Outils pour la preuve par analogie

Régis Curien

Université Henri Poincaré – Nancy I
Faculté des Sciences

CRIN-CNRS
INRIA-Lorraine

[UFR STMIA]
École Doctorale IAE + M
Département de Formation Doctorale en Informatique

OUTILS POUR LA PREUVE PAR ANALOGIE

THÈSE

présentée et soutenue publiquement le 25 janvier 1995

pour l'obtention du

Doctorat de l'Université Henri Poincaré – Nancy I

(Spécialité Informatique)

par

RÉGIS CURIEN

Composition du jury

Président :	H. COMON
Rapporteurs :	R. CAFERRA
	H. COMON
	C. GODART
Examinateurs :	G. DOWEK
	C. KIRCHNER
	D. LUGIEZ
Invité :	Z. QIAN

Je tiens à remercier Hubert Comon, qui m'a fait l'honneur de présider le jury de cette thèse.

Je remercie chaleureusement ceux qui accepté d'écrire un rapport sur ce travail :

Ricardo Caferra, qui a apporté son point de vue de spécialiste du domaine sur ce document. Ses travaux ont, en partie, guidé ma recherche.

Gilles Dowek, pour son travail méticuleux, ses critiques très constructives, sa patience et l'intérêt qu'il a manifesté pour ce travail. Je souhaite à tous les thésards d'avoir un tel rapporteur.

Claude Godart qui a surmonté l'éloignement du sujet de cette thèse par rapport à ses préoccupations habituelles pour y apporter sa contribution.

D'autre part, je voudrais remercier Claude Kirchner qui, en participant à ce jury, a montré son intérêt pour ces travaux, et en tant que responsable du projet PROTHEO, m'a permis de travailler dans des conditions toujours excellentes.

Zhenyu Qian, quant à lui, m'a fait le plaisir de venir spécialement de Brême pour participer à ce jury. Depuis le moment où il s'est intéressé à mes travaux et que nous avons décidé de travailler ensemble sur le problème du filtrage, il a su me faire profiter de ses compétences et de sa clairvoyance avec patience et gentillesse. Cette collaboration a été extrêmement enrichissante, et je lui en suis très reconnaissant.

J'ai effectué cette thèse sous la direction de Denis Lugiez. Il a su guider mon travail par ses qualités de chercheur, pour aborder un sujet encore très peu exploré sous son aspect formel. La rigueur, la disponibilité, et la culture qu'il a mises à mon service ont su me donner le goût de la recherche. Je l'en remercie vivement.

Enfin, je ne saurais terminer sans remercier tous ceux qui ont contribué à cette thèse, par des discussions techniques, mais aussi par une ambiance très amicale au sein et en dehors du laboratoire.

Table des matières

Introduction 7

I Analogie simple 13
- I.1 Idée de la méthode . . . 13
- I.2 Formules . . . 16
- I.3 Relation entre les formules . . . 18
  - I.3.1 La relation $\preceq$ . . . 18
- I.4 Preuves par expansion . . . 21
- I.5 Preuve par analogie via la relation $\preceq$ . . . 28
  - I.5.1 Preuves par expansion réduites . . . 29
  - I.5.2 Elagage des preuves (E) . . . 32
  - I.5.3 Transformation : preuve par expansion réduite $\rightarrow$ preuve par expansion (TR) . . . 33
  - I.5.4 Propriétés des transformations E et TR . . . 37

II Filtrage AC d'ordre deux 45
- II.1 Représentation des formules par des λ-termes simplement typés . . . 45
  - II.1.1 Types . . . 46
  - II.1.2 λ-termes simplement typés . . . 46
  - II.1.3 Représentation de formules . . . 48
- II.2 Filtrage d'ordre deux . . . 49
- II.3 Combinaison de l'ordre deux et d'une théorie . . . 53
  - II.3.1 Propriétés de l'algorithme naïf pour la théorie AC . . . 57
  - II.3.2 Discussion sur la théorie utilisée . . . 60
  - II.3.3 Problèmes d'efficacité de l'algorithme classique . . . 61
- II.4 Algorithme efficace pour le filtrage AC d'ordre deux . . . 63
  - II.4.1 Propriétés du nouvel algorithme . . . 69

III Analogie pour des formules similaires 73
- III.1 Correspondance . . . 74
- III.2 Propriétés AC des connecteurs . . . 77
- III.3 Propriétés des quantificateurs . . . 78
  - III.3.1 Propriétés à prendre en compte . . . 78
  - III.3.2 La forme prénexe . . . 79
  - III.3.3 Forme normale anti-prénexe . . . 79

III.3.4 Transformation des preuves pour des formules ayant la même forme normale anti-prénexe 81
III.4 Combinaison des critères de similitude 91
III.5 Exemple d'application 94
III.5.1 Un système de déduction naturelle pour le calcul propositionnel 94
III.5.2 Typage simple des λ-termes 96
III.5.3 Analogie système logique/système fonctionnel 97

IV Calcul de la différence et application aux preuves par expansion 101
IV.1 Différence entre deux formules 101
IV.1.1 Filtrage de différence 102
IV.2 Processus d'analogie utilisant la différence 105
IV.2.1 Le cadre et la méthode d'analogie 105
IV.2.2 Un exemple complet d'application de cette technique 107
IV.3 Discussion de la complétude pour une différence minime 110
IV.4 Extension à d'autres systèmes de preuve 113
IV.4.1 Analogie pour des preuves non analytiques 114

V Analogie dans un cadre général 117
V.1 Situation dans les travaux sur l'analogie 117
V.2 Principe de l'analogie générale 119
V.3 Analogie immédiate 120
V.3.1 Analogie immédiate pour la synthèse de programmes 121
V.4 Analogie par saut 125
V.4.1 Analogie par saut pour l'analyse 126
V.5 Analogies horizontales 129
V.5.1 Analogie horizontale structurelle 130
V.5.2 Analogie horizontale par la différence 130
V.6 Analogie incomplète 132
V.7 Synthèse et comparaison avec les travaux existants 139
V.8 Conclusion sur l'analogie générale 141

Conclusion 143

Liste des figures 145

Bibliographie 147

Index 151

Introduction

Tous les hommes sont mortels, Socrate est un homme,
donc Socrate est mortel.

Tous les enfants aiment les sucettes à l'anis, Annie est une enfant,
donc Annie aime les sucettes à l'anis.

L'intuition nous dicte qu'il y a une analogie entre ces deux phrases. Pourtant, quel rapport y a-t-il entre Socrate et les sucettes à l'anis? C'est bien sûr la structure de ces propositions qui est identique, puisqu'il s'agit de deux syllogismes. Ainsi, si l'on sait montrer la validité de la première proposition, nous saurons le faire pour la seconde, en procédant *par analogie*. Utiliser le concept d'analogie se fait presque inconsciemment pour un humain, et il est évident qu'il s'agit d'une composante essentielle de son raisonnement. D'autre part, la déduction automatique – dont l'utilité et les applications ne sont plus à montrer – est un problème très difficile, puisqu'indécidable dès l'ordre un. Il semble donc raisonnable, et nous ne sommes pas les premiers à y songer, de tenter d'appliquer à la déduction automatique des méthodes empruntées à l'analogie. L'objectif de cette thèse est de montrer qu'en formalisant une partie du concept d'analogie, il est possible d'exhiber des méthodes, souvent automatiques, qui transforment une preuve d'une formule en preuve d'une autre formule lorsque celles-ci sont jugées analogues par un critère formel. Les méthodes ainsi construites peuvent êtres combinées, et constituent une boîte à outils pour la preuve par analogie.

