctes avant et après application d'une règle.

Enfin, la proposition suivante permettra d'appliquer à Dpll* les résultats obtenus avec T-Dpll* :

Proposition 9.22. *L'ensemble ne contenant que la position* ϵ *est invariant par les règles de réécriture ci-dessus.*

Preuve. Par inspection des règles. Comme une position n'est réécrite que si elle est un suffixe *strict* du motif réécrit, ϵ n'est jamais modifié. □

Corollary 9.23. *Soient deux nœuds ν et η d'un tableau t de* DPLL*. *Si m est un motif tel que $m_{s(\nu)} \rightsquigarrow^{\eta}_{t_\epsilon} m$, alors $m = m_{s(\eta)}$.*

Preuve. Nous avons $m_{s(\nu)} = [m_{s(\nu)}]_\epsilon$. D'après la proposition précédente, pour tout m tel que $m_{s(\nu)} \rightsquigarrow^{\eta}_{t_\epsilon} m$, $m = [m_{s(\eta)}]_\epsilon$. Nous obtenons le résultat avec $[m_{s(\eta)}]_\epsilon = m_{s(\eta)}$. □

9.3.3 Preuve principale

Nous pouvons maintenant nous atteler aux démonstrations que :

$$\{\textstyle\bigwedge_{l \in L(n)} l \mid n \text{ est un nœud d'alignement de } b\}$$

$$\{c_{s(n)} \mid n \text{ est un nœud d'alignement de } b\}$$

$$\{m_{s(n)} \mid n \text{ est un nœud d'alignement de } b\}$$

sont finis à un décalage près, dans cet ordre. Dans toute la suite b est une branche infinie. Pour tout $k \in \mathbb{N}$, nous notons ν_k le k-nœud d'alignement de b, s'il existe (si un nœud ne contenait pas d'itération, la branche serait finie, donc tout nœud contient au moins une itération, donc ν_k est unique).

Littéraux

Dans le premier cas, nous montrons, comme pour les schémas réguliers, que la plupart des littéraux deviennent purs. De plus l'ensemble des littéraux non purs est fini à un décalage près. Soit μ le plus grand entier naturel apparaissant dans s (dans une borne ou dans un indice, précédé d'un "$-$" ou pas, exactement comme en section 6.4.1).

Lemma 9.24. *Soit un nœud ν de t. Tout littéral de $sL(\nu)$ dont l'indice est un entier apparaît dans le schéma initial.*

Preuve. Nous montrons qu'aucune règle de DPLL* ne peut introduire de littéral dont l'indice est un entier (le résultat final est ensuite facilement obtenu par récurrence sur la longueur de la dérivation). C'est évident pour toutes les règles exceptée UNFOLDING qui effectue une substitution et peut donc transformer une variable en un entier. Cependant cette règle n'est appliquée que juste après un nœud d'alignement. Or nous savons d'après la proposition 9.12 qu'un nœud d'alignement est aligné sur $[q_1; n - q_2 - k]$, pour un certain $k \in \mathbb{N}$. De plus la stratégie τ impose que seul le dernier rang soit déplié, c.-à-d. $n - q_2 - k$. Comme nous pouvons le constater cette expression contient n. De plus les indices des littéraux contenus dans une itération ont tous la forme q ou $i + q$ pour un certain $q \in \mathbb{Z}$. Dans le premier cas, UNFOLDING peut effectivement sortir un littéral dont l'indice est un entier, mais, dans ce cas, ce littéral apparaît déjà dans le schéma initial puisqu'il apparaît dans l'itération. Dans le deuxième cas, la substitution opérée par UNFOLDING ne peut pas introduire de littéral dont l'indice soit un entier. □

Lemma 9.25. *Il existe $k_0 \in \mathbb{N}$ tel que, pour tout $k > k_0$, et tout littéral l de ν_k : si l'indice de l a la forme $n + q$ où $q \notin [-2\mu - k; 2\mu - k]$, alors l est pur dans $sL(\nu)$.*

Note. Ce lemme est très proche des lemmes 6.25, 6.26 et 6.27.

Preuve. Soit un littéral l de ν_l. Tout d'abord notons que l est nécessairement pur dans $L(\nu_k)$, c.-à-d. que son complémentaire ne peut pas apparaître dans $L(\nu_k)$: en effet il s'agit d'une des conditions de PROPOSITIONAL SPLITTING pour que l ait pu être ajouté à cet ensemble. Il faut donc uniquement vérifier que l est pur dans $s(\nu_k)$. De plus, d'après la proposition 9.16, nous savons que $s(\nu_k)$ ne contient aucun littéral dont l'indice contient n. Par ailleurs, comme un nœud d'alignement est irréductible par PROPOSITIONAL SPLITTING, EXPANSION et ALGEBRAIC SIMPLIFICATIONS, $s(\nu_k)$ ne contient même aucun littéral dont l'indice est un entier. Donc les seuls littéraux restant sont des littéraux dont l'indice contient une variable liée par une itération. Ainsi pour vérifier qu'un littéral est pur, il suffit de vérifier que son complémentaire n'apparaît pas implicitement dans les itérations (qui, rappelons-le sont alignées sur $[q_1; n - q_2 - k]$).

Notons $n + q$, $q \in \mathbb{Z}$, l'indice de l. Supposons que le complémentaire de l apparaisse implicitement dans une itération, par conséquent, il existe un littéral apparaissant dans une itération dont l'indice a la

forme $i + q'$, où i est la variable liée et $q' \in \mathbb{Z}$, et tel que $q = -q_2 - k + q'$. Or $-q_2 + q' \leq 2\mu$, donc si $q \geq 2\mu - k$ alors l est pur.

Nous montrons ensuite que si k est suffisamment grand nous ne pouvons tout simplement pas avoir $q \leq -2\mu - k$. Soit l un littéral de cette forme. Vues les restrictions de la stratégie τ, l n'a pu être ajouté à $L(\nu_k)$ que s'il apparaît explicitement. C'est donc soit un littéral du schéma initial, soit un littéral qui a été ajouté par une règle. Il est facile de voir que la seule règle ajoutant des littéraux est Unfolding. Or les littéraux ajoutés par cette règle ont un indice de la forme $n - q_2 - k + q'$, où $q' \in \mathbb{Z}$ est tel qu'il existe un littéral d'indice $i + q'$ dans une itération. Il est clair que nous avons $-q_2 - k + q' \geq -2\mu - k$, donc l ne peut pas avoir été introduit par Unfolding. Ainsi c'est nécessairement un littéral du schéma initial. Dans ce cas, il suffit de prendre k suffisamment grand (ce qui donne implicitement la valeur de k_0) pour s'assurer que tous les littéraux du schéma initial aient un indice tel que $q \geq -2\mu - k$, et nous n'avons effectivement plus aucun littéral tel que $q < -2\mu - k$. □

Proposition 9.26. *$\{\bigwedge_{l \in L(\nu)} l \mid \nu$ est un nœud d'alignement de $b\}$ est fini à un décalage près avec l'extension du littéral pur.*

De plus, il existe $k_0 \in \mathbb{N}$ tel que, pour tout $k \geq k_0$, toute expression apparaissant dans un littéral non pur de $\bigwedge_{l \in L(\nu_k)} l$ a la forme $n - q - k$, où $q \in [-2\mu; 2\mu]$.

Notons que le deuxième résultat est une conséquence triviale du fait que les nœuds d'alignement sont alignés (proposition 9.12) et de la proposition 9.16.

Preuve. Cela revient à prouver que :

$$\{\bigwedge_{\substack{l \in L(\nu) \\ l \text{ non pur}}} l \mid \nu \text{ est un nœud d'alignement de } b\}$$

est fini à un décalage près. Montrons que la suite $(\mathcal{L}_k)_{k \in \mathbb{N}}$ est finie à un décalage près, où $\mathcal{L}_k \stackrel{\text{def}}{=} \bigwedge_{\substack{l \in L(\nu_k) \\ l \text{ non pur}}} l$.

Dans la suite, $\mathcal{K}^1$ désigne la conjonction des littéraux de ν_0 dont l'indice est un entier. De plus, pour tout symbole de proposition p apparaissant dans le schéma d'origine et tout $q \in [-2\mu; 2\mu]$, nous définissons les suites :

$$\mathcal{K}^2_k(p,q) \stackrel{\text{def}}{=} \begin{cases} p_{n-q-k} & \text{si } p_{n-q-k} \in L(\nu_k) \\ \top & \text{sinon} \end{cases}$$

$$\mathcal{K}^3_k(p,q) \stackrel{\text{def}}{=} \begin{cases} \neg p_{n-q-k} & \text{si } \neg p_{n-q-k} \in L(\nu_k) \\ \top & \text{sinon} \end{cases}$$

où $k > k_0$ (k_0 est celui du lemme 9.25). Alors, d'après les lemmes 9.24 et 9.25, pour tout $k > k_0$:

$$\bigwedge_{\substack{l \in L(\nu_k) \\ l \text{ non pur}}} l = \mathcal{K}^1 \wedge \bigwedge_{p,q} \mathcal{K}^2_k(p,q) \wedge \mathcal{K}^3_k(p,q)$$

Or pour tous p et q, les suites $\mathcal{K}^2_k(p,q)$ et $\mathcal{K}^3_k(p,q)$ sont clairement finies à un décalage près. Comme p et q appartiennent à des ensembles finis, nous pouvons appliquer successivement le corollaire 5.25 (le fait que q appartienne à un ensemble fini justifie le fait que cette application successive soit finie *et* le fait que nous puissions appliquer le corollaire), ainsi : $\bigwedge_{p,q}(\mathcal{K}^2_k(p,q) \wedge \mathcal{K}^3_k(p,q))$ est fini à un décalage près. Par suite $\mathcal{K}^1 \wedge \bigwedge_{p,q} \mathcal{K}^2_k(p,q) \wedge \mathcal{K}^3_k(p,q)$ est aussi fini à un décalage près. D'où le résultat. □

Contraintes

Nous montrons maintenant que $\{c_{s(n)} \mid n$ est un nœud d'alignement de $b\}$ est fini à un décalage près par l'extension de la contrainte équivalente. Le premier lemme assure que nous n'avons pas besoin de tenir compte de la contrainte initiale car elle est redondante à partir d'un certain rang (ce qui est particulièrement utile car nous n'avons aucune contrôle sur la forme de cette contrainte).

Lemma 9.27. *Notons c la contrainte du schéma initiale. Alors $c_{s(\nu_k)}$, c.-à-d. la contrainte du nœud ν_k, peut s'écrire $c \wedge c_k$ où c_k est la conjonction des contraintes ajoutées par* Dpll*. *Nous avons alors le résultat suivant :*

Il existe $q_1 \dots q_l \in \mathbb{N}$ et $k_0 \in \mathbb{N}$ t.q. pour tout $k > k_0$, $(q_1|n \vee \dots \vee q_l|n) \wedge c_k \models c$.

Preuve. D'après le lemme 9.11 et la proposition 9.12 : $c_{s(\nu_k)} \models q_1 \leq n - q_2 - k$. En observant les preuves de ces résultats, il est clair que nous avons plus précisément $c_k \models q_1 \leq n - q_2 - k$ (c.-à-d. la contrainte initiale n'a pas d'influence sur ce résultat) et donc $n \geq q_1 + q_2 + k$.

Par ailleurs, la contrainte initiale, par élimination des quantificateurs et mise sous forme normale disjonctive, est une disjonction de formules arithmétiques de la forme $q|n \wedge q' \leq n$ ou $q|n \wedge n \leq q'$ ou $q|n \wedge q' \leq n \wedge n \leq q''$ où $q', q'' \in \mathbb{Z}$, $q \in \mathbb{N}$. Or, toute formule de la forme $n \leq q$ est en contradiction avec $n \geq q_1 + q_2 + k$ si k est suffisamment grand; ce qui est impossible car la branche serait alors fermée, donc il n'existe pas de telle formule. D'autre part, toute formule de la forme $q \leq n$ est conséquence de $n \geq q_1 + q_2 + k$ si k est suffisamment grand. Ainsi seule la disjonction de toutes les formules de la forme $q|n$ dans la contrainte initiale n'est pas conséquence de la c_k. Cette disjonction nous donne les $q_1, \ldots, q_l$ permettant d'obtenir le résultat. □

Une conséquence de la terminaison de l'étape 3 (proposition 9.15), est que pour tout k, $c_{s(\nu_k)} \not\models q_1 \leq n - q_2 - k - k'$ pour un certain $k' \in \mathbb{N}$ (autrement, l'étape 3 déroulerait indéfiniment toutes les itérations). Le lemme suivant dit qu'il existe une borne pour ce k', *indépendante du nœud considéré.*

Lemma 9.28. *Il existe $\kappa \in \mathbb{N}$ tel que pour tout nœud d'alignement ν_k tel que $k > k_0$, où k_0 est celui du lemme précédent : $c_{s(\nu_k)} \not\models q_1 \leq n - q_2 - k - \kappa$.*

Preuve. Sans perte de généralité, supposons que k_0 soit suffisamment grand pour que ν_k ne soit pas le premier nœud d'alignement. Par conséquent ν_k a au moins un ancêtre qui est un nœud d'alignement. Soit $k' \in \mathbb{N}$ tel que $\nu_{k'}$ soit le plus proche de ces ancêtres. Nous analysons la façon dont DPLL* ajoute des contraintes entre $\nu_{k'}$ et ν_k. Seules CONSTRAINT SPLITTING ENCADRÉ et CONSTRAINT SPLITTING NON ENCADRÉ peuvent ajouter des contraintes. Comme ν_k est un k-nœud d'alignement, nous savons déjà que CONSTRAINT SPLITTING ENCADRÉ (appliqué lors de l'étape 1 entre $\nu_{k'}$ et ν_k) a introduit la contrainte $q_1 \leq n - q_2 - k$. Par ailleurs, EXPANSION peut introduire des itérations dont le domaine a la forme $e \neq f \wedge i = 0$, donc CONSTRAINT SPLITTING NON ENCADRÉ peut introduire des contraintes de la forme $e = f$ ou $e \neq f$.

Considérons d'abord le premier cas. e et f ont chacun la forme $n+q$ ou q, où $q \in \mathbb{Z}$. Si les deux ont la même forme alors $e = f$ est trivialement valide ou insatisfaisable. Si $e = f$ est valide, alors la contrainte n'est pas modifiée. En revanche si $e = f$ est insatisfaisable alors la contrainte devient insatisfaisable et le nœud est clos, ce qui est impossible car la branche est infinie. Reste le cas où e et f ont tous deux une forme différente : si l'un a la forme $n + q$ et l'autre est un entier q', alors $e = f$ équivaut à $n = q' - q$. Dans ce cas la contrainte revient à donner une valeur à n. Donc la branche est trivialement finie (DPLL* va appliquer UNFOLDING jusqu'à ce que ce nous obtenions la formule propositionnelle correspondant à la valeur de n, puis, comme nous avons une formule propositionnelle, la terminaison est triviale), ce qui est impossible.

Contrairement au cas $e = f$, le cas $e \neq f$ peut modifier la contrainte. Nous pouvons avoir p.ex. $q_1 \neq n - q_2 - k$, et alors la contrainte entraîne $q_1 \leq n - q_2 - k - 1$. Donc si CONSTRAINT SPLITTING NON ENCADRÉ introduit les contraintes $q_1 \neq n - q_2 - k$, $q_1 \neq n - q_2 - k - 1$, $q_1 \neq n - q_2 - k - 2$, etc. alors il n'existe pas de κ ayant la propriété voulue (et, vu le raisonnement effectué, c'est la seule façon de contredire la propriété). Notre but est donc de montrer que nous ne pouvons pas avoir $q_1 \neq n - q_2 - k$, $q_1 \neq n - q_2 - k - 1$, $q_1 \neq n - q_2 - k - 2$, etc., c.-à-d. qu'il existe κ tel que pour tout $k' \in \mathbb{N}$ tel que $q_1 \neq n - q_2 - k - k'$, $k' \leq \kappa$. Nous ne pouvons avoir une telle divergence que si e a la forme $n + q$ pour un certain $q \in \mathbb{Z}$ et f est un entier, ou réciproquement (en effet dans les autres cas la contrainte ajoutée est soit valide, ce qui ne modifie pas les conséquences logiques de la contrainte, soit insatisfaisable, ce qui est impossible car la branche est infinie). Par définition de EXPANSION, e et f sont les indices d'un littéral de $s(\eta)$ et d'un littéral de $L(\eta)$, respectivement, où η est un nœud compris entre $\nu_{k'}$ et ν_k. Comme tous les littéraux dont l'indice est un entier sont éliminés dès le premier nœud d'alignement, le cas où e est un entier n'est pas à considérer. Donc e a la forme $n + q$, $q \in \mathbb{Z}$. Ainsi, toute contrainte ajoutée a la forme $n + q \neq f$, ce qui équivaut à $q_1 \neq n - q_2 - k - (f - q - q_1 - k - q_2)$. Nous voulons donc majorer $f - q - q_1 - k - q_2$. Le littéral dont e est l'indice provient nécessairement du dépliage d'une itération de $\nu_{k'}$ (c'est la seule façon d'introduire un littéral et les littéraux du schéma initial ont été éliminés avant $\nu_{k'}$). Comme les itérations de $\nu_{k'}$ sont alignées sur $[q_1; n - q_2 - k']$ et qu'après dépliage, elles sont alignées sur $[q_1; n - q_2 - k]$: $-q_2 - k - \mu \leq q \leq -q_2 - k' + \mu$. Par ailleurs f est compris entre $-\mu$ et μ. Donc : $f - q - q_1 - k - q_2 \leq \mu + (q_2 + k - \mu) - q_1 - k - q_2$, c.-à-d. $f - q - q_1 - k - q_2 \leq -q_1$. Nous posons donc : $\kappa \stackrel{\text{def}}{=} -q_1 + 1$.