Il ne fait aucun doute que l'analogie est cruciale pour le raisonnement humain. Mais s'y intéresser en tant que mécanisme de raisonnement relève plus des sciences humaines que de l'informatique. Nous ne pouvons que nous inspirer – et c'est déjà une forme d'analogie – de tels mécanismes pour tenter de résoudre des problèmes relativement simples. Pour automatiser ces mécanismes, il faut prendre en compte deux contraintes : réduire le concept d'analogie à des critères simples, et produire un effort de formalisation. En effet, il s'agit d'un phénomène beaucoup trop complexe et éloigné des compétences d'un informaticien pour être étudié globalement. Ce n'est que par fragments que l'on peut prétendre intégrer l'analogie à la déduction automatique. D'autre part, l'analogie fait appel à un grand nombre de notions très floues. Les questions que l'on doit se poser sont par exemple : "Que sont deux problèmes similaires?", ou "Que signifie que deux théorèmes se ressemblent?". C'est pourquoi nous proposons une collection de critères pouvant définir la similitude ou la ressemblance de façon formelle et, pour chacun de ces critères, nous établissons une méthode pour résoudre le problème d'analogie. C'est-à-dire que si deux formules A et B sont similaires selon ces critères et que nous possédons une preuve de A, l'analogie consiste à transformer cette preuve en preuve de B.

Kling a été le premier, en 1971, à à utiliser l'analogie en déduction automatique [Kli71]. L'idée était d'introduire des heuristiques permettant de réutiliser des solutions de problèmes déjà résolus. Malgré l'intérêt de cette idée, ces travaux restèrent lettre morte durant dix ans. C'est Plaisted qui la relança avec la notion d'abstraction. Plaisted [Pla81] proposa alors des méthodes complètes pour la plupart. Mais ce thème de recherche ne s'est réellement développé que récemment, en particulier avec les travaux de Boy de la Tour, Caferra et Kreitz [BdlTC87, BdlTC88, BdlTK92] qui ont proposé des méthodes d'analogie faisant apparaître un effort de formalisation. Par exemple, l'utilisation du filtrage sur des termes d'ordre deux (c'est-à-dire dans lesquelles les variables de fonctions sont autorisées) et l'utilisation du paradigme "Propositions as types" est particulièrement adaptée à la notion d'analogie. D'autre part, Brock et al. [BCP88], ont développé des méthodes d'analogie, propres à la preuve de propriétés sur les réels; et récemment, Kolbe et Walther [KW94] ont proposé des mécanismes adaptés à la preuve par induction. Parallèlement, Owen [Owe90] et Melis [Mel93a], par exemple, ont mené des travaux plus axés sur le raisonnement. Mais à n'en pas douter, l'analogie est un concept pour lequel l'intérêt ne fera que grandir.

De par les objectifs que nous nous sommes fixés, notre approche diffère légèrement de ces travaux. En effet, nous nous sommes intéressés au problème de l'automatisation, afin que nos méthodes soient de véritables outils. Les preuves de spécifications de programmes ne sont pas des preuves complexes mathématiquement, mais sont formées de nombreuses preuves simples, voire triviales. Nous avons donc cherché dans un premier temps, à définir des notions d'analogie qui permettent, à coup sûr et en un temps fini, de passer d'une preuve à la preuve cherchée. Cela ne permet évidemment pas de traiter des cas très généraux, pourtant, nous verrons que les exemples donnés peuvent être intéressants. De plus, nous réutilisons ces outils élémentaires pour aborder des mécanismes d'analogies plus complexes.

La seconde originalité de notre approche est que nous tirons l'information qui dictera l'analogie de la similitude – ou de la différence – entre les formules avant de passer au stade de la preuve. Si l'on reprend les deux syllogismes donnés en début d'introduction, nous pouvons affirmer que ces propositions sont similaires, parce que nous pouvons construire une relation liant le prédicat *"est un homme"* et le prédicat *"aime les sucettes à l'anis"*, etc... En conséquence, un élément essentiel de notre façon de concevoir l'analogie est l'outil qui permet de calculer la similitude et la différence entre les formules. D'autre part, pour définir des schémas de formules du premier ordre, il est nécessaire d'utiliser des formules d'ordre deux. C'est pourquoi la composante principale de ce calcul est un algorithme de filtrage d'ordre deux. En remplaçant les symboles de prédicat et de fonction par des variables dans la formule prise comme référence, nous obtenons un schéma de formules. Si le filtrage entre ce schéma et la conjecture à prouver a des solutions, une preuve de la conjecture peut être construite à partir de la preuve de la formule de référence. C'est une façon forte de reconnaître l'analogie, dans le sens où il suffit de donner les deux formules en entrée avec une preuve de la formule prise comme référence, pour obtenir en sortie, une preuve de la conjecture, si le filtrage n'échoue pas. Dans ces conditions, il n'y a aucun traitement particulier à apporter aux formules pour appliquer l'analogie. Par ailleurs, nous considérons d'autres critères de ressemblance. L'idéal serait de les intégrer au filtrage, comme nous l'avons fait pour les propriétés AC des connecteurs. En effet, cet algorithme de filtrage nous permet de traiter de façon transparente ces propriétés des connecteurs, et il serait avantageux, à terme, de traiter toutes les similitudes de cette façon. Mais si, dans le cas AC, l'algorithme de filtrage reste décidable, nous ne pouvons actuellement rien dire, en ce qui concerne l'intégration des autres similitudes traitées. Par conséquent, ces similitudes sont traitées d'une façon que nous

pouvons appeler faible. C'est-à-dire qu'il s'agit d'une manipulation des formules. C'est le cas de la similitude d'ordre propositionnelle dont il est question au premier chapitre, mais aussi de la similitude modulo la position des quantificateurs, traitée au chapitre III. Savoir si nous sommes capables de pratiquer l'analogie entre deux formules selon ce dernier critère, revient à savoir si ces deux formules possèdent une forme prénexe commune – ou ont même forme anti-prénexe – ce qui demande, bien sûr de manipuler ces formules.

Enfin, les méthodes décrites dans la majeure partie de cette thèse sont appliquées aux preuves par expansion. Elles constituent une alternative à ce que nous aurions pu faire avec le calcul des séquents ou la déduction naturelle par exemple. Mais si nous avons choisi cette représentation, c'est parce qu'elle possède des propriétés précieuses pour l'analogie.

La complexité croissante des types d'analogie abordés constitue le fil conducteur de ce document. Au chapitre I, nous présentons les preuves par expansion – dont les deux composantes sont les arbres d'expansion et les connexions –, puis la première forme d'analogie. En parlant des connexions, Peter Andrews a écrit qu'elles sont *"la raison pour laquelle la formule est valide"*. Il semblerait que nous retombions dans le domaine de l'intuition, mais en réalité, nous aurons de multiples occasions de vérifier le bien fondé de cette affirmation. Les preuves par expansion séparent l'aspect structurel de la preuve de l'aspect validité qui est en quelque sorte matérialisé par les connexions. Par conséquent, nous étudions dans le premier chapitre des transformations de preuves correspondant à des formules ayant les mêmes *"raisons d'être valides"*. Par exemple, intuitivement, la formule $\forall x\ p(x) \Rightarrow \exists x\ p(x)$ a la même raison d'être valide que la formule $\forall x\ p(x) \Rightarrow \exists x\ (p(x) \vee q(x))$, puisqu'en réalité, la seconde contient la première. Nous verrons qu'il est possible de passer de la preuve de l'une de ces formules à la preuve de l'autre de façon automatique. Ce qui veut dire qu'il est possible de passer d'une formule donnée à une formule plus compliquée, mais aussi, de simplifier une formule en même temps que sa preuve. Ce type de transformations fait intervenir une relation entre formules comparable à l'enchâssement.