Donc nous avons au plus $c_{s(\nu_k)} \models q_1 \leq n - q_2 - k - \kappa + 1$, mais il est impossible d'avoir $c_{s(\nu_k)} \models q_1 \leq n - q_2 - k - \kappa$, donc $c_{s(\nu_k)} \not\models q_1 \leq n - q_2 - k - \kappa$. □

Corollary 9.29. *Soient ν_k un nœud d'alignement tel que $k \geq k_0$ (où k_0 est celui du lemme 9.27) et $\nu_{k'}$ le descendant le plus proche de ν_k qui soit un nœud d'alignement, alors :*

(i) $k' - k \leq \kappa$ (où κ est celui du lemme précédent).

(ii) de plus si η est un nœud compris entre ν_k et $\nu_{k'}$, l'indice de tout littéral de $s(\eta)$ a la forme $n - q_2 - k + q$ où $-\kappa - \mu \leq q \leq \mu$.

Preuve. (i) D'après le lemme précédent : $c_{s(\nu_k)} \not\models q_1 \leq n - q_2 - k - \kappa$. Par conséquent l'étape 3 appliquée à ν_k va dérouler les itérations au pire κ fois. Le schéma obtenu alors sera aligné sur $[q_1; n - q_2 - k - \kappa]$. D'après les lemmes 9.8 et 9.9 l'alignement ne change pas durant les étapes 1 et 2, par conséquent $\nu_{k'}$ est aussi aligné sur $[q_1; n - q_2 - k - \kappa]$ au pire. Comme par ailleurs il est aligné sur $[q_1; n - q_2 - k']$ par définition, nous obtenons le résultat.

(ii) Tous les littéraux ajoutés entre ν_k et $\nu_{k'}$ proviennent des dépliages de l'étape 3 : tout indice e d'un littéral provenant du dépliage d'un rang i est de la forme $n - q_2 - i + q$ où $-\mu \leq q \leq \mu$. Comme tous les rangs entre ν_k et $\nu_{k'}$ sont dépliés, nous avons de plus : $k \leq i \leq k'$, et donc, d'après le résultat (i), $k \leq i \leq k + \kappa$. D'où le résultat. □

Proposition 9.30. *$\{c_{s(\nu)} \mid \nu$ est un nœud d'alignement de $b\}$ est fini à un décalage près avec l'extension de la contrainte équivalente.*

De plus il existe $k_0 \in \mathbb{N}$ tel que, pour tout $k > k_0$, toute expression contenant n apparaissant dans $c_{s(\nu_k)}$ (modulo équivalence entre contraintes) a la forme $n - q_2 - k + q$, où $-\kappa - \mu \leq q \leq \mu$.

Preuve. D'après le lemme 9.27, la contrainte initiale ne pose pas de problème car elle est redondante à partir d'un certain rang, excepté les atomes de la formes $q|n$. Or, pour tout $q \in \mathbb{N}$, $q|n$ est équivalent à $q|(n - q)$, et donc, par récurrence, à $q|(n - \lceil \frac{k}{q} \rceil q)$. Nous pouvons donc écrire cette formule sous la forme $q|(n - k + (k - \lceil \frac{k}{q} \rceil q))$, nous avons alors : $k - \lceil \frac{k}{q} \rceil q \leq q$. De plus q est inférieure au maximum des q' tels que $q'|n$ apparaît dans la forme normale disjonctive sans quantificateurs de la contrainte initiale. Ainsi tout atome de cette forme est équivalent à un atome de la forme $q|(n - k + q')$ où q' appartient à un ensemble borné indépendamment de k. Donc la suite $(q|(n - \lceil \frac{k}{q} \rceil q))_{k \in \mathbb{N}}$ est finie à un décalage près. Il en va donc de même de la contrainte initiale.

Ensuite, seule Constraint Splitting *introduit* des contraintes. Comme nous l'avons déjà vu, Constraint Splitting encadré ajoute des contraintes de la forme : $q_1 \leq n - q_2 - k$ (comme déjà vu, $q_1 > n - q_2 - k$ est impossible car il n'y aurait alors plus d'itération et la branche serait finie). Donc la suite des contraintes ajoutées a la forme suivante :

$$\begin{array}{l} q_1 \leq n - q_2 - k_1 \\ q_1 \leq n - q_2 - k_1 \wedge q_1 \leq n - q_2 - k_2 \\ q_1 \leq n - q_2 - k_1 \wedge q_1 \leq n - q_2 - k_2 \wedge q_1 \leq n - q_2 - k_3 \\ \ldots \end{array}$$

qui se ramène, comme $k_1 \leq k_2 \leq k_3 \leq \ldots$ et en utilisant l'extension de la contrainte équivalente, à :

$$\begin{array}{c} q_1 \leq n - q_2 - k_1 \\ q_1 \leq n - q_2 - k_2 \\ q_1 \leq n - q_2 - k_3 \\ \ldots \end{array}$$

qui est clairement fini à un décalage près. De plus, dans toute contrainte de cette forme, la seule expression contenant n est de la forme $n - q_2 - k$ (utile pour la deuxième partie du résultat).

Nous nous focalisons maintenant sur les contraintes ajoutées par Constraint Splitting non encadré. Soient ν_k un nœud d'alignement et $\nu_{k'}$ le plus proche descendant de ν_k qui soit un nœud d'alignement. Comme déjà mentionné dans la preuve du lemme 9.28, les seules contraintes ajoutées par Constraint Splitting non encadré entre ν_k et $\nu_{k'}$ sont issues des itérations introduites par Expansion dans un nœud η. Donc ces contraintes ont la forme $e \star f$, où $\star \in \{=, \neq\}$, e est l'indice d'un littéral de $s(\eta)$ et f l'indice d'un littéral de $L(\eta)$. D'après le corollaire 9.29, e a la forme $n - q_2 - k + q$ où $-\kappa - \mu \leq q \leq \mu$. Soit f contient n, soit c'est un entier.

Considérons d'abord les contraintes de la forme $e \star f$ où f contient n. Alors f a la forme $n+q$, $q \in \mathbb{Z}$. Or e a la forme $n+q'$, $q' \in \mathbb{Z}$. Par conséquent $e \star f \equiv q \star q'$. Donc cette contrainte est triviale, c.-à-d. valide ou insatisfaisable. Elle ne peut pas être insatisfaisable car la branche serait alors fermée. Ainsi toutes les contraintes de ce type présentes dans la branche sont équivalentes à $\top$, donc l'ensemble de ces contraintes est fini.

Ensuite, pour les contraintes de la forme $e \star f$ où f est un entier, nous distinguons les cas où $\star$ est $=$ ou $\neq$. Si c'est une égalité, alors la contrainte revient à donner une valeur à n et, comme nous l'avons déjà vu, la terminaison est triviale dans ce cas, ce qui est contradictoire avec le fait que la branche est infinie. Enfin nous montrons que l'ensemble des contraintes de la forme $e \neq f$ où $f \in \mathbb{Z}$ est fini à un décalage près. Pour tout $k \in \mathbb{N}$, nous posons :

$$\mathcal{C}_k \stackrel{\text{def}}{=} \bigwedge_{\substack{(e \neq f) \in c_{s(\nu_k)} \\ e \text{ contient } n \\ f \in \mathbb{Z}}} e \neq f$$

et pour tout $k \geq k_0$ (où k_0 est celui du lemme 9.27), tout $q \in \mathbb{Z}$ t.q. $-\kappa - \mu \leq q \leq \mu$ et tout $q' \in \mathbb{Z}$ t.q. $-\mu \leq q' \leq \mu$, nous posons :

$$\mathcal{D}_k(q, q') \stackrel{\text{def}}{=} \begin{cases} n - q_2 - k + q \neq q' & \text{si cette formule apparaît dans } c_{s(\nu_k)} \\ \top & \text{sinon} \end{cases}$$

D'après le lemme 9.24, nous avons $-\mu \leq f \leq \mu$ et d'après le corollaire 9.29, si $k \geq k_0$ alors e a la forme $n - q_2 - k + q$ où $-\kappa - \mu \leq q \leq \mu$. Donc il est clair que pour tout $k \geq k_0$:

$$\mathcal{C}_k = \bigwedge_{\substack{-\kappa-\mu \leq q \leq \mu \\ -\mu \leq q' \leq \mu}} \mathcal{D}_k(q, q')$$

Or pour tous q et q', la suite $\mathcal{D}_k(q, q')$ est clairement finie à un décalage près. Comme q et q' appartiennent à des intervalles finis, nous pouvons appliquer successivement le corollaire 5.25 (le fait que q et q' appartiennent à des ensembles finis justifient le fait que cette application successive soit finie *et* le fait que nous puissions appliquer le corollaire), ainsi : $\bigwedge_{\substack{-\kappa-\mu \leq q \leq \mu \\ -\mu \leq q' \leq \mu}} \mathcal{D}_k(q, q')$ est fini à un décalage près. De plus, toutes les expressions ont bien la forme voulue, ce qui permet de conclure. □

Motifs

Nous montrons enfin que $\{m_{s(n)} \mid n \text{ est un nœud d'alignement de } b\}$ est fini à un décalage près. C'est seulement ici que la version "tracée" de DPLL* est utile : la preuve se fait par induction sur le motif du schéma initial. C'est aussi la partie la plus intéressante et originale de la preuve.

Proposition 9.31. *$\{m_{s(\nu)} \mid \nu$ est un nœud d'alignement de $b\}$ est fini à un décalage près.*

De plus, pour tout $k \in \mathbb{N}$, toute expression contenant n et apparaissant dans $m_{s(\nu_k)}$ est égale à $n - q_2 - k$.

Preuve. Nous montrons l'énoncé suivant, plus général :

Pour toute position p du schéma initial, l'ensemble :

$\{m \mid [m_{s(\nu)}]_p \rightsquigarrow^{\nu}_{t_p} m$, *$\nu$ est un nœud d'alignement de $b\}$*

est fini à un décalage près (t_p désigne la décoration de t par rapport à p, voir déf. 9.19).

Nous obtenons ensuite le résultat voulu en appliquant le corollaire 9.23. La preuve de l'énoncé se fait par induction sur la structure de $m_{s(\nu)}$: soit un sous-motif m de $m_{s(\nu)}$ en position p, nous montrons que l'ensemble :

$$\{m' \mid m \rightsquigarrow^{\nu}_{t_p} m', \nu \text{ est un nœud d'alignement de } b\}$$

est fini à un décalage près. Pour tout motif m' tel que $m \rightsquigarrow^{\nu}_{t_p} m'$, m' est le fruit de multiples transformations appliquées à m. D'après la proposition 9.21, ces transformations peuvent être les suivantes :

1. identité,
2. réécriture de m ou d'un sous-motif de m,
3. substitution d'une variable par une expression arithmétique.

Seuls les cas 2 et 3 ont donc une influence sur les motifs présents tels que $m \rightsquigarrow^{\nu}_{t_p} m'$. Nous montrons donc que la combinaison de ces transformations ne mènent qu'à un ensemble fini de schémas.

Nous raisonnons donc par induction structurelle sur $m_{s(\nu)}$: soit un sous-motif m de $m_{s(\nu)}$.

- Si m est un littéral d'indice e. Seule la règle Expansion peut réécrire m, et elle ne s'applique – par définition de la stratégie τ – que si e ne contient pas de variable liée. Nous faisons donc la distinction entre le cas où e ne contient pas de variable liée et le cas où e contient une variable liée i.
  - Si e ne contient pas de variable liée, alors Expansion peut s'appliquer. Dans ce cas, le littéral apparaît explicitement dans le motif (car les itérations ne sont pas vides, suivant le même raisonnement que dans la preuve de la proposition 9.16) donc Propositional Splitting est applicable. Comme Propositional Splitting est aussi dans l'étape 2, cette règle va donc s'appliquer avant le prochain nœud d'alignement. Par conséquent, à moins que le littéral n'ait déjà été supprimé, Expansion englobera m dans une itération dont le domaine est $e \neq e \wedge i = 0$. Évidemment, ce domaine est insatisfaisable, par conséquent Algebraic simplifications va s'appliquer et remplacer le littéral par $\top$ ou $\bot$ selon l'itération (et si d'autres itérations englobant m ont été insérées entre temps, alors elles seront supprimées par Constraint Splitting, éventuellement en combinaison avec Algebraic simplifications). Ainsi les seuls motifs générés au nœud d'alignement suivant sont $\top$ ou $\bot$. Donc l'ensemble des valeurs possibles est fini.
  - Si e contient une variable liée i. Alors Expansion ne peut pas s'appliquer, donc seule Unfolding le peut. Cette règle va donc substituer une variable par une expression f, et, d'après la définition de la stratégie τ, cette expression est la borne supérieure des itérations. Or cette règle ne s'applique que durant l'étape 3, dans laquelle toutes les itérations ont pour borne supérieure une expression de la forme $n - q_2 - k$ pour un certain $k \in \mathbb{N}$. Par conséquent f a la forme $n - q_2 - k$ et l'expression e ne contient que la variable n, ainsi nous nous retrouvons dans le cas précédent.
- Si $m = m_1 \amalg m_2$, où $\amalg \in \{\bigvee, \bigwedge\}$. Toute transformation sur m est le résultat d'une ou plusieurs des transformations suivantes :
  - une transformation sur m_1 produisant m'_1,
  - une transformation sur m_2 produisant m'_2,
  - une transformation à la racine produisant soit m'_1, soit m'_2 (car la seule transformation applicable à la racine est Algebraic simplifications, dont nous voyons facilement qu'elle ne peut que supprimer un des conjoints),
  - une application d'une substitution σ.

  L'application d'une substitution est directement traitée par induction, donc tout schéma issu de m a l'une des trois formes suivantes :
  - $m'_1 \amalg m'_2$ où $m_1 \rightsquigarrow^{\nu'}_{t_p} m'_1$ et $m_2 \rightsquigarrow^{\nu'}_{t_p} m'_2$,
  - m'_1 où $m_1 \rightsquigarrow^{\nu'}_{t_p} m'_1$,
  - m'_2 où $m_2 \rightsquigarrow^{\nu'}_{t_p} m'_2$.

  Alors, par hypothèse d'induction l'ensemble $\{m'_1 \mid m_1 \rightsquigarrow^{\nu'}_{t_p} m'_1\}$ est fini à un décalage près, de même que $\{m'_2 \mid m_2 \rightsquigarrow^{\nu'}_{t_p} m'_2\}$. De plus $\{m'_1 \amalg m'_2 \mid m_1 \rightsquigarrow^{\nu'}_{t_p} m'_1 \text{ et } m_2 \rightsquigarrow^{\nu'}_{t_p} m'_2\}$ est fini à un décalage près grâce au corollaire 5.25. En effet, les conditions d'applications de ce corollaire sont bien vérifiées : comme nous ne considérons que les nœuds d'alignement, nous savons que m'_1 et m'_2 sont alignés et que, de plus, n n'apparaît que dans les bornes des itérations (d'après la proposition

9.16); par conséquent, toutes les expressions contenant n sont exactement les mêmes, donc les conditions d'applications sont vérifiées. Par conséquent : $\{m'_1 \mid m_1 \leadsto^{\nu'}_{t_p} m'_1\} \cup \{m'_2 \mid m_2 \leadsto^{\nu'}_{t_p} m'_2\} \cup \{m'_1 \amalg m'_2 \mid m_1 \leadsto^{\nu'}_{t_p} m'_1 \text{ et } m_2 \leadsto^{\nu'}_{t_p} m'_2\}$ est fini à un décalage près ce qui permet de conclure (l'union finie d'ensembles finis à un décalage près est trivialement finie à un décalage près).