Le chapitre II est consacré au filtrage. Comme nous l'avons dit, c'est le fondement de la reconnaissance de similitudes entre les formules. Lorsqu'il a des solutions, l'analogie peut se faire. Mais lorsqu'il n'en a pas, il est encore possible de l'exploiter – en analysant ses échecs – pour proposer des méthodes d'analogie plus ambitieuses. Le filtrage d'ordre deux est connu depuis longtemps [HL78], et c'est un outil puissant qui possède le gros avantage d'être décidable. Mais il est encore très restrictif dans le sens où il ne permet pas de prendre en compte les propriétés algébriques des opérateurs. Or, dans notre cas, cela signifie que nous ne serions pas capables de reconnaître l'analogie entre $A \wedge B$ et $B \wedge A$, ou plus généralement, entre deux formules équivalentes modulo l'associativité-commutativité (AC) des connecteurs logiques. C'est pourquoi nous avons intégré cette théorie au filtrage d'ordre deux. La théorie AC offre un bon compromis entre l'expressivité et les possibilités d'obtenir un algorithme utilisable en pratique. De plus, on se heurte rapidement à des problèmes – comme la perte de la complétude ou de la terminaison – lorsqu'on essaie de combiner le filtrage d'ordre deux avec des théories plus complexes. Nous avons donc construit un algorithme réalisant la combinaison entre le filtrage dans la théorie AC et l'ordre deux. Mais la première forme de cet algorithme a fait apparaître un certain nombre de faiblesses, comme la redondance excessive des solutions, et l'inefficacité en général. Ce constat a donné lieu à un nouvel algorithme remédiant à ces problèmes et dont l'approche diffère sensiblement de l'approche classique. Nous avons eu la chance, pour cela, de travailler en collaboration avec Zhenyu Qian, qui avait auparavant – et entre autres – proposé des algorithmes combinant l'unification d'ordre supérieur et l'unification équationnelle [NQ91, QW92].

Au cours du chapitre III, nous examinons certains critères de similitude entre les formules.

Tout d'abord, nous exploitons l'algorithme de filtrage modulo AC. En effet, celui-ci permet de traiter simultanément et sans traitement sur les formules, les similitudes que nous appelons correspondance – qui est un peu plus générale que le renommage – et les propriétés AC des connecteurs. Ceci montre combien l'intégration des critères de similitude au filtrage est souhaitable. Par ailleurs, et cette fois de façon déconnectée du filtrage, nous abordons le problème du positionnement des quantificateurs dans les formules : notre ambition étant de transformer la preuve d'une formule en preuve d'une autre formule lorsque celles-ci sont équivalentes modulo le positionnement des quantificateurs. Le principal problème réside dans la reconnaissance de cette équivalence. En effet, la forme prénexe n'apporte pas une réponse satisfaisante, puisqu'elle n'est pas unique. C'est pourquoi nous utilisons une forme anti-prénexe, qui elle, est unique pour une formule donnée. Ainsi, lorsque deux formules possèdent la même forme anti-prénexe – ou ont une forme prénexe commune –, nous avons une transformation automatique des preuves. A ce stade, on pourrait objecter qu'il s'agit de formules équivalentes du point de vue de la logique, et que ce type de transformations est inutile puisque l'on peut considérer que ces formules sont les mêmes. Mais, en définitive, toutes les formules valides sont équivalentes à la formule $Vrai$... Avoir une méthode capable de procéder automatiquement à l'analogie lorsque les formules concernées sont équivalentes modulo une correspondance, les propriétés AC des connecteurs et le positionnement des quantificateurs ne paraît pas si dérisoire. En effet, cela permet par exemple de prouver sans intervention de l'utilisateur la formule $\forall x\ (\forall y\ A(y,x) \wedge \exists z\ B(x,z))$ à partir d'une preuve de $\forall x\ \exists z\ \forall y\ (A(y,x) \wedge B(x,z))$ et réciproquement.

Le chapitre IV affine la recherche d'analogie entre les formules, avec l'introduction du filtrage de différence. Il s'agit d'un moyen de savoir ce qui différencie deux formules, en utilisant les échecs du filtrage. Une idée similaire a été introduite par Boy de la Tour et Caferra dans [BdlTC88]. Elle consiste à filtrer les types de formules, dans le cadre de l'isomorphisme de Curry-Howard, et de garder les échecs pour générer des lemmes intermédiaires. Ainsi, grâce à ce que nous appelons le filtrage de différence, nous calculons les parties similaires et les parties réellement différentes de deux formules. Si l'on considère que ce qui est similaire peut être traité automatiquement, c'est sur la différence qu'il faut concentrer nos efforts. Partant d'une différence élémentaire, qui consiste à remplacer un symbole de constante par un autre dans une formule, nous avons essayé de produire un mécanisme qui conserve la propriété de complétude. C'est-à-dire qu'il construit une preuve de la formule objet si et seulement si celle-ci est valide. Nous avons obtenu un processus complet, mais seulement sous certaines conditions sur le théorème utilisé comme référence pour l'analogie. Cette méthode permet néanmoins d'automatiser des exemples de référence du domaine. Notons au passage qu'il s'agit maintenant de formules qui ne sont pas nécessairement équivalentes, puisque la conjecture à prouver n'est pas forcément valide.

Au chapitre V, nous abordons l'analogie d'un point de vue plus général, sans pour autant abandonner ce qui a été fait auparavant. En effet, les techniques décrites jusque là forment un ensemble d'outils réutilisables. Par contre, ce qui a été défini pour les preuves par expansion doit être adapté à d'autres formalismes. Nous proposons dans ce chapitre une classification possible des analogies. A chacune des classes définies, correspond une méthode. Le but est de trouver la meilleure façon de réutiliser complètement la preuve utilisée comme référence. Ainsi, partant du cas simple où l'analogie est immédiate, nous arrivons au cas général, où des lemmes sont générés. Mais les méthodes décrites ne font jamais appel à un démonstrateur. C'est évidemment ce qu'il faudrait faire en pratique, mais dans ce cas, il est souvent difficile de distinguer ce qui doit être attribué au démonstrateur, de ce qui doit l'être au mécanisme d'analogie lui-même. Or nous voulons dégager plus clairement les possibilités de méthodes entièrement basées sur l'analogie

afin d'en estimer la portée. Il est bien évident que l'analogie ne se conçoit que comme une aide à la déduction automatique, et non pas comme un système de déduction autonome. Chacune des formes d'analogie que nous présentons est illustrée d'un ou plusieurs exemples. C'est l'occasion de montrer que le principe d'analogie est indépendant du formalisme. Les exemples donnés concernent la synthèse de programmes, l'analyse, la logique, etc... Ces exemples confirment également que le filtrage de différence constitue un outil de base. Pour finir ce chapitre, nous résumons la progression suivie globalement au cours du document, et procédons à une comparaison avec les travaux voisins. Enfin, la conclusion dresse un bilan et donne les directions dans lesquelles ce travail devrait être poursuivi.