- Si $m = \Pi_{i=q_1}^{n-q_2-k} m_1$, $\Pi \in \{\bigvee, \bigwedge\}$. Comme dans le cas précédent, la réécriture d'un sous-motif ne peut mener qu'à un motif de la forme $\Pi_{i=q_1}^{n-q_2-k} m'_1$ où $m_1 \leadsto^{\nu'}_{t_p} m'_1$. L'application d'une substitution σ engendre donc un motif de la forme $(\Pi_{i=q_1}^{n-q_2-k} m'_1)\sigma$. Mais, comme aucune substitution appliquée à m ne substitue n, cette expression est égale à $\Pi_{i=q_1}^{n-q_2-k} m'_1\sigma$. Enfin, comme $m'_1\sigma \in \{m''_1 \mid m_1 \leadsto^{\nu'}_{t_p} m''_1\}$, cette dernière forme est en faite incluse dans la forme $\Pi_{i=q_1}^{n-q_2-k} m'_1$ où $m_1 \leadsto^{\nu'}_{t_p} m'_1$.

  Il reste à tenir compte de la réécriture de m lui-même. Seules Algebraic simplifications et Unfolding peuvent réécrire m. Dans le cas de Algebraic simplifications, il ne s'agit que de réécritures qui terminent trivialement. Donc nous nous focalisons sur Unfolding. Dans ce cas, après plusieurs applications, m peut être transformé en :

$$m'^1_1 \amalg \ldots \amalg m'^l_1 \amalg \Pi_{i=q_1}^{n-q_1-k-l} m'_1$$

  Or, d'après le corollaire 9.29, l est borné indépendamment de k. Par conséquent, m ne peut être déplié indéfiniment. De plus, chacun des $m'^1_1, \ldots, m'^l_1$ sont tels que $m_1 \leadsto^{\nu'}_{t_p} m'^1_1, \ldots, m_1 \leadsto^{\nu'}_{t_p} m'^l_1$. De même $m_1 \leadsto^{\nu'}_{t_p} m'_1$. Par conséquent tous ces motifs appartiennent à un ensemble fini à un décalage près par hypothèse d'induction. Comme dans le cas précédent, nous pouvons appliquer (l fois) le corollaire 5.25 car le motif est aligné et n n'apparaît que dans les bornes des itérations.

□

Recollement

Corollary 9.32. *$\{sL(s(\nu)) \mid \nu$ est un nœud d'alignement de $b\}$ est fini à un décalage près, avec les extensions du littéral pur et de la contrainte équivalente.*

Preuve. Il suffit d'appliquer le corollaire 5.25 aux résultats des propositions 9.26, 9.30 et 9.31. Dans chacun des cas toutes les expressions contenant n ont la forme $n - k - q$ où q appartient à un intervalle dont les bornes sont indépendantes de k. Ceci justifie donc l'application du corollaire. □

Theorem 9.33. Dpll* *avec la stratégie τ termine pour tout schéma régulier imbriqué.*

Preuve. D'après le corollaire précédent, l'ensemble des schémas générés dans une branche infinie b est fini à un décalage près avec les extensions du littéral pur et de la contrainte équivalente. Par conséquent la règle de bouclage s'applique nécessairement à un moment donné, contredisant alors le fait que la branche est infinie. Dpll* ne peut donc pas générer de branche infinie dans ce cas. □

Annexe

9.A Transformation Into Regular Schemata

Nous venons de démontrer la complétude d'une classe de schémas contenant des itérations imbriquées grâce à Dpll*. Dans [ACP10b], nous esquissons des extensions possibles de la preuve à des sous-classes plus larges, p.ex. des schémas non alignés. Plus précisément, nous autorisons qu'une variable liée soit utilisée dans la borne d'une itération : il devient possible p.ex. d'exprimer $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{i+1} p_i \vee q_j$.

Nous ne présentons pas ces résultats ici, car cela complique encore la preuve de terminaison. Il est en fait plus simple d'obtenir la décidabilité pour de tels schémas en les traduisant en schémas réguliers. Dès lors, la décidabilité des schémas réguliers permet de conclure quant à la complétude de cette classe. Nous définissons la nouvelle classe, appelée *classe des schémas mono-liés* : Cette classe est plus générale que celle des schémas réguliers imbriqués.

Definition 9.34. *Un schéma est* mono-lié *ssi :*

- *il a au plus un paramètre (noté n par la suite);*
- *chacun de ses atomes est de la forme $p_{q_1 n+\delta i+q_2}$, où i est une variable liée, $q_1, q_2 \in \mathbb{Z}$ et $\delta \in \{-1, 0, 1\}$;*
- *chacune de ses itérations est de la forme $\Pi_{i=q_3 n+q_4}^{q_1 n+\delta j+q_2} m$, où j est une variable liée, $q_1, q_2, q_3, q_4 \in \mathbb{Z}$ et $\delta \in \{-1, 0, 1\}$*

Cette classe présente deux restrictions principales : les littéraux sont monadiques, et toute expression arithmétique contient au plus une variable liée (d'où le nom "mono-lié"). De plus le coefficient de cette variable ne peut être supérieur à 1 (en effet nous savons d'après les résultats esquissés en section 3.B qu'il est facile de prouver l'indécidabilité si une variable liée peut être multipliée par 2 dans un indice, voir théorème 3.33 (i)). Tout schéma régulier est clairement mono-lié, de même que tout schéma régulier imbriqué.

Example 9.35. *Les schémas suivants sont mono-liés mais pas réguliers imbriqués :*

- $\bigwedge_{i=1}^{n} p_{n+i}$ *: le schéma n'est pas régulier imbriqué car il n'est pas à translation bornée, puisque nous ajoutons i à n;*
- $\bigwedge_{i=1}^{n} p_i \wedge \bigvee_{i=1}^{n+1} \neg p_i$ *: le schéma n'est pas aligné;*
- $\bigwedge_{i=n}^{2n} p_i \wedge \bigvee_{i=1}^{n} \neg p_i$ *: le schéma est aligné mais pas sur un intervalle de la forme $[q_1; n - q_2]$ où $q_1, q_2 \in \mathbb{Z}$;*
- $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{i} p_i \Rightarrow q_j$ *: le schéma n'est pas aligné, la borne supérieure de l'itération à l'intérieur dépend d'une variable liée.*

Les schémas suivants ne sont pas mono-liés (et, a fortiori, pas réguliers imbriqués) :

- *le principe des pigeonniers (exemple 3.2) n'est pas mono-lié car il n'est pas monadique;*
- $\bigwedge_{i=1}^{n} p_i \wedge \bigwedge_{i=1}^{l} q_i$ *: il y a deux paramètres;*
- $\bigwedge_{i=1}^{n} p_{2i}$ *: i est multiplié par 2;*
- $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} \bigvee_{k=1}^{i+j} p_i \Rightarrow q_k$ *: la borne supérieure de l'itération la plus à l'intérieur contient* deux *variables liées.*

Nous renvoyons le lecteur à [ACP10d] pour la présentation d'un algorithme permettant de traduire tout schéma mono-lié en un schéma régulier équivalent pour la satisfaisabilité.

CHAPTER 10

Related Works

À notre connaissance, il n'existe pas de travaux en déduction automatique traitant explicitement des schémas de formules. Comme nous l'avons dit dans l'introduction, la notion de schéma a longuement été étudiée en logique [Cor06]. Le sens donné aux schémas dans ces travaux est adapté à une description abstraite du concept, mais trop général pour permettre l'automatisation. Ces travaux sont donc intéressants sur le plan conceptuel, mais sont très éloignés de la façon dont nous utilisons les schémas.

Dans [KP88, Ore91] la notion de "proof schema" est explicitement utilisée, aussi appelée "proof skeleton". Cependant il ne s'agit que d'un homonyme : ces schémas de preuve n'ont rien à voir (hormis conceptuellement) avec notre sujet : d'une part, il ne s'agit pas de schémas itérés, d'autre part, il n'est pas question dans ces travaux d'automatiser le raisonnement. En effet dans ce contexte, les schémas de preuve ne sont qu'un outil permettant d'obtenir des résultats sur la taille des preuves.

Nous mentionnons aussi la "schema logic" [Fef96, FS00], qui, là aussi, n'a de commun avec notre travail que le nom : les schémas considérés sont des schémas d'axiomes. Le but n'est pas de raisonner avec ces schémas, et encore moins d'automatiser un quelconque raisonnement. Ces schémas sont utilisés dans le but d'ajouter de nouveaux axiomes à l'arithmétique.

En revanche de nombreux travaux sont proches des schémas de formules sans pour autant mettre en avant l'aspect "schéma", ce que nous présentons maintenant. Nous rappelons que notre but est d'exprimer des schémas et de prouver de façon complètement automatique que de tels schémas sont insatisfaisables ou satisfaisables.

10.1 Higher-Order Theorem Provers

Bien évidemment, les logiques d'ordre supérieur peuvent exprimer les schémas. Nous pourrions ainsi utiliser les démonstrateurs pour les logiques d'ordre supérieur afin de raisonner sur les schémas, p.ex. Coq [dt04], Isabelle/HOL [GM93], PVS [ORS92] ou Twelf [PS99]. Par exemple, le système Coq permet de définir une fonction de type $\mathsf{nat} \rightarrow \mathsf{Prop}$, c.-à-d. une fonction prenant un entier en paramètre et retournant une formule dans Coq. Pour ce faire, nous avons à notre disposition un λ-calcul très expressif, le *Calcul des Constructions Inductives* (CIC), largement suffisant pour exprimer tous les schémas présentés dans cette thèse : en effet, les fonctions décrivant les schémas terminent de façon triviale. Par exemple nous pouvons représenter le schéma $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$ par la fonction *schema* définie comme suit :

Parameter p : nat → Prop.

Fixpoint *iteration* (n:nat) :=
  match n **with**
    | O ⇒ True

```
  | S n ⇒ iteration n ∧ (p n → p (S n))
end.

Definition schema n := p O ∧ iteration n ∧ ¬p n.
```

ou encore à l'aide d'un prédicat inductif :

```
Inductive Iteration : nat → Prop :=
|Base : p O → Schema O
|Ind  : ∀ i, ((p i → p (S i)) ∧ Schema i) → Schema (S i).

Definition schema n := Schema n ∧ ¬p n.
```

Dans un cas comme dans l'autre nous démontrons facilement le théorème :

```
Theorem schema_is_unsat : ∀ n, ¬schema n.
```

Cependant, cette démonstration *ne se fait absolument pas* de façon automatique : l'utilisateur doit lui-même construire la preuve. Évidemment Coq permet d'automatiser de nombreux pas dans la construction de la preuve. Néanmoins il n'existe pas, actuellement, de tactique automatisant suffisamment la recherche de preuve pour pouvoir démontrer le théorème *schema_is_unsat*. En fait, d'une façon générale, il existe peu de tactiques permettant l'automatisation du raisonnement par induction en Coq.

Cependant, le langage de tactiques de Coq est particulièrement puissant et extensible, ce qui laisse envisager la construction d'une tactique dédiée, p.ex., aux schémas réguliers. Développer une telle tactique et assurer sa terminaison demanderait un travail théorique équivalent à celui présenté dans cette thèse, mais il est probable que le résultat soit suffisamment efficace pour être utilisable en pratique. Une autre possibilité consisterait à implémenter une procédure de preuve *dans Coq* (c.-à-d. en tant que fonction exprimée dans le Cic), p.ex. Stab ou Dpll*, et à utiliser cette procédure dans la recherche de preuve par "réflexivité". On trouvera dans [LC08] un exemple de cette méthode pour la logique propositionnelle et Dpll. Cette approche a plusieurs avantages : elle permet de *prouver formellement* la correction et la complétude d'un système de preuve (notons que celui-ci doit nécessairement être restreinte à une classe décidable et dont la décidabilité est démontrable en Coq), d'obtenir une *implémentation certifiée* (de toute fonction exprimée en Coq nous pouvons extraire une version OCaml de cette fonction, et donc une implémentation utilisable indépendamment de Coq), et d'utiliser cette même procédure pour *automatiser* la recherche de preuve dans Coq. On obtient ainsi à la fois un démonstrateur automatique et une tactique utilisable dans un assistant de preuve. Cependant, il y a probablement peu de chance que le résultat soit aussi efficace que RegStab.

Ces voies sont à approfondir, en particulier dans l'optique d'une extension des schémas au premier ordre (nécessaire pour l'utilisation des schémas dans CERES, voir section 1.2.1) où il est très peu probable que nous obtenions des procédures de décision pour des classes suffisamment expressives. Dans un tel contexte, les assistants de preuve seront certainement la seule possibilité de raisonnement formel sur les schémas. L'utilisation d'autres assistants de preuve que Coq est aussi à étudier (voir section 11.3).

10.2 Fixed-point Logics

D'une façon générale, les itérations sont des cas particuliers de points fixes. Il est donc naturel de faire le rapprochement avec les logiques à points fixes.

10.2.1 μ-calcul modal

Comme nous considérons des schémas de formules propositionnelles, le μ-calcul propositionnel (modal) [Koz83, BS07] vient naturellement à l'esprit. Ce calcul est défini en étendant la logique modale avec des opérateurs de point fixe et des actions associées aux modalités. Plus précisément, étant donnés un ensemble d'*actions* A et un ensemble de variables $\mathcal{X}$, la syntaxe du μ-calcul est donnée de la façon suivante

:

$$\Phi \quad ::= \quad \mathcal{X} \mid \Phi \wedge \Phi \mid \neg\Phi \mid [A]\Phi \mid \mu\mathcal{X}.\Phi$$

avec la restriction que, dans toute formule $\mu X.\phi \in \Phi$, la variable X apparaît sous un nombre pair de négations dans ϕ. Une *interprétation* du μ-calcul est la paire d'un système de transition avec racine dont les transitions sont étiquetées par des éléments de A et d'une fonction $\mathcal{E}$ associant à chaque variable un ensemble d'états. Dans ce contexte, l'ensemble $[\![\phi]\!]_{\mathcal{E}}$ des états satisfaisant une formule ϕ est défini par induction :

$$\begin{aligned}
[\![X]\!]_{\mathcal{E}} &\stackrel{\text{def}}{=} \mathcal{E}(X) \\
[\![\phi_1 \wedge \phi_2]\!]_{\mathcal{E}} &\stackrel{\text{def}}{=} [\![\phi_1]\!]_{\mathcal{E}} \cap [\![\phi_2]\!]_{\mathcal{E}} \\
[\![\neg\phi]\!]_{\mathcal{E}} &\stackrel{\text{def}}{=} Q \setminus \mathcal{E}(\phi) \text{ où } Q \text{ désigne l'ensemble de tous les états} \\
[\![[a]\phi]\!]_{\mathcal{E}} &\stackrel{\text{def}}{=} \{q \mid \forall q'(q \xrightarrow{a} q' \Rightarrow q' \in [\![\phi]\!]_{\mathcal{E}})\} \text{ où } a \in A \\
[\![\mu X.\phi]\!]_{\mathcal{E}} &\stackrel{\text{def}}{=} \bigcap\{Q' \subseteq Q \mid [\![\phi]\!]_{\mathcal{E}[X:=Q']} \subseteq Q'\}
\end{aligned}$$

L'interprétation est un modèle d'une formule ϕ ssi la racine du système de transition appartient à $[\![\phi]\!]_{\mathcal{E}}$. Les références [Koz83, BS07] contiennent des exemples et des explications quant à l'intuition derrière ces définitions.

Nous savons que tout schéma n'est pas exprimable en μ-calcul, car le problème de la satisfaisabilité est décidable pour ce dernier. Mais il semble possible, dans certains cas, de définir des traductions entre schémas et μ-calcul. Par exemple, soit un schéma *régulier* s de paramètre n. Supposons de plus que s soit aligné sur $[1; n]$, que tout indice d'un littéral ait la forme $i+k$ ou $n+k$ avec $k \geq 0$ et que la contrainte de s soit $n \geq 0$. Notre but est de définir une formule $|s|$ du μ-calcul telle que s a un modèle ssi $|s|$ en a un. Plus précisément nous voulons que tout modèle de $|s|$ soit constitué d'un système de transition ayant la forme $q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \ldots \xrightarrow{a} q_k$ de racine q_0 et d'une fonction $\mathcal{E}$ de sorte à ce qu'il existe un modèle $\mathfrak{M}$ de s tel que $k = \mathfrak{I}_{\text{arith}}(n)$ et $\mathcal{E}(p) = \{q_i \mid \mathfrak{I}_{\text{prop}} \models p_i\}$ (à chaque proposition indexée p_i apparaissant dans s nous associons de façon unique une variable $p \in \mathcal{X}$).

Pour cela, nous devons d'abord spécifier l'existence d'une chaîne de transitions étiquetées par a depuis la racine. Par la même occasion nous introduisons une variable N telle que N est vraie dans un état ssi cet état est le k^{th} dans la chaîne (note : en fait c'est N qui définit k, c.-à-d. que N peut potentiellement être à n'importe quel rang, mais une fois une interprétation fixée, N apparaît à un certain rang et c'est celui-ci qui définit k). Nous pouvons le faire grâce à la formule suivante :

$$\mu X.\,((\neg N \Rightarrow [a]X) \wedge (N \Rightarrow [a]\nu Y.(\neg N \wedge [a]Y))) \tag{10.1}$$

où $\nu Y.\phi$ désigne le *plus grand point fixe*, défini par $\nu Y.\phi(Y) \stackrel{\text{def}}{=} \neg\mu Y.\neg\phi(\neg Y)$ ($\phi(Y)$ signifie que Y est une variable libre de ϕ, et $\phi(\neg Y)$ signifie que $\neg Y$ est substitué à Y dans ϕ). Intuitivement, cette formule signifie que nous avons une chaîne d'actions a, que $\neg N$ est vrai pour chaque état le long de cette chaîne jusqu'à ce que nous ayons N. Dans ce cas, le deuxième point fixe impose que N est faux pour chaque état ultérieur (ce qui spécifie une infinité d'états, d'où l'utilisation du plus grand point fixe). En fait, nous avons aussi besoin de caractériser les états dont le rang est inférieur à N. Pour cela nous ajoutons une variable $N_{\leq}$ que nous spécifions comme étant vraie pour les états dont le rang est inférieur à N. Pour ce faire, nous remplaçons la formule (10.1) par :

$$\mu X.\,(N_{\leq} \wedge (\neg N \Rightarrow [a]X) \wedge (N \Rightarrow [a]\nu Y.(\neg N_{\leq} \wedge \neg N \wedge [a]Y))) \tag{10.2}$$

Nous pouvons ensuite définir notre traduction, par induction sur le motif de s (q désigne toujours un

entier naturel) :

$$
\begin{aligned}
|p_q| &\stackrel{\text{def}}{=} [a]\ldots[a]p \text{ où } [a] \text{ apparaît } q \text{ fois}\\
|p_{n+q}| &\stackrel{\text{def}}{=} \mu X.(N \Rightarrow [a]\ldots[a]p) \wedge [a]X \text{ où } [a] \text{ apparaît } q \text{ fois devant } p\\
|p_{i+q}| &\stackrel{\text{def}}{=} [a]\ldots[a]p \text{ où } [a] \text{ apparaît } q \text{ fois}\\
|m_1 \wedge m_2| &\stackrel{\text{def}}{=} |m_1| \wedge |m_2|\\
|m_1 \vee m_2| &\stackrel{\text{def}}{=} |m_1| \vee |m_2|\\
|\neg m| &\stackrel{\text{def}}{=} \neg|m|\\
|\bigwedge_{i=1}^{n} m| &\stackrel{\text{def}}{=} \mu X.(N_{\leq} \Rightarrow |m|) \wedge (\neg N_{\leq} \Rightarrow \top) \wedge [a]X\\
|\bigvee_{i=1}^{n} m| &\stackrel{\text{def}}{=} \mu X.(N_{\leq} \Rightarrow |m|) \wedge (\neg N_{\leq} \Rightarrow \bot) \vee [a]X
\end{aligned}
$$

Note. Les traductions des itérations peuvent évidemment être simplifiées en :

$$
\begin{aligned}
|\bigwedge_{i=1}^{n} m| &\stackrel{\text{def}}{=} \mu X.(N_{\leq} \Rightarrow |m|) \wedge [a]X\\
|\bigvee_{i=1}^{n} m| &\stackrel{\text{def}}{=} \mu X.(N_{\leq} \Rightarrow |m|) \wedge N_{\leq} \vee [a]X
\end{aligned}
$$

Enfin nous posons :

$$|s| \stackrel{\text{def}}{=} |m_s| \wedge (10.2)$$

Nous conjecturons que nous avons alors bien le résultat voulu, à savoir : s est satisfaisable ssi $|s|$ l'est. Plus précisément pour chaque modèle de s nous pouvons construire un modèle de $|s|$ et réciproquement.

Note. Tous les modèles de $|s|$ n'ont cependant pas nécessairement la forme mentionnée plus haut : nous pouvons avoir p.ex. un système arborescent, auquel cas c'est *chaque branche* de cette arbre qui aura bien la forme voulue. De plus, nous ne spécifions que la composante connexe de la racine. Il est évident que les autres composantes n'ont pas de conséquences sur le résultat.

Si ce résultat est correct, nous disposons alors, pour un certain type de schéma, d'une traduction vers le μ-calcul. Notons que parmi les restrictions mentionnées plus haut, l'alignement sur $[1; n]$ peut être relaxé : tout schéma régulier est aligné sur $[q_1; n - q_2]$ pour deux entiers $q_1, q_2 \in \mathbb{Z}$; il est facile de ramener un tel schéma à l'alignement voulu en décalant les indices, tout en préservant la satisfaisabilité (une transformation similaire est présentée dans [ACP10d]). Seuls les atomes dont l'indice a la forme kn constituent un véritable obstacle à la traduction de n'importe quel schéma régulier. Nous ne savons pas si toute formule du μ-calcul est traduisible dans les schémas[1]. Si c'était le cas, nous pensons que ce résultat serait intéressant car la définition des schémas nous semblent plus simple que celle du μ-calcul. Les relations précises entre ces deux formalismes restent à étudier. Elles devraient permettre d'identifier de nouvelles classes décidables pour les schémas et d'obtenir des résultats de complexité. Évidemment il serait aussi intéressant de comparer les performances des procédures de décision pour le μ-calcul sur cette traduction et celles de RegStab.

10.2.2 Logiques avec opérateurs de point fixe au premier ordre

Comme le problème de la satisfaisabilité est décidable pour le μ-calcul propositionnel, ce dernier ne permet donc pas, en général, de capturer le pouvoir d'expression des schémas. En revanche, il est possible d'étendre la logique du premier ordre avec une construction de plus petit point fixe : nous obtenons alors la "logique avec plus petit point fixe" (*LPF*) [CH82], étudiée principalement en théorie des modèles finis [Fag93]. Concrètement, nous ajoutons à la syntaxe de la logique du premier ordre des *variables de relation*, chacune munies d'une arité, et un opérateur de plus petit point fixe, très semblable

[1] Cela semble peu probable à cause de l'arborescence des modèles; c'est en revanche peut-être envisageable pour l'extension des schémas évoquée à la fin du chapitre.

à celui du μ-calcul : $\mu X(x_1, \ldots, x_k).\phi$, où X est une variable de relation, k son arité, $x_1 \ldots x_k$ sont des variables d'individus et ϕ une formule de LPF. Le terme obtenu est lui-même un prédicat d'arité k. Comme pour le μ-calcul nous imposons que X apparaisse sous un nombre pair de négations dans ϕ. La sémantique est définie de façon similaire à celle du μ-calcul [CH82]. Soit, p.ex., le prédicat *nat* suivant : $nat \stackrel{\text{def}}{=} \mu N(x).(x = 0 \vee \exists y(x = s(y) \wedge N(y)))$, alors $nat(x)$ est vrai ssi x est un entier.

Les schémas peuvent être traduits dans LPF. À partir d'un schéma s, nous pouvons construire une formule $|s|$ de LPF telle que s est satisfaisable ssi $|s|$ l'est. Tout d'abord nous définissons $|m_s|$ par induction sur la structure du motif :

$$
\begin{aligned}
|p_e| &\stackrel{\text{def}}{=} p(e) \\
|\neg\phi| &\stackrel{\text{def}}{=} \neg|\phi| \\
|\phi_1 \wedge \phi_2| &\stackrel{\text{def}}{=} |\phi_1| \wedge |\phi_2| \\
|\phi_1 \vee \phi_2| &\stackrel{\text{def}}{=} |\phi_1| \vee |\phi_2| \\
|\bigwedge_{i|\delta} \phi| &\stackrel{\text{def}}{=} [\mu X(i).(\delta \Rightarrow |\phi|) \wedge (\neg\delta \Rightarrow \top) \wedge X(i-1)](f) \\
|\bigvee_{i|\delta} \phi| &\stackrel{\text{def}}{=} [\mu X(i).(\delta \Rightarrow |\phi|) \wedge (\neg\delta \Rightarrow \bot) \vee X(i-1)](f)
\end{aligned}
$$

où f est défini tel que $\delta \models i \leq f$ (f existe car, par définition, tout domaine est encadré, voir p.17). Nous définissons ensuite $|s|$ comme $|m_s| \wedge c_s \wedge Lin$ où Lin désigne les axiomes de l'arithmétique linéaire. Il est alors facile de voir que nous avons effectivement la propriété voulue, à savoir : s est satisfaisable ssi $|s|$ l'est. Notons que nous avons pu expérimenter cette traduction dans l'assistant de preuve Tac [BMS10], (`http://slimmer.gforge.inria.fr/tac`) qui implémente une variante de LPF.

Le problème de la satisfaisabilité est évidemment indécidable pour LPF. Cette logique est surtout étudiée en complexité descriptive [Imm82]. Par conséquent peu de travaux se sont intéressés à l'identification de classes décidables de LPF. À notre connaissance, seuls les travaux présentés dans [Bae08, Bae09] étudient le raisonnement automatique dans de telles logiques. En particulier, il est prouvé dans [Bae09] la décidabilité de la prouvabilité pour une classe de formules appelées *formules régulières*. Nous ne donnons pas leur définition précise, mais uniquement une version simplifiée de leur "propriété fondamentale" qui donne plus d'informations sur leur forme. Cette propriété est la suivante. Tout point fixe $\mu X(x).\phi$ apparaissant dans une formule régulière est équivalent à :

$$\exists y(x = C[y]) \vee \exists y(x = C_1[y] \wedge \phi_1(y)) \vee \ldots \vee \exists y(x = C_k[y] \wedge \phi_k(y))$$

où $C, C_1, \ldots, C_k$ sont des contextes linéaires sans variable[2] et $\phi_1, \ldots, \phi_k$ sont des formules régulières ou X. Dans la véritable définition, un point fixe peut avoir plus d'une variable libre et la logique utilisée est la logique linéaire [Gir06, GLT89].

Nous ne connaissons pas de classes non triviales de schémas qui soient capturées par les formules régulières. Notons tout d'abord que la traduction des schémas réguliers vers LPF décrite ci-dessus n'est jamais une formule régulière, en effet, les seuls atomes autorisés dans ces formules sont des égalités ou des variables de prédicats liées par des opérateurs de points fixes : il est impossible d'avoir des variables de prédicat libres. Ainsi, même la traduction d'un littéral n'est pas une formule régulière. Cependant nous pouvons, comme c'est couramment le cas, exprimer un atome $p(n)$ comme une égalité $p(n) = \top$, en faisant de p un symbole de fonction plutôt qu'un symbole de prédicat et de $\top$ une constante. Mais même dans ce cas, la forme très particulière des formules régulières empêche de définir le moindre schéma non trivial. Par exemple essayons de traduire le schéma $\bigwedge_{i=1}^{n} p_i$:

$$\mu X(n).n = 0 \vee \exists m(n = m + 1 \wedge \underline{p(n) = \top} \wedge X(m))$$

La sous-formule soulignée ($p(n) = \top$) n'a pas la forme $n = C[m]$, quel que soit C. En fait il est uniquement possible de décomposer les termes passés en paramètre (ici n) dans un point fixe : nous ne pouvons donc pas ajouter de contexte autour de ces termes.

Nous ne voyons pas d'autre traduction possible des schémas vers les formules régulières. Cependant la technique utilisée dans [Bae09] pour démontrer la décidabilité de la prouvabilité des formules régulières

[2] C'est-à-dire des termes clos avec un "trou"; $C[y]$ désigne le contexte C dans lequel le trou est remplacé par y.

est originale, de plus ces travaux reposent sur de solides fondations en théorie de la preuve. Il serait donc probablement fructueux d'essayer d'étendre ces techniques au cas des schémas. Réciproquement nos travaux pourraient permettre de caractériser d'autres classes décidables de la logique du premier ordre avec points fixes.

10.3 Inductive Theorem Provers

La section précédente regroupe tous les formalismes qui, à notre connaissance, présentent une similarité syntaxique évidente avec les schémas. Mais il y a une autre possibilité : traduire les schémas en logique du premier ordre (sans point fixe), ce qui permettrait d'utiliser les démonstrateurs existants. Évidemment nous ne pourrons pas ainsi capturer toute la logique des schémas puisque la logique du premier ordre est complète et celle des schémas ne l'est pas. Mais, de toute manière, même en définissant une procédure dédiée aux schémas, nous ne pourrons jamais capturer que partiellement le raisonnement sur les schémas, du fait de l'incomplétude de leur logique. Nous définissons deux traductions différentes.

10.3.1 Une première traduction en logique du premier ordre

Une première possibilité, la plus intuitive, consiste à traduire les itérations en quantifications bornées. Pour tout schéma s, nous définissons la formule $|s|_1$ t.q. si $|s|_1$ est insatisfaisable, alors s l'est (évidemment nous n'avons pas l'implication réciproque car la satisfaisabilité des schémas serait alors décidable). Nous définissons d'abord $|m_s|_1$ par induction sur la structure du motif :

$$
\begin{aligned}
|p_{e_1,\ldots,e_k}|_1 &\stackrel{\text{def}}{=} p(e_1,\ldots,e_k)\\
|\neg\phi|_1 &\stackrel{\text{def}}{=} \neg|\phi|_1\\
|\phi_1 \wedge \phi_2|_1 &\stackrel{\text{def}}{=} |\phi_1|_1 \wedge |\phi_2|_1\\
|\phi_1 \vee \phi_2|_1 &\stackrel{\text{def}}{=} |\phi_1|_1 \vee |\phi_2|_1\\
|\bigwedge_{i|\delta} \phi|_1 &\stackrel{\text{def}}{=} \forall i(\delta \Rightarrow |\phi|_1)\\
|\bigvee_{i|\delta} \phi|_1 &\stackrel{\text{def}}{=} \exists i(\delta \wedge |\phi|_1)
\end{aligned}
$$

Puis $|s|_1$ est défini comme $\exists n_1,\ldots,n_k(|m_s|_1 \wedge c_s \wedge Lin)$ où $n_1,\ldots,n_k$ sont les paramètres de s et Lin désigne une axiomatisation de l'arithmétique linéaire (une telle axiomatisation ne pose pas de problème en théorie mais ne semble pas raisonnable en pratique). Notons bien que $|s|_1$ n'appartient pas à l'arithmétique linéaire à cause des propositions indexées qui deviennent des prédicats (non interprétés). Il est alors facile de montrer que s est satisfaisable ssi $\mathbb{Z}$ (avec ses opérations habituelles) est un modèle de $|s|_1$. Nous avons alors bien le résultat que si $|s|_1$ est insatisfaisable alors s l'est. En revanche, il se peut que s soit insatisfaisable mais que $|s|_1$ ait un modèle (nécessairement non standard).

Note. Il est possible et justifié de considérer que transformer les itérations en quantifications fait perdre la *structure* originale du problème, ce qui ôte de l'intérêt à cette traduction.