I
Analogie simple

Ce chapitre introduit la première méthode de preuve par analogie que nous allons étudier. Il s'agit d'établir un processus capable de transformer automatiquement la preuve d'une formule en preuves d'autres formules auxquelles elle est liée par une relation propositionnelle. Par exemple, de façon très simplifiée, si nous possédons une preuve de A, nous devons être capables de construire une preuve de $A \vee B$. Il semble en effet, qu'avant d'essayer de modéliser le raisonnement par analogie, il est nécessaire de savoir appliquer des principes simples, sur lesquels pourrons reposer des processus plus élaborés. De plus, si l'on formule une requête à un démonstrateur, afin qu'il construise une preuve de A, puis une preuve de $A \vee B$, celui-ci reprendra vraisemblablement la même procédure; y compris si la preuve de A est compliquée et coûteuse. Il ne semble donc pas totalement futile de se pencher sur ce type de problèmes. D'autant que les méthodes décrites ici vont constituer une base d'outils qui seront réutilisés pour des cas plus complexes. Ce chapitre contient en outre, les définitions relatives aux formules utilisées tout au long du document, ainsi que l'introduction des preuves par expansion, largement exploitées dans la suite. Nous verrons d'ailleurs que l'utilisation de cette représentation de preuves est motivée par bien des aspects.

I.1 Idée de la méthode

Nous présentons ici de façon informelle un type de preuves par analogie que nous appellerons *simple*. Comme pour toute analogie, le point de départ est une formule valide A, une preuve de cette formule, notée $P(A)$, et une formule objet B. La figure I.1 illustre notre façon de voir ce problème.

Le point de vue développé ici est celui selon lequel l'analogie entre les preuves trouve son fondement dans l'analogie entre les formules. Le processus de preuve par analogie est alors représenté par l'implication du schéma de la figure I.1 qui fait le lien entre la relation liant les formules et la relation liant les preuves. Pour mener à bien cette idée, l'information de base de cette méthode doit provenir de la relation qui existe entre les deux formules. Si l'on essaie de prouver la formule B par analogie à la preuve de la formule A, c'est que l'on estime que *"B ressemble à A"*, que *"B a la même structure que A"*, ou que *"B a les mêmes raisons que A d'être valide"*. Nous devons donc, en premier lieu, définir formellement une relation qui représente ces idées intuitives. Dans le présent chapitre, cette relation est simple. Elle repose sur des principes propositionnels, et elle constitue une première étape. Elle sera réutilisée par la suite pour des cas plus complexes. Néanmoins, l'intégration de ce type de processus dans un démonstrateur

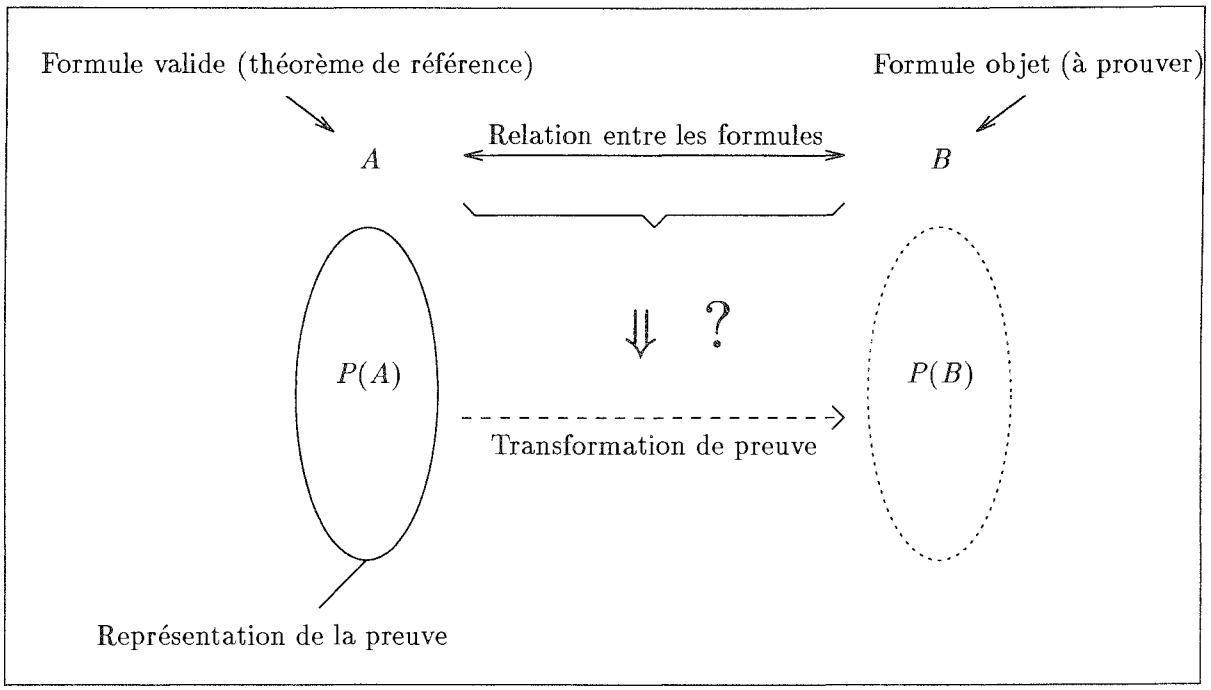


Figure I.1: Schématisation de la preuve par analogie

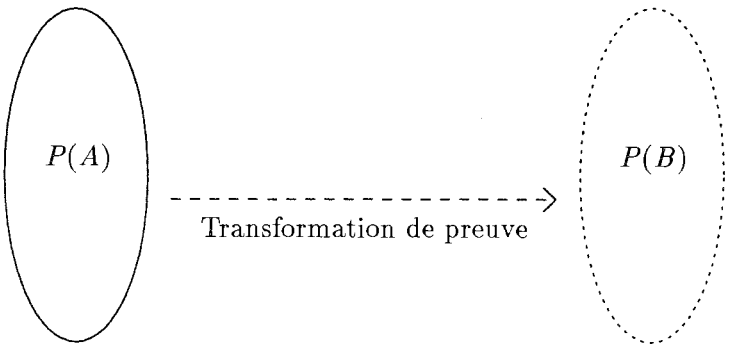
peut être déjà utile puisqu'elle permet d'éviter, de façon totalement automatique, de refaire des calculs similaires à ceux déjà effectués.

Supposons que A est la formule de référence dont on possède une preuve $P(A)$, et B, celle que l'on vise à prouver. Si A et B satisfont la relation $A \preceq B$ définie dans ce chapitre, d'une part, $A \Rightarrow B$ est une formule valide, et d'autre part, nous sommes en mesure de produire automatiquement et en un temps fini une preuve de B à partir de $P(A)$. En d'autres termes, $P(A)$ peut être transformée en $P(B)$ lorsque $A \preceq B$. La première étape consiste donc à définir la relation qui lie A et B :

$$A \quad \xleftrightarrow{\text{Relation entre les formules}} \quad B$$

Cette relation est une sorte d'enchâssement de A dans B. Pour représenter les preuves, nous utilisons les *Preuves par expansion* introduites par Miller [Mil87] et reprises par Pfenning [Pfe87]. Le principe des preuves par expansion est basé sur les expansions au sens de Herbrand. Ce type de preuves est constitué de deux éléments : un arbre d'expansion et un ensemble de connexions :

- l'arbre d'expansion représente la formule à prouver. La principale conséquence est que la représentation de la preuve est très proche de la représentation de la formule, ce qui amène à penser que la relation entre les formules doit, elle aussi, être proche de la relation entre les preuves. En effet, le fait que les formules et les preuves s'expriment de façon voisine implique que la transformation entre les preuves:



doit s'inspirer directement de la relation

$$A \quad \xleftarrow{\text{Relation entre les formules}} \quad B$$

Nous verrons effectivement que la transformation de $P(A)$ en $P(B)$ est calquée sur la définition de la relation $A \preceq B$.

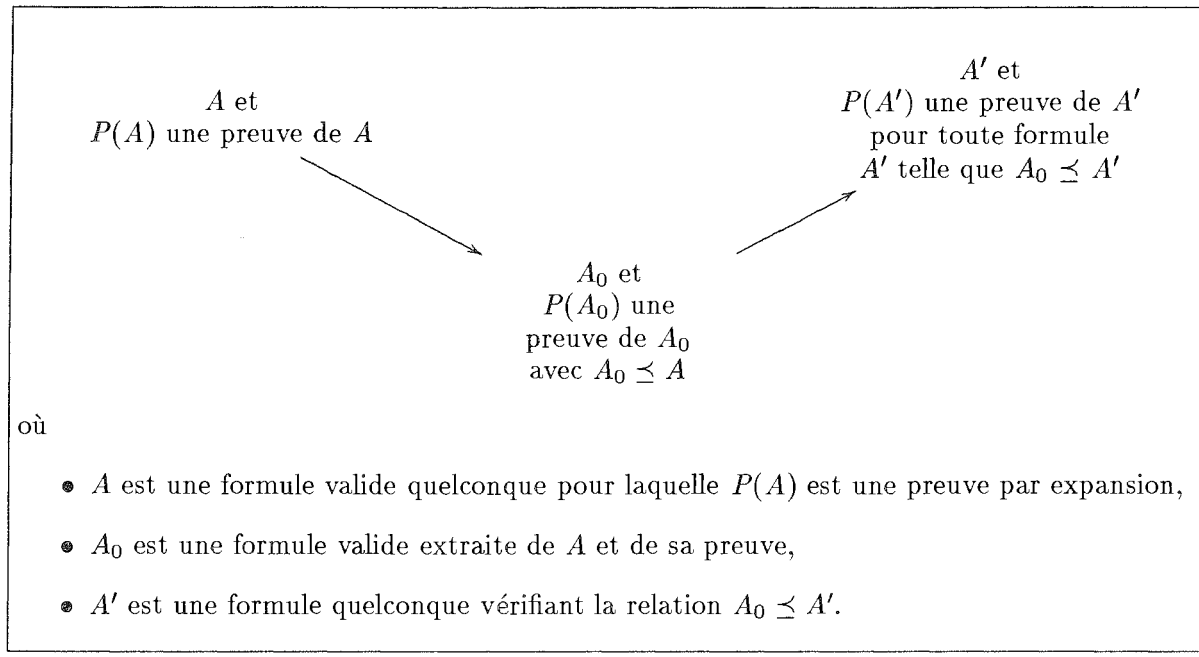
- La seconde composante d'une preuve par expansion est un ensemble de connexions. Pour Andrews [And86], qui, parallèlement à Bibel [Bib81], a été l'un des premiers à les utiliser pour la déduction automatique, les connexions expriment la *raison* pour laquelle la formule prouvée est valide. En ce sens, nous pouvons dire que les preuves par analogie développées dans ce chapitre utilisent des formules ayant les mêmes raisons d'être valides. Cette notion, qui semble très intuitive, sera précisée avec les définitions formelles.

Les preuves par expansion présentent en réalité bon nombre d'avantages et de propriétés qui nous seront utiles. De plus, elles ont été conçues pour supporter la logique d'ordre supérieur, ce qui nous permet d'envisager l'extension de ces travaux à l'ordre supérieur dans un avenir proche.

Les preuves par expansion possèdent donc deux composantes bien distinctes. La première est structurelle – l'arbre d'expansion – et la seconde est une relation entre certains éléments de cette structure – les connexions. Les algorithmes que nous allons décrire ici ne manipulent que la structure et en aucun cas les connexions. C'est en cela que l'on peut se permettre de dire que nous manipulons des formules ayant "les mêmes raisons d'être valides". De façon approximative, nous pouvons dire que la structure représentant la formule A est transformée en une structure représentant la formule B, mais que nous conservons l'information essentielle contenue dans les connexions. Ce qui peut être symbolisé de la façon suivante :

$$P(A) : \begin{cases} Q : & \text{arbre d'expansion} \\ & \text{représentant } A \\ \mathcal{M} : & \text{connexions de } Q \end{cases} \quad \rightarrow \quad P(B) : \begin{cases} Q' : & \text{arbre d'expansion} \\ & \text{représentant } B \\ \mathcal{M} : & \text{connexions de } Q' \end{cases}$$

Nous insistons sur le fait que la connexion $\mathcal{M}$ est la même dans $P(A)$ et $P(B)$. D'autre part, le choix du théorème de référence peut s'avérer important. Ce théorème doit permettre de prouver un maximum d'autres formules par analogie. Pour cela, on doit le simplifier pour



où

- A est une formule valide quelconque pour laquelle $P(A)$ est une preuve par expansion,
- A_0 est une formule valide extraite de A et de sa preuve,
- A' est une formule quelconque vérifiant la relation $A_0 \preceq A'$.

Figure I.2: Schéma des transformations réalisées

ne garder que la partie essentielle de la formule valide. Par exemple, il est inutile de prendre la formule $A \vee B$ comme référence si l'on sait que la formule A elle-même, est valide. Ainsi, on pourra prouver toute formule dans laquelle A est enchâssée, plutôt que $A \vee B$, ce qui est bien sûr plus puissant. Pour simplifier cette formule de référence, il faut utiliser l'information contenue dans les connexions liées à la preuve. Nous serons ainsi capables, à partir d'une formule A et de sa preuve $P(A)$, d'extraire une formule A_0 et sa preuve $P(A_0)$ telles que $A_0 \preceq A$ selon la définition donnée plus loin pour la relation $\preceq$, avec $P(A_0)$ contenant les mêmes connexions que $P(A)$. Nous aurons ainsi conservé l'information essentielle de la preuve, et simplifié la formule prouvée. En résumé, les transformations présentées dans ce chapitre peuvent être schématisées par la figure I.2. Notons que la transformation qui permet de passer de la formule A à la formule A_0 peut être vue comme une généralisation de théorème. Ce qui peut constituer un intérêt en soi, indépendamment de l'analogie.

Ces transformations sont correctes, terminent et sont complètes. Le but de ce chapitre n'est pas uniquement d'appliquer l'analogie à la relation $\preceq$. Il est aussi de construire des outils que nous pourrons réutiliser par la suite.

I.2 Formules

Nous introduisons dans cette section les formules du calcul des prédicats du premier ordre. Lorsque nous en aurons besoin, c'est-à-dire au chapitre II, nous donnerons la définition des λ-termes simplement typés que nous utiliserons pour représenter ces formules.

Définition 1 A partir de la signature $\Sigma_f \cup \Sigma_r \cup \mathcal{V}$ où

- Σ_f est un ensemble dénombrable de symboles de fonction,
- Σ_r est un ensemble dénombrable de symboles de relation (fonction à valeur de vérité),
- $\mathcal{V}$ est un ensemble dénombrable de symboles de variables d'arité 0,

on construit l'ensemble des termes finis habituellement noté $T_{\Sigma_f}(\mathcal{V})$. □

Nous construisons les formules de la façon suivante.

Définition 2
Formules :

▷ Un atome $r(t_1, \ldots, t_n)$ avec $r \in \Sigma_r$ est une formule atomique.

▷ $\neg t$ est une formule pour toute formule t.

▷ $t \wedge t'$ est une formule pour toutes formules t et t'.

▷ $t \vee t'$ est une formule pour toutes formules t et t'.

▷ $\forall x\ t$ est une formule pour toute variable x et toute formule t.

▷ $\exists x\ t$ est une formule pour toute variable x et toute formule t.