Certains auteurs ont défini des calculs combinant une procédure de preuve générique pour la logique du premier ordre [BGW94] avec des théories particulières telles que l'arithmétique linéaire [AKW09]. Il est clair que nous pouvons utiliser les procédures de preuve ainsi définies pour réfuter les formules obtenues par traduction (au lieu de considérer une axiomatisation de l'arithmétique). Cependant la formule obtenue par traduction ne rentre pas dans les classes pour lesquelles la terminaison, ou même la complétude, de ces procédures de preuve peut être assurée. Le pouvoir d'expression de ces théories semble beaucoup trop faible pour exprimer les schémas (même avec une autre traduction) : le critère de "sufficient completeness" donné dans [BGW94] et utilisé par [AKW09] est particulièrement restrictif; il empêche notamment d'utiliser des variables quantifiées existentiellement sauf s'il est possible de prouver que celles-ci sont toujours égales à un terme de la théorie. Ce n'est pas le cas ici, car la condition $n \in \mathbb{N}$, où n est un paramètre entier, est une donnée du problème, et non une conséquence des formules considérées. Cependant cette piste est à étudier plus en détail car la combinaison de théories est un sujet de recherche particulièrement actif (voir, en particulier : [KV07, BLdM09]).

10.3.2 Les démonstrateurs inductifs

Il se peut que s soit insatisfaisable mais que $|s|_1$ ait un modèle (nécessairement non standard). Idéalement, nous voudrions pouvoir restreindre le raisonnement au modèle standard. Évidemment, nous savons que c'est impossible, cependant nous pouvons éviter de considérer tous les modèles possibles. Typiquement, nous pouvons ne considérer que les interprétations dont le domaine est restreint au domaine de Herbrand, c.-à-d. à l'ensemble des termes clos. La notion de conséquence considérée alors est appelée *conséquence inductive*. Notons que, en général, la conséquence inductive n'est même pas semi-décidable : non seulement, comme pour la conséquence déductive, il n'y a pas complétude pour la satisfaisabilité, mais en plus, il n'y a pas complétude réfutationnelle. Les démonstrateurs permettant de raisonner par induction sont appelés des *démonstrateurs inductifs*.

Ces derniers sont, pour la plupart, conçus pour prouver des conjectures de la forme :

$$\forall x.\phi \qquad \text{où } \phi \text{ est une formule } \textit{sans quantificateurs} \tag{10.3}$$

De plus les seuls atomes généralement acceptés dans ϕ sont des égalités. Tous ces démonstrateurs sont conçus pour raisonner avec des fonctions définies par induction.

La démonstration de conjectures inductives contenant des quantifications existentielles est connu pour être un problème difficile. Notons que, contrairement à la plupart des démonstrateurs au premier ordre, ces systèmes cherchent bien à *prouver* la conjecture, et non pas à la *réfuter*. Par conséquent, pour utiliser ces techniques, nous considérons $\neg|s|_1$ au lieu de $|s|_1$. Ainsi la traduction est bien préfixée par des quantificateurs universels, ce qui est en accord avec 10.3. Cependant, il suffit qu'un schéma s contienne une seule conjonction itérée pour que $|s|_1$ contienne une quantification existentielle, ce qui exclut beaucoup de schémas. À notre connaissance, seul [HW09] identifie une classe décidable incluant des quantificateurs existentiels. Plus précisément il est prouvé la décidabilité du problème :

$$N \models_{\text{ind}} \forall x \exists y_1, \ldots, y_n (A_1 \wedge \ldots \wedge A_k)$$

où $A_1, \ldots, A_k$ sont des atomes et N est un ensemble de clauses de Horn finiment saturable par résolution (sans avoir le droit de considérer d'autre notion de subsomption que l'égalité à un renommage près). Il est facile de voir que, hormis pour des schémas triviaux, la traduction ne peut pas rentrer dans ce cadre extrêmement restrictif : quelle que soit la façon dont la traduction est modifiée, il est quasi impossible d'obtenir simultanément un ensemble de clauses qui soit de Horn et qui ait la propriété de saturation évoquée. Même l'exemple récurrent $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$ est difficile à exprimer sans des transformations conséquentes.

10.3.3 Une deuxième traduction

Nous donnons donc une deuxième traduction $|\text{-}|_2$, plus en adéquation avec ces considérations. L'idée informelle est que les schémas vont être traduits en *termes* dont le type est la sorte des booléens et dont les variables libres sont des entiers (correspondant aux paramètres du schéma). Ces termes sont composés de symboles de fonctions possiblement définis récursivement. Le schéma est alors insatisfaisable ssi le terme obtenu est égal à $\bot$ pour tout valeur des variables libres ($\bot$ est ici une constante). Ainsi, si t_s désigne le terme obtenu à partir d'un schéma s, alors s est insatisfaisable ssi $\forall n_1, \ldots, n_k (t_s = \bot)$ est valide, où $n_1, \ldots, n_k$ désignent les paramètres du schéma. Cette formule a bien la forme 10.3, ses seuls atomes sont des égalités et t_s est composé de symboles de fonctions définies récursivement. Nous donnons maintenant la définition de cette traduction.

Note. Comme nous passons du niveau des formules à celui des termes, nous ne pouvons pas traduire des itérations dont le domaine est une formule quelconque de l'arithmétique linéaire. Par conséquent nous nous restreignons aux schémas syntaxiquement et linéairement encadrés. De plus, la contrainte du schéma est laissée telle quelle par la traduction donc nous imposons aussi que cette contrainte ne contienne que des quantificateurs universels. Ceci n'est pas restrictif puisque nous pouvons toujours appliquer l'élimination des quantificateurs sur la contrainte.

Comme pour $|\text{-}|_1$, nous définissons d'abord $|\text{-}|_2$ par induction sur la structure du motif. $|\text{-}|_2$ doit retourner à la fois la traduction du schéma d'origine et les équations définissant (récursivement) les symboles composant cette traduction. Ainsi $|\text{-}|_2$ prend en entrée la paire d'un motif et d'un ensemble $\mathcal{E}$

d'équations, initialement égal à l'ensemble vide :

$$
\begin{aligned}
|p_{e_1,\ldots,e_k}| &\stackrel{\text{def}}{=} (p(e_1,\ldots,e_k),\emptyset)\\
|\neg\phi| &\stackrel{\text{def}}{=} (\tilde{\neg}\phi',\mathcal{E}') \text{ où } (\phi',\mathcal{E}') = |\phi|_2\\
|\phi_1 \wedge \phi_2| &\stackrel{\text{def}}{=} (\phi_1'\tilde{\wedge}\phi_2', \mathcal{E}_1 \wedge \mathcal{E}_2) \text{ où } (\phi_1',\mathcal{E}_1) = |\phi_1|_2 \text{ et } (\phi_2',\mathcal{E}_2) = |\phi_2|_2\\
|\phi_1 \vee \phi_2| &\stackrel{\text{def}}{=} (\phi_1'\tilde{\vee}\phi_2', \mathcal{E}_1 \wedge \mathcal{E}_2) \text{ où } (\phi_1',\mathcal{E}_1) = |\phi_1|_2 \text{ et } (\phi_2',\mathcal{E}_2) = |\phi_2|_2\\
|\bigwedge_{i=e}^{f}\phi| &\stackrel{\text{def}}{=} \left(g(f), \mathcal{E}' \wedge \left\{\begin{array}{l} g(e-1) = \top \\ g(i+1) = \phi'\tilde{\wedge}g(i)\end{array}\right\}\right) \text{ où } (\phi',\mathcal{E}') = |\phi|_2\\
|\bigwedge_{i=e}^{f}\phi| &\stackrel{\text{def}}{=} \left(g(f), \mathcal{E}' \wedge \left\{\begin{array}{l} g(e-1) = \bot \\ g(i+1) = \phi'\tilde{\vee}g(i)\end{array}\right\}\right) \text{ où } (\phi',\mathcal{E}') = |\phi|_2
\end{aligned}
$$

Notons que les symboles $\tilde{\neg}$, $\tilde{\vee}$ et $\tilde{\wedge}$ sont des symboles de fonctions, et non pas des connecteurs logiques. De même p est un symbole de fonction et pas un symbole de prédicat. Ensuite, $|s|_2$ est défini comme :

$$|s|_2 \stackrel{\text{def}}{=} \forall n_1,\ldots,n_k(\phi = \bot \wedge \mathcal{E} \wedge Lin \wedge Prop \wedge c_s)$$

où :

- $n_1,\ldots,n_k$ sont les paramètres de s,
- $(\phi,\mathcal{E}) = |m_s|_2$,
- *Lin* désigne une axiomatisation de l'arithmétique linéaire,
- et *Prop* désigne une axiomatisation de $\neg$, $\vee$ et $\wedge$ en tant que symboles de fonction.

Alors un schéma s est insatisfaisable ssi $|s|_2$ est valide.

Note. Contrairement à la traduction précédente, celle-ci préserve la *structure* originale du problème.

10.3.4 Survol des différents démonstrateurs inductifs

Les démonstrateurs automatiques utilisent de nombreuses techniques différentes : "induction explicite" [Bun01], "induction sans induction" (ou "preuve par consistance") [Com01], "induction implicite". Nous analysons brièvement chacune d'entre elles.

Induction explicite

Tout d'abord les techniques d'*induction explicite* [Bun01, Wir99], dont la plupart sont aujourd'hui subsumées par le "rippling" [BSvH+93], sont surtout utilisées dans le contexte d'assistants de preuve : elles sont principalement basées sur des heuristiques destinées à *améliorer* l'automatisation mais pas à obtenir une automatisation complète. En particulier, leur but n'est pas d'identifier des classes décidables mais plutôt de faciliter la tâche de l'utilisateur. Ces travaux sont donc très différents des nôtres. Cependant, comme il a été dit pour les démonstrateurs d'ordre supérieur, il est peu probable que nous puissions identifier des classes décidables dans le contexte, p.ex., des schémas au premier ordre. Ainsi l'extension de la portée des schémas passera probablement par ce type d'outil. Notons que, en tout cas, le travail effectué dans cette thèse va permettre de mieux étudier l'application de ces heuristiques aux schémas.

Nous avons tout de même mené des expérimentations avec ACL2 [BM79], connu pour son efficacité dans ce domaine. La théorie d'ACL2 est une théorie du premier ordre dont le seul prédicat est l'égalité et dont les termes sont un sous-ensemble strict de Common LISP (en particulier les fonctions d'ordre supérieur ne sont pas autorisées). Vues ces restrictions, nous ne pouvons pas appliquer la deuxième traduction telle quelle : les propositions indexées deviennent des symboles de fonctions non définis (c.-à-d. que nous devons, en fait, quantifier *universellement* sur ces symboles), il faut donc les passer en paramètre à la fonction représentant le schéma. Mais comme les fonctions d'ordre supérieur sont interdites, ceci est impossible. La solution consiste à représenter les propositions indexées par des listes (listes de listes si la proposition est d'arité 2, listes de listes de listes si elle est d'arité 3, etc.). Considérons p.ex. le schéma

$s = p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n$. On peut définir ce schéma dans ACL2 :

(**defun** *sch* (n p)
  (**if** ((**integerp** n) $\wedge$ ($n \geq 0$) $\wedge$ ((**len** p) $\geq$ (n+2)))
    (**if** (n <= 0)
      $\top$
      (((**nth** n p) $\Rightarrow$ (**nth** (n+1) p)) $\wedge$ (*sch1* (n-1) p)))
    nil))

(**defthm** *sch-unsat* (((**nth** 1 p) $\wedge$ (*sch* n p) $\wedge$ ($\neg$(**nth** (n+1) p))) = $\bot$))

---

où **nil**, **len**, **nth** désignent respectivement la liste vide, la fonction calculant la longueur d'une liste et la fonction prenant un nombre n et une liste et retournant le n^{th} élément de la liste; **integerp** n permet juste d'assurer que n est un entier. ACL2 a réussi à démontrer la conjecture *sch-unsat*. Toutefois, il s'agit du seul exemple que ACL2 a réussi à prouver dans un temps raisonnable parmi tous ceux prouvés par RegStab (p.ex. l'additionneur, voir chapitre 7). Pour tous les autres, ACL2 n'a pas terminé, malgré un délai bien supérieur à celui accordé à RegStab. Peut-être aurait-il pu terminer si plus de temps lui avait été accordé. Par conséquent nous ne pouvons pas dire qu'ACL2 soit capable de les prouver mais nous ne pouvons pas dire non plus qu'il en soit incapable. Peut-être un expert d'ACL2 aurait su exprimer la conjecture de façon à ce que le système la démontre. Ceci illustre bien les problèmes que posent la semi-décidabilité. A contrario, RegStab termine tout le temps. Ainsi, bien que théoriquement possible, cette technique ne semble guère utilisable en pratique. De plus, nous pensons que ces expérimentations justifient que nous valorisions à ce point l'obtention de classes décidables.

Induction sans induction

En *induction sans induction*, nous nous intéressons uniquement à la validité dans le plus petit modèle de Herbrand $\mathfrak{M}$ (seuls des ensembles de clauses de Horn sont considérés, assurant ainsi l'existence de ce plus petit modèle), ce qui équivaut à la validité inductive si nous nous restreignons aux clauses *positives* (voir [Com01]). Avec cette méthode, nous ajoutons à la conjecture ϕ une axiomatisation $\mathcal{A}$ de $\mathfrak{M}$. Si $\mathcal{A}$ caractérise effectivement $\mathfrak{M}$, alors ϕ est valide dans $\mathfrak{M}$ ssi $\phi \cup \mathcal{A}$ est satisfaisable. Des démonstrateurs classiques sont alors utilisés pour saturer l'ensemble de clauses : s'il y a terminaison et que l'ensemble obtenu est cohérent, alors la conjecture est vraie dans $\mathfrak{M}$. Cette technique a pour avantage la possibilité d'utiliser des démonstrateurs existants, de plus il y a terminaison si la conjecture est fausse.

Contrairement aux méthodes d'induction explicite, l'induction sans induction a pour but l'automatisation complète. En revanche, une fois encore, il est difficile d'obtenir des classes décidables. Une façon simple d'assurer cela est que ϕ et $\mathcal{A}$ appartiennent à une classe décidable connue de la logique du premier ordre. Cependant, entre la traduction des schémas, la mise sous forme clausale et la restriction à une classe décidable de la logique du premier ordre, cette piste ne semble pas raisonnable en pratique, même si elle mériterait probablement de plus longues investigations. Une fois encore, il nous semble plus facile d'étudier l'éventuel intérêt de cette méthode dans le cadre des schémas *maintenant que nous avons définies et examiné des méthodes directes*. En particulier, il est peut-être possible de considérer la détection de cycle comme une forme de détection de redondance permettant d'assurer une saturation finie.

Enfin les méthodes par induction implicite (typiquement : cover set [Zha88], test set [BKR92], term rewriting induction [Red90]) ont des propriétés très similaires à l'induction sans induction, du moins pour ce qui nous concerne : automatisation complète, mais absence de classe décidable.

Classes décidables

Parmi les méthodes d'induction implicite, il existe tout de même quelques résultats de décidabilité [KS00, GK01, FK06] (implémentés, semble-t-il, dans *RRL*, qui n'est plus disponible sur le web). Les formules de ces classes ont pour forme :

$$\forall x_1, \ldots x_k (f(x_1, \ldots, x_k) = t)$$

où t est un terme et f est une fonction définie récursivement, ce qui est bien conforme à la traduction $|\text{-}|_2$ (les axiomatisations de l'arithmétique linéaire et des connecteurs logiques sont définis par ailleurs et ne posent pas de problème par rapport à la décidabilité). La restriction majeure qui autorise la décidabilité

est le fait que tous les symboles apparaissant dans la définition d'une fonction récursive doivent être des symboles *interprétés* d'une théorie décidable. Par exemple nous pouvons définir la fonction :

$$double(0) \stackrel{\text{def}}{=} 0$$
$$double(s(x)) \stackrel{\text{def}}{=} s(s(double(x)))$$

où 0 et s sont interprétés dans l'arithmétique de Presburger. En revanche, la fonction f suivante ne rentre pas dans cette classe :

$$f(0,n) \stackrel{\text{def}}{=} \top$$
$$f(i+1,n) \stackrel{\text{def}}{=} (\tilde{\neg}p(i)\tilde{\vee}p(i+1))\tilde{\wedge}f(i,n)$$

car p n'est pas un symbole interprété. Comme pour ACL2 nous résolvons le problème en représentant les propositions indexées par des listes.

Ceci ne résout le problème que pour les propositions indexées. En effet, dans notre traduction $|\text{-}|_2$, *toute itération est traduite en une fonction*. Par conséquent la traduction du schéma fait appel à autant de fonctions qu'il y a d'itérations. En fait, f peut contenir des symboles non interprétés mais à condition qu'ils soient définis, ce qui est bien le cas des fonctions issues de la traduction d'itérations (mais pas de celles issues des propositions indexées; il faut donc bien conserver la traduction des propositions indexées sous forme de listes). Dans ce cas, si la définition de f utilise les symboles définis $f_1, \ldots, f_n$, alors la conjecture doit avoir la forme suivante :

$$\forall x_1, \ldots, x_{n,k_n}\,(f(x_1, \ldots, x_k) = t \wedge f_1(x_{1,1}, \ldots, x_{1,k_1}) = t_1 \wedge \ldots \wedge f_n(x_{n,1}, \ldots, x_{n,k_n}) = t_n)$$

Autrement dit, nous sommes obligés de donner un lemme intermédiaire pour chaque fonction utilisée par f, c.-à-d., dans notre contexte, pour chaque itération. Nous sommes donc obligés de donner des lemmes intermédiaires correspondant à la *valeur* de chaque itération. Ceci pose non seulement un problème vis-à-vis de l'automatisation car cela signifie que nous devons trouver cette valeur, mais en plus *ces itérations n'ont pas nécessairement une valeur constante* : une itération peut être vraie dans une certaine interprétation mais fausse dans une autre. Seul le schéma entier a pour valeur false quel que soit l'interprétation (s'il est insatisfaisable).

Dans le cas des schémas réguliers, nous pouvons dans certains cas, nous ramener à un schéma ne contenant qu'une itération. Auquel cas la traduction ne contient qu'une fonction et appartient bien à la classe décidable. En revanche, pour les schémas réguliers en général, et a fortiori pour les schémas réguliers imbriqués, la traduction n'appartient pas à la classe. Il nous semble impossible de contourner cette limitation. En revanche il pourrait être intéressant d'essayer de traduire nos résultats de décidabilité dans le contexte de ces travaux, ce qui pourrait permettre de caractériser de nouvelles classes décidables pour la validité inductive en logique du premier ordre.

10.4 Cyclic Proofs

Comme nous l'avons vu au Chapitre 5, la *détection de cycles* dans la recherche de preuve est un point essentiel dans la terminaison des procédures sur les schémas. C'est en fait une forme d'induction implicite. Cette technique n'est pas nouvelle en déduction automatique, elle est standard p.ex. dans les logiques modales avec cadres transitifs (p.ex. S4, S5), où il est nécessaire de détecter si une éventualité peut se réaliser, voir [Gor99]. On trouve les même idées dans des méthodes à base de tableaux pour le μ-calcul [Cle90].

Cependant la détection de cycle mise en œuvre avec les schémas est assez différente. D'abord il n'y a aucune garantie qu'un cycle soit toujours détecté : en effet nous pouvons générer une quantité infinie de schémas tous différents, et dont l'ensemble n'est pas exprimable de façon finie. Ce n'est pas le cas pour les autres logiques évoquées dont nous pouvons facilement prouver que l'ensemble des "états" possibles est fini. Autrement dit, il y aura toujours un cycle, ce qui n'est pas le cas avec les schémas. On peut, partiellement, contourner cette situation en considérant qu'un cycle a lieu non en cas d'égalité mais en cas d'équivalence comme vu au chapitre 5. Cependant, dû à l'indécidabilité, aucune notion d'équivalence

n'assurera la terminaison dans tous les cas. En fait, comme cet exemple le laisse deviner, la détection de cycles dans le cas des schémas se substitue à une démonstration par récurrence. C'est ce qui constitue la différence principale avec les méthodes évoquées ci-dessus.

Dans ce contexte, des "preuves cycliques" ont été étudiées en théorie de la preuve [Bro06] : un calcul à base de séquents est défini où, contrairement aux systèmes habituels, les règles traitant de l'induction sont de simples règles de distinction de cas (en particulier il n'y a pas d'invariant). Ainsi les preuves sont en général des objets infinis. Un critère global est alors nécessaire pour assurer la correction de la preuve. Un avantage des preuves cycliques par rapport aux preuves par induction classique est qu'il n'y a pas besoin de chercher un invariant, ce qui les rend particulièrement adaptées à l'automatisation. De plus [Bro05] et [SD03] montrent que les preuves cycliques sont aussi puissantes que les preuves par induction dans de nombreux cas. Toutefois ces travaux restent théoriques et ne sont pas utilisés, à notre connaissance, en déduction automatique. Il s'agit, selon nous, d'une des contributions originales de cette thèse que de considérer des preuves par induction avec détection de cycle en déduction automatique. Notons enfin que, les preuves cycliques étant étudiées en théorie de la preuve, la notion de cycle considérée est une notion *syntaxique* alors que notre notion de cycle est définie de façon *sémantique*. L'étude des liens précis entre les deux formalismes pourrait nous permettre de considérer une notion syntaxique de cycle pour les schémas. Enfin [Bro06] étudie des connexions entre preuves cycliques et automates ce qui n'est pas sans rappeler le lien entre Stab et SchAut.

10.5 Inductive Boolean Functions

Dans [Gup94], Gupta étudie des "fonctions booléennes inductives" dans le but de modéliser des circuits de façon inductive, exactement comme nous l'avons fait avec les schémas réguliers. Il s'agit, à notre connaissance, des travaux les plus proches des schémas de formules. Deux classes de fonctions définies récursivement sont introduites : les *LIF* pour "Linearly Inductive Functions" et les *EIF* pour "Exponentially Inductive Functions". Nous étudions d'abord les LIF.

10.5.1 Fonctions linéairement inductives

Definition 10.1. *Notons $\mathcal{P}rop(X)$ l'ensemble des formules propositionnelles dont les atomes appartiennent à l'ensemble X. Une* fonction linéairement inductive *f de paramètre n est une fonction récursive définie de la façon suivante :*

$$f(n) \stackrel{def}{=} \begin{cases} \phi_1 \in \mathcal{P}rop(P_1) & si\ n = 1 \\ \phi_n \in \mathcal{P}rop(P_n \cup F_{n-1}) & sinon \end{cases}$$

où P_1 (resp. P_n) est un ensemble de propositions indexées d'indice 1 (resp. n) et F_{n-1} est un ensemble de LIF (pouvant contenir f) ayant $n-1$ pour paramètre.

Contrairement aux schémas de formules, nous pouvons itérer sur *n'importe quelle fonction* et pas seulement des conjonctions ou disjonctions itérées. En revanche, les propositions indexées ne sont indicées que par 1 ou le paramètre, il est impossible de référer à la valeur d'une proposition dans "un autre rang" sauf via l'appel à une autre LIF. Notons que seules des propositions monadiques sont considérées.

Nous donnons une traduction des LIF vers les schémas réguliers. Soit une LIF f. À toute LIF g apparaissant dans la définition de f (f comprise), nous associons une proposition indexée $\|g\|_i$. Soient de plus ϕ_1 et ϕ_n les formules telles que :

$$g(n) \stackrel{\text{def}}{=} \begin{cases} \phi_1 & \text{si } n = 1 \\ \phi_n & \text{sinon} \end{cases}$$

nous définissons le schéma régulier $|g|$ de la façon suivante :

$$|g| \stackrel{\text{def}}{=} (\|g\|_1 \Leftrightarrow \phi_1) \wedge \bigwedge_{i=2}^{n} \|g\|_i \Leftrightarrow \phi_n[\forall h, \|h\|_{i-1}/h(n-1)]$$

La traduction de f est alors le schéma régulier suivant :

$$\|f\|_n \wedge |g_1| \wedge \ldots \wedge |g_k| \,\&\, n \geq 1$$

où $g_1, \ldots, g_k$ sont les LIF apparaissant dans la définition de f.

Nous nous intéressons maintenant à la réciproque, c.-à-d. traduire les schémas réguliers en LIF. Pour simplifier la traduction, nous nous restreignons aux schémas réguliers ayant les propriétés suivantes :

- aligné sur $[2; n-1]$,
- toute proposition indexée apparaissant dans une itération $\Pi_{i=1}^{n}\phi$ a pour indice i, $i-1$ ou 1,
- toute proposition indexée apparaissant en dehors d'une itération a pour indice 1 ou $n-1$,
- la contrainte est $n \geq 2$.

Par exemple, le schéma $p_1 \wedge \bigwedge_{i=2}^{n-1} p_{i-1} \Rightarrow p_i \wedge \neg p_{n-1} \,\&\, n \geq 1$ a bien la forme voulue (ce schéma est très proche de $p_0 \wedge (\bigwedge_{i=0}^{n-1} p_i \Rightarrow p_{i+1}) \wedge \neg p_n \,\&\, n \geq 0$). Pour un tel schéma s, nous définissons alors $|m_s|$ par induction sur le motif :

$$|p_1| \stackrel{\text{def}}{=} f(n-1) \text{ où } f \text{ est définie par } f(n) \stackrel{\text{def}}{=} \begin{cases} p_1 & \text{si } n = 1 \\ f(n-1) & \text{sinon} \end{cases}$$

$$|p_i| \stackrel{\text{def}}{=} p_i \text{ si } i \text{ est liée}$$

$$|p_{i-1}| \stackrel{\text{def}}{=} f(n-1) \text{ où } f \text{ est définie par } f(n) \stackrel{\text{def}}{=} \begin{cases} p_1 & \text{si } n = 1 \\ p_n & \text{sinon} \end{cases}$$

$$|p_{n-1}| \stackrel{\text{def}}{=} f(n-1) \text{ où } f \text{ est définie par } f(n) \stackrel{\text{def}}{=} \begin{cases} p_1 & \text{si } n = 1 \\ p_n & \text{sinon} \end{cases}$$

$$|\neg\phi| \stackrel{\text{def}}{=} \neg|\phi|$$

$$|\phi_1 \wedge \phi_2| \stackrel{\text{def}}{=} |\phi_1| \wedge |\phi_2|$$

$$|\phi_1 \vee \phi_2| \stackrel{\text{def}}{=} |\phi_1| \vee |\phi_2|$$

$$|\bigwedge_{i=2}^{n-1} \phi| \stackrel{\text{def}}{=} f(n-1) \text{ où } f \text{ est définie par } f(n) \stackrel{\text{def}}{=} \begin{cases} \top & \text{si } n = 1 \\ |\phi| \wedge f(n-1) & \text{sinon} \end{cases}$$

$$|\bigvee_{i=2}^{n-1} \phi| \stackrel{\text{def}}{=} f(n-1) \text{ où } f \text{ est définie par } f(n) \stackrel{\text{def}}{=} \begin{cases} \bot & \text{si } n = 1 \\ |\phi| \vee f(n-1) & \text{sinon} \end{cases}$$

Puis $|s|$ est défini comme la LIF f, elle-même définie de la façon suivante :

$$f(n) \stackrel{\text{def}}{=} \begin{cases} \langle s \rangle_{\{n \mapsto 1\}} & \text{si } n = 1 \\ |m_s| & \text{sinon} \end{cases}$$

Nous avons alors la propriété : pour tout $N \geq 2$, $\langle s \rangle_{\{n \mapsto N\}} = f(N)$.

Par exemple, le schéma $p_1 \wedge \bigwedge_{i=2}^{n-1} p_{i-1} \Rightarrow p_i \wedge \neg p_{n-1} \,\&\, n \geq 1$ est traduit en $f(n)$, où :

$$f(n) \stackrel{\text{def}}{=} \begin{cases} p_1 \wedge \neg p_1 & \text{si } n = 1 \\ f_1(n-1) \wedge f_2(n-1) \wedge \neg f_3(n-1) & \text{sinon} \end{cases}$$

$$f_1(n) \stackrel{\text{def}}{=} \begin{cases} p_1 & \text{si } n = 1 \\ f_1(n-1) & \text{sinon} \end{cases}$$

$$f_2(n) \stackrel{\text{def}}{=} \begin{cases} \top & \text{si } n = 1 \\ (f_3(n-1) \Rightarrow p_n) \wedge f_2(n-1) & \text{sinon} \end{cases}$$

$$f_3(n) \stackrel{\text{def}}{=} \begin{cases} p_1 & \text{si } n = 1 \\ p_n & \text{sinon} \end{cases}$$

Il est alors facile de vérifier que :

$$f(2) = p_1 \wedge \top \wedge \neg p_1$$
$$f(3) = p_1 \wedge \top \wedge (p_1 \Rightarrow p_2) \wedge \neg p_2$$
$$f(4) = p_1 \wedge \top \wedge (p_1 \Rightarrow p_2) \wedge (p_2 \Rightarrow p_3) \wedge \neg p_3$$

etc.

Les restrictions sur le schéma d'entrée sont assez importantes. Il semble qu'il soit nécessaire de restreindre les indices entiers à 1 ce qui fixe les indices des autres propositions indexées, et empêche, p.ex. des translations sur ces indices. Cependant il serait très facile de modifier le formalisme des LIF pour autoriser d'autres entiers, sans perdre les propriétés des LIF (il suffirait d'autoriser que le cas $n = 0$ d'une LIF puisse accéder à d'autres rangs que le premier, à condition qu'ils soient entiers). Dans ce cas, il serait possible de ramener tout schéma non aligné sur $[2; n-1]$ en un schéma équivalent aligné sur $[2; n-1]$, par translation des indices. La restriction que les indices contenant une variable liée ait la forme i ou $i-1$ pourrait aussi être aisément relaxée en considérant n'importe quelles expressions dont la différence est 1, p.ex. i ou $i+1$, $i+1$ ou $i+2$, etc. Autoriser n'importe quelle différence est plus difficile : il faut alors utiliser des propositions indexées intermédiaires encodant le décalage (de façon informelle, nous pourrions définir $q_i \Leftrightarrow p_{i-1}$, $r_i \Leftrightarrow q_{i-1}$, etc.). Au final, nous pouvons considérer que le pouvoir d'expression des LIF est sensiblement le même que celui des schémas réguliers.

Les différences entre schémas réguliers et LIF ne résident donc pas essentiellement dans leur pouvoir d'expression mais plutôt dans l'approche de leur étude : le but des LIF est véritablement de modéliser des schémas de *circuits*. Cette orientation se ressent dans les problématiques et les solutions apportées par Gupta : le raisonnement sur les LIF est fait en étendant les BDD (en définissant ce que nous pouvons qualifier, dans le contexte de ce mémoire, des "schémas de BDD"), la problématique de la représentation graphique d'une LIF est aussi abordée. Ainsi les procédures ne sont pas généralisées à d'autres schémas (simplement car dans ce contexte d'étude, il n'y a pas forcément d'intérêt à considérer d'autres schémas). A contrario, nos procédures, en plus de proposer des approches différentes[3], sont plus générales d'un point de vue logique. En particulier elles sont d'avantage susceptibles d'être étendues à d'éventuels schémas de formules du premier ordre.

10.5.2 Fonctions exponentiellement inductives

Nous définissons maintenant l'autre formalisme introduit dans [Gup94] : les *EIF*. Comme nous allons le voir, ce formalisme n'est pas capturé par les schémas réguliers, et ne semble pas l'être de façon simple par les schémas en général. Au lieu de prendre en paramètre un nombre seul, ces fonctions prennent aussi un vecteur de propositions indexées qu'elle peuvent utiliser.

Definition 10.2. *Soit $P = (P_i)_{i\in[1;2^{i+1}]}$ une suite de propositions indexées. Nous notons P^i la sous-suite $\{P_1, \ldots, P_{2^i}\}$, P^i_L la sous-suite $\{P_1, \ldots, P_{2^{i-1}}\}$ et P^i_R la sous-suite $\{P_{2^{i-1}+1}, \ldots, P_{2^i}\}$.*

Alors une fonction exponentiellement inductive *(ou* EIF*) f prend un paramètre n et une suite P de longueur 2^n de propositions indexées et $f(n, P)$ est définie de la façon suivante :*

$$\begin{cases} \phi_1 \in \mathcal{P}rop(P^0) & \text{si } n = 1 \\ \phi_n \in \mathcal{P}rop(\{g_1(n-1, P_L^{n-1}), g_2(n-1, P_R^{n-1}), g_3(n-1, P_R^{n-1})\}) & \text{sinon} \end{cases}$$

où g_1, g_2 et g_3 sont des EIF (possiblement f).