□

Les opérateurs $\neg$, $\wedge$ et $\vee$ sont des connecteurs, $\forall$ et $\exists$ sont des quantificateurs.

Définition 3 Dans une formule $Qx\ t$ où Q est un quantificateur, les occurrences de x dans t sont *liées* par Q. □

Nous considérons les formules logiques en *forme normale négative* (FNN). Rappelons que le système ci-dessous transforme toute formule en sa forme normale négative qui est unique modulo le renommage des variables liées.

Transformation en forme normale négative :

▷ $\neg\neg A \rightarrow A$

▷ $\neg(A \vee B) \rightarrow \neg A \wedge \neg B$

▷ $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$

▷ $\neg(\forall x A) \rightarrow \exists x \neg A$

▷ $\neg(\exists x A) \rightarrow \forall x \neg A$

La priorité implicite des opérateurs est, par ordre décroissant :

- $\forall$, $\exists$, $\neg$
- $\wedge$
- $\vee$

Lorsque A est une formule atomique (atome), A ou $\neg A$ est un *littéral*. Nous utiliserons également l'opérateur $\Rightarrow$, bien que théoriquement, nous ne manipulions que des formules en FNN. La transformation $A \Rightarrow B \rightarrow \neg A \vee B$ est alors implicite. Enfin, la notation $\models A$ signifie que la formule A est valide, où valide s'entend au sens de la logique classique. Les formules que nous manipulons étant celles de la logique du premier ordre.

Nous utiliserons les conventions suivantes : X, Y, Z, F, G représentent des variables libres. A, B, C et t, u, v représentent des formules.

I.3 Relation entre les formules

Nous présentons ici notre première notion de similitude entre les formules. Elle est basée sur une relation de type propositionnel. Notons que des travaux ont déjà été faits sur la simplification booléenne de formules du premier ordre. En effet, Socher [Soc87] a défini une méthode décidant l'équivalence de deux formules logiques du premier ordre modulo l'associativité, la commutativité et l'idempotence des connecteurs logiques. Cette méthode utilise l'isomorphisme entre graphe. Mais pour notre part, il s'agit de donner une relation telle que nous soyons capables de transformer la preuve d'une des formules en preuve de l'autre. Par conséquent, notre relation est moins puissante.

I.3.1 La relation $\preceq$

A présent, nous donnons la définition de la relation $\preceq$. Comme nous l'avons dit précédemment, il s'agit d'une sorte d'enchâssement à ceci près qu'il est adapté aux formules logiques de façons à respecter certaines propriétés.

La relation $\preceq$ sur les formules

0- $A \preceq A$

1- $A \preceq A_1 \vee A_2$ si $A \preceq A_1$ ou $A \preceq A_2$

2- $A \preceq A_1 \wedge A_2$ si $A \preceq A_1$ et $A \preceq A_2$

3- $A_1 \vee A_2 \preceq B_1 \vee B_2$ si $A_1 \preceq B_1$ et $A_2 \preceq B_2$

4- $A_1 \wedge A_2 \preceq B_1 \wedge B_2$ si $A_1 \preceq B_1$ et $A_2 \preceq B_2$

5- $\exists x A \preceq \exists x B$ si $A \preceq B$

6- $\forall x A \preceq \forall x B$ si $A \preceq B$

Lorsque la propriété $A \preceq B$ est vérifiée, nous dirons que B est moins générale que A. Quand A et B ne sont pas comparables, c'est à dire $A \not\preceq B$ et $B \not\preceq A$, nous le noterons $A \not\sim B$. L'exemple qui suit donne une idée intuitive de cette relation, tant du point de vue de sa signification que de celui de l'utilisation que nous allons en faire.

Exemple 4

Prenons la formule $A = \forall xy\ (p(x,y) \wedge q(y)) \Rightarrow \exists z\ p(a,z)$

On voit aisément que l'hypothèse $q(y)$ n'est pas utile pour prouver A. Il doit donc exister une formule plus simple que A qui ait les mêmes raisons que A d'être valide. Considérons par exemple

$$A_0 = \forall xy\ p(x,y) \Rightarrow \exists z\ p(a,z),$$

cette formule est valide et satisfait la relation $A_0 \preceq A$. Le premier objectif de la présente méthode consiste à épurer la formule de départ afin d'en extraire une formule plus simple – d'après notre relation – et qui soit toujours valide. En général, ce raffinement de la formule permet d'éliminer des hypothèses inutiles. Il faut signaler que cette modification de la formule se fait grâce à l'information qui est contenue dans les connexions de la preuve qui est liée à cette formule, comme nous le verrons dans la section I.5.2. Prenons maintenant

$$B_1 = \forall xy\ p(x,y) \Rightarrow \exists z\ (p(a,z) \vee p(z,a)),$$

là encore, intuitivement, l'information nécessaire pour montrer B_1 doit se trouver dans la preuve de A_0. Effectivement, la relation $A_0 \preceq B_1$ est valide, et nous allons être en mesure de construire une preuve de B_1 à partir de la preuve de A_0. Il en est de même pour la formule

$$B_2 = \forall xy\ p(x,y) \wedge \exists x\ \neg q(x) \Rightarrow \exists z\ (p(a,z) \vee p(z,a))$$

qui satisfait également la relation $A_0 \preceq B_2$. Le second objectif de cette méthode est donc d'être capable de construire une preuve de n'importe quelle formule B vérifiant $A_0 \preceq B$ à partir de la preuve de la formule A_0 déduite de A et de sa preuve. ■

Le lemme qui suit formule une propriété essentielle de cette relation.

Lemme 5 *Si $A \preceq B$, alors $\models A \Rightarrow B$.*

Preuve Par induction sur la définition de $\preceq$. □

Lemme 6 *Si $A \preceq B$ et $B \preceq C$, alors $A \preceq C$.*

Preuve Par induction structurelle sur les formules. D'après la définition de $\preceq$, il est clair que le nombre de symboles de A est plus petit ou égal au nombre de symboles de B. On examine alors tous les cas où A, B et C sont comparables.
Cas de base : C est un littéral. Alors, $B \preceq C$ implique $B = C$ et $A \preceq B$ implique $A = B$. Le cas est donc évident.
Cas inductifs :

1 $C = C_1 \vee C_2$.

- **1.1** $B \preceq C_1$ (ou $B \preceq C_2$).
  $A \preceq B$ et $B \preceq C_1$ implique, par hypothèse d'induction $A \preceq C_1$. D'où, par la règle 1 de la définition, $A \preceq C$. (ou $A \preceq B$ et $B \preceq C_1$ implique, par induction $A \preceq C_2$...)
- **1.2** $B = B_1 \vee B_2$ avec $B_1 \preceq C_1$ et $B_2 \preceq C_2$.

1.2.1 $A \preceq B_1$ (ou $A \preceq B_2$)
$A \preceq B_1 \preceq C_1$ implique, par induction $A \preceq C_1$. D'où, $A \preceq C$. De la même façon pour $A \preceq B_2 \preceq C_2$.

1.2.2 $A = A_1 \vee A_2$ avec $A_1 \preceq B_1$ et $A_2 \preceq B_2$.
$A_i \preceq B_i \preceq C_i$ pour $i = 1, 2$. D'où, par hypothèse d'induction, $A_i \preceq C_i$ pour $i = 1, 2$. Donc, $A \preceq C$.

2 $C = C_1 \wedge C_2$.

2.1 $B \preceq C_1$ et $B \preceq C_2$.
$A \preceq B \preceq C_i$ pour $i = 1, 2$. Soit, par induction, $A \preceq C_i$ pour $i = 1, 2$. On a donc $A \preceq C$.