L'intérêt des EIF est qu'elles peuvent représenter des circuits ayant une structure arborescente plutôt qu'une structure linéaire : à chaque appel récursif, une fonction s'occupe de la partie gauche des entrées et l'autre (en fait les deux autres) de la partie droite. Ceci permet de représenter des circuits de type "diviser pour régner". Nous reportons le lecteur à [Gup94] pour des exemples de tels circuits.

Comme nous l'avons vu (sans le prouver) dans l'exemple 6.8 ce type de circuit n'est pas représentable par un schéma régulier. En revanche ce même exemple donne des pistes sur la façon dont il est possible d'encoder cette structure avec un schéma général. Cependant cet encodage ne serait pas très naturel et surtout Stab et Dpll* ne termineraient probablement pas : il faudrait définir des raffinements de bouclage dédiés. Le moyen le plus simple serait en fait d'étendre le formalisme des schémas de sorte à considérer des indices non entiers. Nous pourrions p.ex. représenter les EIF en considérant deux symboles de fonctions unaires l et r et une constante a : nous pourrions avoir les propositions indexées $p_{l(a)}$, $p_{r(a)}$, ou encore $p_{l(l(r(l(a))))}$, ce qui permet d'organiser de façon arborescente les variables. Il semble, a priori, assez simple d'étendre Stab de façon à gérer ce type d'objet. Cette extension fait partie des travaux futurs de cette thèse.

[3]Bien que des travaux aient mis en évidence les rapports entre les BDD et les tableaux

CHAPTER 11

Conclusions and Perspectives

11.1 Résumé des résultats

Nous avons proposé une formalisation (syntaxe et sémantique) de la notion de *schéma itéré de formule* en logique propositionnelle. Cette formalisation permet de capturer, entre autres, les exemples informels présentés en introduction comme étant des schémas itérés de formule (problème des pigeonniers, k-coloriabilité d'un graphe, circuit additionneur).

Nous avons prouvé que la logique associée n'est pas complète (pour la réfutation). Par conséquent le problème de la satisfaisabilité est *indécidable*. En revanche il existe une procédure naïve qui permet de trouver un modèle lorsqu'une conjecture est satisfaisable (propriété appelée "complétude pour la satisfaisabilité"). Par conséquent ce problème est seulement *semi-décidable*. Cependant peu de conjectures insatisfaisables peuvent être réfutées par la procédure naïve. Ainsi, de nouvelles procédures sont introduites ensuite, conçues dans le but de réfuter plus de conjectures insatisfaisables que cette procédure naïve, évidemment *tout en préservant la propriété de complétude pour la satisfaisabilité*.

Nous avons introduit un premier système de preuve, très simple, intitulé Stab. Comme son nom le suggère, Stab est une procédure étendant les tableaux propositionnels aux schémas. Elle est strictement plus générale que la procédure naïve évoquée ci-dessus. De plus Stab est également plus efficace que la méthode naïve même dans des cas où la conjecture est satisfaisable.

Le raisonnement sur les schémas requiert un raisonnement *inductif* : la réfutation d'un schéma insatisfaisable se fait généralement par récurrence sur la valeur du ou des paramètre(s). La détection de cycle est un moyen simple d'étendre une procédure sans raisonnement inductif en une procédure permettant un tel raisonnement. Nous avons formellement défini cette notion dans le contexte de procédures construisant des tableaux. C'est surtout dans ce contexte que Stab montre sa supériorité par rapport à la procédure naïve : les tableaux générés par Stab sont particulièrement adaptés à la détection de cycles, car les règles de décomposition des itérations visent précisément à relier l'instance n du schéma à l'instance $n-1$ (ce que ne fait pas la méthode naïve).

Muni de cette détection de cycle, Stab nous permet d'identifier une classe de schémas pour laquelle le problème de la satisfaisabilité est décidable : les *schémas réguliers*. Cette classe est donc complète pour la réfutation. Nous avons, de plus, présenté une procédure *dédiée aux schémas réguliers*, appelée SchAut. Dû au fait que SchAut ne prend en entrée que des schémas réguliers, cette procédure est beaucoup plus simple et définit un cadre plus adéquat pour étudier la complexité de cet algorithme. Nous avons ainsi prouvé que SchAut résout le problème de la satisfaisabilité en un temps exponentiel par rapport au codage des entiers dans le schéma d'entrée (c.-à-d. exponentiel si les entiers sont codés en unaire, double exponentielle s'ils sont codés en binaire). Ce résultat montre que les schémas réguliers restent relativement utilisables en pratique (de plus ce résultat peut être relativisé par le gain en expressivité et en concision par rapport aux formules propositionnelles). De plus, une stratégie d'application de Stab dédiée aux schémas réguliers a été implémentée dans un prototype appelée RegStab. Grâce à un

important travail d'optimisation et de raffinement de la stratégie de preuve et des structures de données utilisées, ce programme a permis de prouver de manière raisonnablement efficace un certain nombre de benchmarks que les démonstrateurs par induction existants ne peuvent prendre en charge. Notons que ce sont ces raffinements qui ont donné lieu à la conception de SchAut (en fait RegStab est plus proche de SchAut que de Stab).

11.2 Sur le travail réalisé

Nous passons en revue des points développés dans cette thèse qui nous semblent perfectibles. À notre décharge, nous pensons que les problèmes mentionnés n'étaient visibles qu'avec le recul, et, en particulier, seulement après avoir exploré plusieurs possibilités comme nous l'avons fait dans cette thèse, ce qui n'avait jamais été fait auparavant.

Formalisme lui-même

Tout d'abord le formalisme présenté ici nous semble plutôt complexe. Le formalisme original (tel que présenté dans [ACP09b]) était beaucoup plus simple (il correspondait en fait à ce que nous appelons ici les schémas linéairement et syntaxiquement encadrés, c.-à-d. toute itération est de la forme $\Pi_{i=e}^{f} m$, voir proposition 3.52). Sa forme actuelle provient de modifications appliquées afin de rendre les procédures Stab et Dpll* plus faciles à exprimer, de réduire le nombre de règles et de les rendre plus uniformes. Cependant le formalisme résultant peut sembler moins naturel car l'introduction de contraintes complique la présentation (en revanche certains types de schémas sont exprimés de façon intuitive dans le formalisme actuel, p.ex. une itération de la forme $\Pi_{\substack{i=1\\ i\neq j}}^{n} m$ s'exprime très simplement par $\Pi_{i|i\in[1;n]\wedge i\neq j}\, m$). Surtout, nous sommes finalement obligés de remettre les schémas en forme normale durant la preuve afin de pouvoir appliquer les règles (cependant, en pratique, nous pouvons nous passer de la plupart de ces transformations). Ainsi l'avantage de travailler sans forme normale est un peu perdu.

Si encore le nouveau formalisme était beaucoup plus expressif, cela compenserait ce manque de simplicité, mais nous pouvons facilement faire un formalisme beaucoup plus expressif, par exemple en considérant n'importe quel fonction récursive dont le codomaine serait l'ensemble des formules de la logique propositionnelle. Évidemment, nous n'aurions alors que peu d'espoir d'automatisation et devrions considérer des sous-classes, mais c'est précisément ce que nous faisons déjà avec le formalisme actuel. Pour résumer il nous semble que le compromis "expressivité/simplicité" du formalisme actuel n'est pas optimal. Pour ce qui est du bénéfice consistant à simplifier la définition des procédures, il nous semble qu'il serait en fait plus judicieux de conserver un formalisme simple quitte à avoir des procédures légèrement plus complexes.

Dpll*

De même concernant Dpll* : cette procédure est beaucoup trop complexe, surtout avec l'utilisation de la procédure subordonnée $\mathrm{R_{syn}}$. La preuve de complétude pour la satisfaisabilité (théorème 8.13) est particulièrement difficile sans que les concepts sous-jacents soient spécialement intéressants. De même la preuve de terminaison pour les schémas réguliers imbriqués (théorème 9.33) consiste, pour la majeure partie, en la définition d'une stratégie sur Dpll* et en l'étude des propriétés de cette stratégie. Une fois encore cette étude est rendue fastidieuse par la complexité du système et du non déterminisme dans l'application des règles. À l'avenir, il nous semble plus judicieux de se focaliser sur Stab et de travailler au développement de divers raffinements de bouclage qui permettraient, peut-être, d'arriver aux mêmes résultats que Dpll* mais de façon beaucoup plus modulaire. Évidemment la complexité serait transférée dans la définition des raffinements de bouclage, mais le fait que ceux-ci puissent être clairement séparés selon ce pour quoi ils ont été conçus (comme nous l'avons vu pour les littéraux purs et les contraintes équivalentes) simplifierait probablement le raisonnement.

Schémas réguliers imbriqués

Comme nous l'avons dit, la preuve de terminaison des schémas réguliers est compliquée : la méthode de traduction évoquée en section 9.A est beaucoup plus simple. Cependant, comme précisé dans la même section, cette preuve nous semble facile à étendre à d'autres classes (p.ex. en relaxant la contrainte

d'alignement, ou en autorisant des variables liées dans les bornes des itérations). Nous mentionnons ceci uniquement à titre indicatif, car les classes pour lesquelles nous pourrions alors démontrer la terminaison ne nous semblent pas suffisamment pertinentes et appuyées par des utilisations pratiques pour justifier un tel effort. Ainsi, bien que le cœur de la preuve nous semble intéressant, il ne nous semble pas pertinent de persévérer dans cette voie.

11.3 Perspectives

Schémas polyadiques

Comme nous l'avons vu tout au long de cette thèse, les seules classes décidables identifiées imposent que les schémas soient monadiques. Il existe différents moyens de considérer des extensions de ces classes mais il nous semble que la plus pertinente consiste à considérer des schémas non monadiques. Ceci permettrait par exemple de formaliser le principe des pigeonniers, la k-coloriabilité, les n reines ou encore le jeu de la vie. Un exemple plus concret serait la modélisation d'un circuit multiplicateur [KS96]. Notons que cette piste semble difficile à concrétiser : il est très facile de prouver l'indécidabilité du problème de la satisfaisabilité dès lors que nous autorisons plusieurs indices. Par conséquent les classes décidables que nous pourrons identifier seront probablement très restreintes par ailleurs.

Schémas itérés en logique du premier ordre

Une évolution naturelle de ce travail serait l'extension des schémas à la logique du premier ordre. Comme nous l'avons mentionné en introduction (section 1.3), ceci irait dans le sens de l'une des motivations originales de cette thèse : l'utilisation de schémas pour la formalisation des mathématiques, comme le fait p.ex. le groupe d'Alexander Leitsch à Vienne. Cette piste nécessiterait la combinaison de schémas de formules et de schémas de termes. De plus il faudrait probablement considérer aussi des "quantificateurs itérés", p.ex. pour pouvoir exprimer le schéma mentionné en introduction :

$$\forall n \geq 1, \forall x_1, \ldots, x_n \in \mathbb{R} : \left(\sum_{i=1}^{n} x_i^2\right) \leq \left(\sum_{i=1}^{n} x_i\right)^2$$

Note. Les sommes peuvent être exprimées par des schémas de termes. En revanche elles font appel à une infinité de variables indicées, ce qui est connu pour poser des problèmes, notamment au niveau de l'unification.

Évidemment, les schémas en logique du premier ordre ont un intérêt qui va potentiellement au delà de la formalisation des mathématiques. En fait tous les avantages vus en introduction sur les schémas en logique propositionnelle restent valables au premier ordre : gain en lisibilité, en structure, intérêts potentiels du compromis entre pouvoir d'expression et possibilité d'automatisation. Tous ces avantages potentiels feraient des schémas en logique du premier ordre de bons candidats pour être utilisés, p.ex., en vérification de logiciels.

Procédure basée sur la résolution

Toujours dans l'idée de développer les schémas afin de les utiliser dans le projet ASAP, en partenariat avec l'équipe d'Alexander Leitsch, il est nécessaire de développer un calcul basé sur la *résolution* : en effet la méthode CERES repose de façon essentielle sur les propriétés de la résolution, en particulier sur le fait qu'une réfutation par résolution peut être vue comme une preuve en calcul des séquents constituée uniquement de coupures atomiques. Une telle procédure ne semble pas si évidente à définir car la résolution ne construit pas un arbre de la même façon que le font les tableaux sémantiques ou DPLL : la notion de bouclage semble nécessiter des adaptations conséquentes.

Étude des connexions précises avec les logiques d'ordre supérieur

Comme évoqué au chapitre 6, de nombreuses connexions semblent être envisageables avec la logique monadique du second ordre. Le fait notamment que cette dernière soit utilisée pour formaliser des propriétés sur des circuits (voir section 1.3 et [BK95]) laisse envisager des liens intéressants, de même que le fait que nous puissions exprimer le comportement d'automates finis avec les schémas réguliers : en effet

des résultats classiques importants mettent en lumière les relations profondes entre logique monadique du second ordre (faible) et langages réguliers [Bü59]. Des résultats similaires sont envisageables sur les schémas, ce qui aurait pour conséquence intéressante de pouvoir comparer deux différentes formes d'abstraction : l'ordre supérieur et les algorithmes. Il est dit (rapidement) dans [BK95] que l'ordre supérieur permet une forme d'abstraction plus déclarative que les algorithmes. Nous pensons que c'est effectivement le cas mais que cela ne signifie pas pour autant que cette forme d'abstraction soit toujours à privilégier.

Assistants de preuve

Même pour une classe très simple comme celle des schémas réguliers la complexité du problème de la satisfaisabilité est relativement importante. De plus, il semble difficile d'obtenir des résultats de décidabilité pour des schémas ayant des propriétés intéressantes comme la polyadicité. Il semble donc difficilement imaginable d'obtenir des classes décidables expressives pour les schémas au premier ordre, et il y a fort à parier que, même si c'est le cas, le problème de la satisfaisabilité pour de telles classes aura une complexité très grande. Ainsi l'intégration des schémas à des assistants de preuves interactifs semblent être une voie bien plus réaliste pour un traitement formel du raisonnement avec les schémas. De plus, un tel travail pourrait aussi être bénéfique aux assistants de preuve en permettant d'améliorer l'automatisation dans le cas où la formule à prouver est un schéma (notons que, dans ce contexte, même une automatisation *partielle* peut être bénéfique).

Index

Bibliography

[ACP08] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : More Flexible Term Schematisations via Extended Primal Grammars. *In* Berthe Y. Choueiry et Bob Givan, éditeurs : *Electronic Proceedings of the 10th International Symposium of Artificial Intelligence and Mathematics*, 2008.

[ACP09a] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : A Dpll Proof Procedure for Propositional Iterated Schemata. *In* Michel Parigot et Lutz Strassburger, éditeurs : *Workshop Structures and Deduction*, pages 24–38, 2009.

[ACP09b] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : A Schemata Calculus for Propositional Logic. *In* Giese et Waaler [GW09], pages 32–46.

[ACP10a] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : A Decidable Class of Nested Iterated Schemata. *In* Giesl et Hähnle [GH10], pages 293–308.

[ACP10b] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : A Decidable Class of Nested Iterated Schemata (extended version). Rapport technique, Laboratoire d'Informatique de Grenoble, 2010. `http://arxiv.org/abs/1001.4251`.

[ACP10c] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : Complexity of the Satisfiability Problem for a Class of Propositional Schemata. *In* Adrian-Horia Dediu, Henning Fernau et Carlos Martín-Vide, éditeurs : *Language and Automata Theory and Applications*, volume 6031, pages 58–69. Springer, Heidelberg, 2010.

[ACP10d] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : Decidability and Undecidability Results for Propositional Schemata. 2010. Soumis, `http://membres-liglab.imag.fr/aravantinos/Site/Publications_files/jair.pdf`.

[ACP10e] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : RegSTAB: A SAT-Solver for Propositional Iterated Schemata. *In* Giesl et Hähnle [GH10], pages 309–315.

[ACP10f] Vincent Aravantinos, Ricardo Caferra et Nicolas Peltier : Simplified Handling of Iterated Term Schemata. *Annals of Mathematics and Artificial Intelligence*, 2010. À paraître.

[AKW09] Ernst Althaus, Evgeny Kruglov et Christoph Weidenbach : Superposition modulo linear arithmetic SUP(LA). *In* Silvio Ghilardi et Roberto Sebastiani, éditeurs : *Frontiers of Combining Systems : 7th International Symposium, FroCoS 2009*, volume 5749, pages 84–99. Springer, September 2009.

[Ara07] Vincent Aravantinos : *Schémas de preuves et de formules : vers plus de structure et d'efficacité en déduction automatique*. Master thesis, Institut National Polytechnique de Grenoble & Université Joseph Fourier, jun 2007. Disponible à: `http://membres-liglab.imag.fr/aravantinos`.