2.2 $B = B_1 \wedge B_2$ avec $B_1 \preceq C_1$ et $B_2 \preceq C_2$.

2.2.1 $A \preceq B_1$ et $A \preceq B_2$.
$A \preceq B_i \preceq C_i$ implique, par induction, $A \preceq C_i$ pour $i = 1, 2$. On a donc $A \preceq C$.

2.2.2 $A = A_1 \wedge A_2$ avec $A_1 \preceq B_1$ et $A_2 \preceq B_2$.
$A_i \preceq B_i \preceq C_i$ pour $i = 1, 2$. Par induction, $A_i \preceq C_i$ pour $i = 1, 2$ d'où $A \preceq C$.

3 $C = \exists x C_1$.
Par conséquent, $B = \exists x B_1$, et $A = \exists x A_1$ avec $A_1 \preceq B_1 \preceq C_1$. D'où, par hypothèse d'induction, $A_1 \preceq C_1$ et donc, $A \preceq C$.

4 $C = \forall x C_1$.
De la même façon, $B = \forall x B_1$, et $A = \forall x A_1$ avec $A_1 \preceq B_1 \preceq C_1$. Par hypothèse d'induction, $A_1 \preceq C_1$, d'où $A \preceq C$.

□

Nous aurons également besoin par la suite de la propriété suivante qui stipule que le remplacement d'un sous-terme par une variable conserve la relation $\preceq$.

Lemme 7 *Pour toutes formules $A(t)$ et $B(t)$,*

$$B(t) \preceq A(t) \quad \Rightarrow \quad B(x) \preceq A(x)$$

où toutes les occurrences de t sont remplacées par x et où t ne peut avoir plus d'occurrences dans $B(t)$ que dans $A(t)$.

Preuve Par induction structurelle.

- Cas de base : $B(t) = A(t)$. Alors, $B(x) = A(x)$, d'où $B(x) \preceq A(x)$.
- $A(t) = A_1(t) \vee A_2(t)$ avec $B(t) \preceq A_1(t)$ (resp. $B(t) \preceq A_2(t)$). Par hypothèse d'induction, $B(x) \preceq A_1(x)$ (resp. $B(x) \preceq A_2(x)$). D'où $B(x) \preceq A(x)$.
- $A(t) = A_1(t) \wedge A_2(t)$ avec $B(t) \preceq A_1(t)$ et $B(t) \preceq A_2(t)$. Par induction, $B(x) \preceq A_i(x)$ pour $i = 1, 2$. D'où $B(x) \preceq A(x)$.

- $A(t) = A_1(t) \vee A_2(t)$, $B(t) = B_1(t) \vee B_2(t)$, et $B_1(t) \preceq A_1(t)$ et $B_2(t) \preceq A_2(t)$. Par induction, $B_i(x) \preceq A_i(x)$ pour $i = 1, 2$. D'où $B(x) \preceq A(x)$.
- $A(t) = A_1(t) \wedge A_2(t)$, $B(t) = B_1(t) \wedge B_2(t)$, et $B_1(t) \preceq A_1(t)$ et $B_2(t) \preceq A_2(t)$. Par induction, $B_i(x) \preceq A_i(x)$ pour $i = 1, 2$. D'où $B(x) \preceq A(x)$.
- $A(t) = \exists y\ A'(y,t)$ et $B(t) = \exists y\ B'(y,t)$ avec $B'(y,t) \preceq A'(y,t)$. Par induction, $B'(y,x) \preceq A'(y,x)$. D'où $B(x) \preceq A(x)$.
- $A(t) = \forall y\ A'(y,t)$ et $B(t) = \forall y\ B'(y,t)$ avec $B'(y,t) \preceq A'(y,t)$. Par induction, $B'(y,x) \preceq A'(y,x)$. D'où $B(x) \preceq A(x)$.

□

Maintenant que nous avons défini notre relation et ses propriétés de base, nous allons nous intéresser à la représentation des preuves.

I.4 Preuves par expansion

Notre but est de transposer les différences entre les formules aux différences entre les preuves. C'est pourquoi une représentation des preuves qui est proche de la représentation des formules nous convient tout particulièrement. Les preuves par expansion constituent une méthode directe, et non pas réfutationnelle. Par conséquent, la structure de la formule est préservée, puisque nous n'avons pas à en prendre la négation. Ce qui est important dans notre cas, puisque lorsque l'on tente de prouver une formule par analogie à une autre, on suppose qu'elles ont même structure ou que leurs preuves doivent avoir même structure. Enfin, cette structure est séparée de l'information détenue par les connexions qui est tout à fait essentielle.

Nous allons, avant de passer aux définitions formelles, illustrer les composantes des preuves par expansion. Situons-nous dans le calcul propositionnel pour commencer. Pour faire apparaître la structure d'une tautologie, Andrews [And86] utilise la notation matricielle introduite par Bibel [Bib79]. Considérons par exemple la tautologie T suivante :

$$T = (A \wedge B) \vee \neg A \vee (A \wedge \neg B)$$

En distribuant la disjonction sur la conjonction, on obtient la conjonction suivante :

$$\begin{array}{ll} & (A \vee \neg A \vee A) \\ \wedge & (A \vee \neg A \vee \neg B) \\ \wedge & (B \vee \neg A \vee A) \\ \wedge & (B \vee \neg A \vee \neg B) \end{array}$$

Ainsi, pour éviter cette distribution explicite, l'écriture sous forme de matrice représente la disjonction horizontalement, et la conjonction verticalement. On obtient pour notre exemple :

$$\begin{bmatrix} A \\ B \end{bmatrix} \vee [\neg A] \vee \begin{bmatrix} A \\ \neg B \end{bmatrix}$$

Pour retrouver la conjonction, il suffit de considérer les chemins traversant cette matrice de gauche à droite. Pour notre exemple, il en existe quatre :

$$A \vee \neg A \vee A$$
$$A \vee \neg A \vee \neg B$$
$$B \vee \neg A \vee A$$
$$B \vee \neg A \vee \neg B$$

Ces chemins représentent les éléments de la conjonction, et T est une tautologie si chacun de ces chemins est une tautologie. Dans la définition des preuves par expansion que nous allons voir, les chemins sont caractérisés par les *clauses complètes* (*cc*), et la conjonction des chemins par la *conjonction des clauses complètes* ($\wedge$cc). Pour montrer que T est une formule valide, il suffit donc de montrer que chacun des éléments de cette conjonction – qui est une disjonction représentée par un chemin – est une tautologie, ce qui revient à montrer que chaque chemin contient deux formules A et $\neg A$. Par exemple pour la formule T, chacun des chemins est de la forme $C \vee D \vee \neg D$. L'idée des connexions est donc de construire une relation définie par des couples $(D, \neg D)$, telle que chaque chemin contienne l'un de ces couples. Si c'est le cas, cette connexion *couvre* la conjonction des clauses complètes. Pour la formule T, la connexion constituée des trois couples

$$\{(A, \neg A);\ (B, \neg B);\ (\neg A, A)\}$$

couvre la conjonction, puisque dans chacun de ses éléments, on retrouve l'un de ces couples :

$$\begin{array}{ll} & \boxed{A} \vee \boxed{\neg A} \vee A \\ \wedge & \boxed{A} \vee \boxed{\neg A} \vee \neg B \\ \wedge & B \vee \boxed{\neg A} \vee \boxed{A} \\ \wedge & \boxed{B} \vee \neg A \vee \boxed{\neg B} \end{array}$$

L'existence d'une telle connexion est en quelque sorte la concrétisation de la validité de la formule. D'où l'affirmation d'Andrews selon laquelle cette connexion représente la raison pour laquelle la formule est valide.