[Ara10] Vincent Aravantinos : RegsTAB's user manual. Disponible à `http://regstab.forge.ocamlcore.org/`, 2010.

[Bae08] David Baelde : *A Linear Approach to the Proof-Theory of Least and Greatest Fixed Points*. Thèse de doctorat, École Polytechnique, 2008.

[Bae09] David Baelde : On the Proof Theory of Regular Fixed Points. *In* Giese et Waaler [GW09], pages 93–107.

[BCP07] Hicham Bensaid, Ricardo Caferra et Nicolas Peltier : Towards systematic analysis of theorem provers search spaces: First steps. *In* Daniel Leivant et Ruy J. G. B. de Queiroz, éditeurs : *Proceedings Wollic'07 (Workshop on Logic, Language, Information and Computation)*, volume 4576, pages 38–52. Springer, July 2007.

[BCP09] Hicham Bensaid, Ricardo Caferra et Nicolas Peltier : Dei: A Theorem Prover for Terms with Integer Exponents. *In* Renate A. Schmidt, éditeur : *CADE*, volume 5663 de *Lecture Notes in Computer Science*, pages 146–150. Springer, 2009.

[BCP10a] Hicham Bensaid, Ricardo Caferra et Nicolas Peltier : I-terms in ordered resolution and superposition calculi : retrieving lost completeness. *In* Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo et Alan P. Sexton, éditeurs : *AISC 2010 (10th International Conference on Artificial Intelligence and Symbolic Computation)*, volume 6167, pages 19–33. Springer, 2010.

[BCP10b] Hicham Bensaid, Ricardo Caferra et Nicolas Peltier : Perfect discrimination graphs: indexing terms with integer exponents. *In* Giesl et Hähnle [GH10], pages 369–383.

[BdlT92] Thierry Boy de la Tour : An optimality result for clause form translation. *Journal of Symbolic Computation*, 14:283–301, 1992.

[BdlT07] Thierry Boy de la Tour : Logique classique. Polycopié de cours de DEA, 2007.

[BGW94] Leo Bachmair, Harald Ganzinger et Uwe Waldmann : Refutational theorem proving for hierachic first-order theories. *Applicable Algebra in Engineering, Communication and Computing*, 5:193–212, 1994.

[BHL+08] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter et Hendrik Spohr : Ceres: An analysis of Fürstenberg's proof of the infinity of primes. *Theoretical Computer Science*, 403(2-3):160–175, 2008.

[BK95] David A. Basin et Nils Klarlund : Hardware verification using monadic second-order logic. *In* Pierre Wolper, éditeur : *Proceedings of the 7th International Conference on Computer Aided Verification*, volume 939, pages 31–41. Springer-Verlag, 1995.

[BKR92] Adel Bouhoula, Emmanuel Kounalis et Michaël Rusinowitch : Automated mathematical induction. Research Report RR-1663, INRIA, 1992. Projet EURECA.

[BLdM09] Maria Paola Bonacina, Christopher A. Lynch et Leonardo de Moura : On deciding satisfiability by DPLL($\Gamma + \mathcal{T}$) and unsound theorem proving. *In* Renate Schmidt, éditeur : *Proceedings of the Twenty-second Conference on Automated Deduction (CADE)*, volume 5663 de *Lecture Notes in Artificial Intelligence*, pages 35–50. Springer, 2009.

[BM79] Robert Stephen Boyer et J. Strother Moore : *A Computational Logic*. Academic Press, New York, 1979.

[BMS10] David Baelde, Dale Miller et Zachary Snow : Focused Inductive Theorem Proving. *In* Giesl et Hähnle [GH10], pages 278–292.

[Bon05] Maria Paola Bonacina : Towards a unified model of search in theorem proving: subgoal-reduction strategies. *Journal of Symbolic Computation*, 39(2):209–255, February 2005.

[BP97] Samuel R. Buss et Toniann Pitassi : Resolution and the weak pigeonhole principle. *In* Mogens Nielsen et Wolfgang Thomas, éditeurs : *11th International Workshop, CSL '97, Annual Conference of the EACSL, Aarhus, Denmark, August 23-29, 1997. Proceedings*, volume 1414, pages 149–156. Springer, 1997.

[Bro05] James Brotherston : Cyclic Proofs for First-Order Logic with Inductive Definitions. *In* Bernhard Beckert, éditeur : *Automated Reasoning with Analytic Tableaux and Related Methods: Proceedings of TABLEAUX 2005*, volume 3702, pages 78–92. Springer-Verlag, 2005.

[Bro06] James Brotherston : *Sequent Calculus Proof Systems for Inductive Definitions*. Thèse de doctorat, University of Edinburgh, November 2006.

[BS07] Julian Bradfield et Colin Stirling : Modal Mu-Calculi. *In* Patrick Blackburn, Johan van Benthem et Frank Wolter, éditeurs : *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., 2007.

[BSvH+93] Alan Bundy, Andrew Stevens, Frank van Harmelen, Andrew Ireland et Alan Smaill : Rippling: a heuristic for guiding inductive proofs. *Artificial Intelligence*, 62(2):185–253, 1993.

[Bun01] Alan Bundy : The Automation of Proof by Mathematical Induction. *In* Robinson et Voronkov [RV01], pages 845–911.

[Bus87] Samuel R. Buss : Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52:916–927, 1987.

[BZ94] Matthias Baaz et Richard Zach : Short Proofs of Tautologies Using the Schema of Equivalence. *In* Egon Börger, Yuri Gurevich et Karl Meinke, éditeurs : *Computer Science Logic (CSL '93)*, pages 33–35. Springer-Verlag, 1994.

[Bü59] J. Richard Büchi : Weak second-order arithmetic and finite automata. Rapport technique, The University of Michigan Research Institute, 1959.

[Caf10] Ricardo Caferra : Une approche simple de la logique pour l'informatique et la formalisation de l'inférence. Polycopié de cours Ensimag, Grenoble I.N.P., 2010.

[CH82] Ashok Chandra et David Harel : Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.

[CHK90] Hong Chen, Jieh Hsiang et Hwa-Chung Kong : On Finite Representations of Infinite Sequences of Terms. *In* Stéphane Kaplan et Mitsuhiro Okada, éditeurs : *Conditional and Typed Rewriting Systems*, volume 516, pages 100–114. Springer, 1990.

[Cle90] Rance Cleaveland : Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27:725–747, 1990.

[Com95] Hubert Comon : On Unification of Terms with Integer Exponents. *Mathematical System Theory*, 28:67–88, 1995.

[Com01] Hubert Comon : Inductionless Induction. *In* Robinson et Voronkov [RV01], chapitre 14, pages 913–962.

[Coo72] D. C. Cooper : Theorem Proving in Arithmetic Without Multiplication. *Machine Intelligence 7*, pages 91–99, 1972.

[Cor06] John Corcoran : Schemata: the concept of schema in the history of logic. *The Bulletin of Symbolic Logic*, 12:219–240, 2006.

[Cor08] John Corcoran : Schema. *In* Edward N. Zalta, éditeur : *The Stanford Encyclopedia of Philosophy*. Fall 2008 édition, 2008.

[DLL62] Martin Davis, George Logemann et Donald Loveland : A Machine Program for Theorem Proving. *Communication of the ACM*, 5:394–397, 1962.

[DLL+10] Cvetan Dunchev, Alexander Leitsch, Tomer Libal, Daniel Weller et Bruno Woltzenlogel Paleo : The Proof Transformation System CERES. *In* Giesl et Hähnle [GH10], pages 427–433.

[dt04] The Coq development team : *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[End72] Herbert B. Enderton : *A Mathematical Introduction to Logic*. Academic Press, 1972.

[Eps90] Richard L. Epstein : *The semantic foundations of Logic. Volume 1: Propositional Logics*, volume 35 de *Nijhoff International Philosophy Series*. Kluwer Academic Publishers, 1990.

[Fag93] Ronald Fagin : Finite-Model Theory - A Personal Perspective. *Theoretical Computer Science*, 116:3–31, 1993.

[Fef96] Solomon Feferman : Gödel's program for new axioms: Why, where, how and what? *In IN GÖDEL '96*, pages 3–22. Springer, 1996.

[Fit90] Melvin Fitting : *First-Order Logic and Automated Theorem Proving*. Texts and Monographs in Computer Science. Springer-Verlag, 1990.

[FK06] Stephan Falke et Deepak Kapur : Inductive Decidability Using Implicit Induction. *In* Miki Hermann et Andrei Voronkov, éditeurs : *Proceedings of the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '06)*, Phnom Penh, Cambodia, volume 4246, pages 45–59. Springer-Verlag, 2006.

[FR74] M.J. Fisher et M.O. Rabin : Super Exponential Complexity of Presburger's Arithmetic. *SIAM-AMS Proceedings*, 7:27–41, 1974.

[Fre60] Edward Fredkin : Trie memory. *Communications of the ACM*, 3:490–499, 1960.

[FS00] Solomon Feferman et Thomas Strahm : The unfolding of non-finitist arithmetic. *Annals of Pure and Applied Logic*, 104:75 – 96, 2000.

[Gen35] Gerhard Gentzen : Untersuchungen über das logische schließen ii. *Mathematische Zeitschrift*, 39, 1935.

[GH10] Jürgen Giesl et Reiner Hähnle, éditeurs. *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, Scotland, July 16-19, 2010, Proceedings*, volume 6173. Springer, 2010.

[Gir06] Jean-Yves Girard : *Le Point Aveugle: Tome 1. Cours de Logique, Vers la perfection.* Hermann, Juillet 2006.

[GK01] Jürgen Giesl et Deepak Kapur : Decidable Classes of Inductive Theorems. *In* Rajeev Goré, Alexander Leitsch et Tobias Nipkow, éditeurs : *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, volume 2083, pages 469–484. Springer-Verlag, 2001.

[GLT89] Jean-Yves Girard, Yves Lafont et Paul Taylor : *Proofs and Types (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, April 1989.

[GM93] Michael J. C. Gordon et Tom F. Melham, éditeurs. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[Gor99] Rajeev Goré : Chapter 6: Tableau Methods for Modal and Temporal Logics. *In* Marcello D'Agostino and Dov M. Gabbay and Reiner Hähnle and Joachim Posegga, éditeur : *Handbook of Tableau Methods*, pages 297–396. Kluwer Academic Publishers, 1999.

[Gup94] Aarti Gupta : *Inductive Boolean Function Manipulation*. Thèse de doctorat, Carnegie Mellon University, 1994. A Hardware Verification Methodology for Automatic Induction.

[GW09] Martin Giese et Arild Waaler, éditeurs. *Automated Reasoning with Analytic Tableaux and Related Methods, 18th International Conference, TABLEAUX 2009, Oslo, Norway, July 6-10, 2009. Proceedings*, volume 5607. Springer, 2009.

[Her94] Miki Hermann : *Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes*. Habilitation, Université Henri Poincaré Nancy 1, mar 1994.

[HG97] Miki Hermann et Roman Galbavý : Unification of Infinite Sets of Terms Schematized by Primal Grammars. *Theoretical Computer Science*, 176:111–158, 1997.

[Hod83] Wilfrid Hodges : Elementary Predicate Logic. *In* Dov M. Gabbay et Franz Guenthner, éditeurs : *Handbook of Philosophical Logic*, chapitre I.1, pages 2–131. D. Reidel, 1983. Vol. 1, Elements of Classical Logic.

[HW09] Matthias Horbach et Christoph Weidenbach : Deciding the inductive validity of $\forall\exists*$ queries. *In* Erich Grädel et Reinhard Kahle, éditeurs : *CSL'09/EACSL'09: Proceedings of the 23rd CSL international conference and 18th EACSL Annual conference on Computer science logic*, volume 5771, pages 332–347. Springer-Verlag, 2009.

[Imm82] Neil Immerman : Relational Queries Computable in Polynomial time (Extended Abstract). *In STOC '82: Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 147–152. ACM, 1982.

[KB70] Donald E. 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.

[KK67] Jean-Louis Krivine et Georg Kreisel : *Éléments de Logique Mathématique (théorie des modèles)*. Dunod, 1967.

[Koz83] Dexter Kozen : Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[KP88] Jan Krajíèek et Pavel Pudlák : The number of proof lines and the size of proofs. in first order logic. *Archive for Mathematical Logic*, 27:69–84, 1988.

[KS96] Deepak Kapur et M. Subramaniam : Mechanically verifying a family of multiplier circuits. *In* Rajeev Alur et Thomas A. Henzinger, éditeurs : *CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification*, volume 1102, pages 135–146. Springer-Verlag, 1996.

[KS00] Deepak Kapur et Mahadevan Subramaniam : Extending Decision Procedures with Induction Schemes. *In* David A. McAllester, éditeur : *CADE-17: Proceedings of the 17th International Conference on Automated Deduction*, volume 1831, pages 324–345. Springer-Verlag, 2000.

[KV07] Konstantin Korovin et Andrei Voronkov : Integrating linear arithmetic into superposition calculus. *In* Jacques Duparc et Thomas A. Henzinger, éditeurs : *Proceedings of the Sixteenth EACSL Annual Conference on Computer Science Logic (CSL)*, volume 4646 de *Lecture Notes in Computer Science*, pages 223–237. Springer, 2007.

[LC08] Stéphane Lescuyer et Sylvain Conchon : A reflexive formalization of a sat solver in coq. *In* Otmane Ait Mohamed, César Muñoz et Sofiène Tahar, éditeurs : *Electronic proceedings of Emerging Trends of the 21st International Conference on Theorem Proving in Higher Order Logics*, volume 5170 de *Lecture Notes in Computer Science*. Springer, 2008.

[Ore91] Vladimir P. Orevkov : Proof schemata in Hilbert-type axiomatic theories. *Journal of Mathematical Sciences*, 55:1610–1620, 1991.

[ORS92] Sam Owre, John M. Rushby, et Natarajan Shankar : PVS: A prototype verification system. *In* Deepak Kapur, éditeur : *11th International Conference on Automated Deduction (CADE)*, volume 607, pages 748–752. Springer-Verlag, 1992.

[PG86] David Plaisted et Steven Greenbaum : A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[Pos46] Emil Leon Post : A Variant of a Recursively Unsolvable Problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[PS99] Frank Pfenning et Carsten Schürmann : System description: Twelf - a meta-logical framework for deductive systems. *In* Harald Ganzinger, éditeur : *CADE-16: Proceedings of the 16th International Conference on Automated Deduction*, volume 1632, pages 202–206. Springer-Verlag, 1999.

[Red90] Uday S. Reddy : Term rewriting induction. *In* Mark E. Stickel, éditeur : *CADE-10: Proceedings of the tenth international conference on Automated deduction*, volume 449, pages 162–177. Springer-Verlag, 1990.

[Rob65] J. A. Robinson : A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[RV01] John Alan Robinson et Andrei Voronkov, éditeurs. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[Sal92] Gernot Salzer : The Unification of Infinite Sets of Terms and its Applications. *In* Andrei Voronkov, éditeur : *Logic Programming and Automated Reasoning (LPAR'92)*, volume 624, pages 409–429. Springer, July 1992.

[SD03] Cristoph Sprenger et Mads Dam : On the Structure of Inductive Reasoning: Circular and Tree-shaped Proofs in the mu-Calculus. *In* Andrew D. Gordon, éditeur : *Proceedings of FOSSACS'03*, volume 2620, pages 425–440. Springer-Verlag, 2003.

[Sha91] Stewart Shapiro : *Foundations without Foundationalism: A Case for Second-order Logic*. Oxford: Oxford University Press, 1991.

[Smu68] Raymond M. Smullyan : *First-Order Logic*. Springer-Verlag, 1968.

[Wika] Wikipedia : Méthode de descente infinie. `http://fr.wikipedia.org/wiki/M%C3%A9thode_de_descente_infinie`.

[Wikb] Wikipedia : True quantified boolean formula. `http://en.wikipedia.org/wiki/True_quantified_Boolean_formula`.

[Wir99] Claus-Peter Wirth : Full first-order free variable sequents and tableaux in implicit induction. *In* Neil V. Murray, éditeur : *TABLEAUX '99: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1617, pages 293–307. Springer-Verlag, 1999.

[Zha88] Hantao Zhang : *Reduction, superposition and induction: automated reasoning in an equational logic*. Thèse de doctorat, Rensselaer Polytechnic Institute, 1988.