Passons maintenant au calcul des prédicats. Pour une formule $F = \exists x\ p(x)$, s'il existe une instance a de x telle que $p(a)$ est une tautologie, alors F est valide. Nous pouvons voir la formule $p(a) \vee \ldots \vee p(b)$ comme une *expansion tautologique* de la formule F. Si nous considérons maintenant la formule $\exists y\ \forall x\ (p(x) \Rightarrow p(y))$, une expansion de cette formule peut être $\forall x\ (p(x) \Rightarrow p(a)) \vee \forall x\ (p(x) \Rightarrow p(b))$. En remplaçant x par b dans le premier membre et par c dans le second, nous obtenons $(p(b) \Rightarrow p(a)) \vee (p(c) \Rightarrow p(b))$. C'est-à-dire, en forme normale négative, $\neg p(b) \vee p(a) \vee p(c) \vee p(b)$. C'est une instance tautologique de cette formule où nous pouvons considérer b et c comme des paramètres et a et b comme des instances quelconques. La représentation matricielle de cette instance est simplement

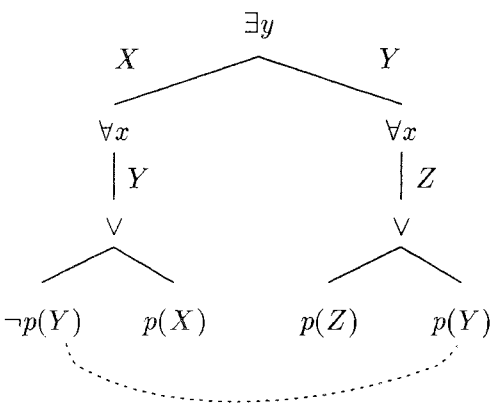
$$[\neg p(b)] \vee [p(a)] \vee [p(c)] \vee [p(b)]$$

Tous les chemins – il n'en existe qu'un – sont couverts par la connexion $\{(\neg p(b), p(b))\}$. Toutefois, pour que la validité de cette instance implique la validité de la formule de départ, il faut prendre garde aux instances choisies. En effet, habituellement, on utilise les fonctions de Skolem pour représenter une sorte de dépendance entre les variables. Ainsi, la version skolémisée de la formule

serait $\neg p(f(u)) \vee p(u)$ avec u, un paramètre, et f, une fonction de Skolem. L'instance que nous avons choisi pourrait alors s'écrire :

$$\neg p(f(u)) \vee p(u) \vee p(g(f(u))) \vee p(f(u))$$

en prenant $f(u) = b$, $g(f(u)) = c$ et $u = a$. Notre instance tautologique respecte donc bien la notion de skolémisation. Cette notion est définie dans les preuves par expansion, grâce à une relation de dépendance $<_Q$ sur les termes instanciant les variables existentielles. Cette relation peut être vue comme la condition de fraîcheur des variables utilisée en déduction naturelle et en calcul des séquents. Par ailleurs, lorsque l'on prouve des formules skolémisées, il n'y a plus d'alternance de quantificateurs, et donc plus de condition de fraîcheur des variables. Cette relation peut alors se retrouver comme une relation de sous-terme. La relation $<_Q$ n'est en fait que l'adaptation aux arbres d'expansion de notions classiques des formalismes courants. Donnons à présent la preuve par expansion de cette formule pour la décrire de façon intuitive.



Cet arbre est un arbre d'expansion. Sa structure est celle de la formule de départ. X, Y et Z sont des paramètres. Comme nous l'avons dit, Y est dépendant de X. Ainsi, si nous avions utilisé X à la place de Z, cette preuve ne serait pas correcte, puisque X dépendrait de Y. Du point de vue de la preuve par expansion ceci se traduit par le fait que la relation $<_Q$ sur les expansions n'est pas cyclique. L'expansion, sous forme matricielle peut être représentée par :

$$\neg p(Y) \vee p(X) \vee p(Z) \vee p(Y)$$

Il n'y a qu'une seule clause complète associée à cet arbre. Il s'agit de

$$\exists y \, \forall x \, (p(x) \Rightarrow p(y)) \vee (\neg p(Y) \vee p(X)) \vee (p(Z) \vee p(Y))$$

Par conséquent la conjonction des clauses complètes se réduit à cette clause qui est couverte par la connexion $(\neg p(Y), p(Y))$ matérialisée par des pointillés sur la figure. On observe que pour le calcul des prédicats, la formule est répétée dans la clause complète. Ceci est simplement dû au fait que les connexions peuvent lier directement deux nœuds de l'arbre.

Le seul inconvénient sérieux des preuves par expansion est qu'elles ne s'expriment pas sous une forme très lisible pour l'homme. Mais on peut objecter à cela que les travaux de Miller [Mil87] et de Pfenning [Pfe87] ont fourni des algorithmes de transformation des preuves par expansion en preuves du type déduction naturelle, et réciproquement. Ces transformations sont d'ailleurs

implantées avec les preuves par expansions dans le prouveur TPS [ACM82] développé et utilisé à l'université de Carnegie Mellon.

Andrews est, avec Bibel [Bib81], l'un des précurseurs de l'utilisation des connexions pour la déduction automatique [And86]. Ces travaux ont inspirés Miller pour proposer la première version des preuves par expansion [Mil87]. Cette représentation a été reprise et légèrement modifiée par Pfenning [Pfe87]. La définition que nous en donnons ici est proche de celle formulée par Pfenning restreinte au premier ordre. Passons à la définition formelle de cette représentation des preuves.

Définition 8 Nous définissons simultanément (figure I.3)

- l'arbres d'expansion (noté Q),
- et la formule qu'il représente, dite *formule superficielle*, notée Q^s,

X et Y représentent des paramètres.

□

Définition 9 Nous définissons à présent l'ensemble des clauses complètes d'un arbre d'expansion (noté $cc(Q)$), ainsi que la conjonction de toutes ces clauses (notée $\wedge\!\!cc(Q)$).

▷ Si une formule A est un arbre d'expansion, alors A est sa seule clause complète et $\wedge\!\!cc(A) = A$.

▷ Soient $Q_1, \ldots, Q_n$ des arbres d'expansion. Pour Q, tel que

$$Q = \begin{array}{ccccc} & & \wedge & & \\ & \swarrow & & \searrow & \\ Q_1 & & \cdots & & Q_n \end{array}$$

on a
$Q^s \vee c \in cc(Q)$ pour toute clause complète c d'un Q_i,
et $\wedge\!\!cc(Q) = Q^s \vee (\wedge_{i=1}^n (\wedge\!\!cc(Q_i)))$

▷ Soient $Q_1, \ldots, Q_n$ des arbres d'expansion. Pour Q, tel que

$$Q = \begin{array}{ccccc} & & \vee & & \\ & \swarrow & & \searrow & \\ Q_1 & & \cdots & & Q_n \end{array}$$

on a
$Q^s \vee c_1 \vee \cdots \vee c_n \in cc(Q)$ pour toutes
les clauses complètes c_i de Q_i,
et $\wedge\!\!cc(Q) = Q^s \vee_{i=1}^n (\wedge\!\!cc(Q_i))$

▷ Soient $Q_1, \ldots, Q_n$ des arbres d'expansion tels que $Q_i^s = A(t_i)$. Pour Q, tel que

$$Q = \begin{array}{ccccc} & & \exists x A(x) & & \\ & \overset{t_1}{\swarrow} & & \overset{t_n}{\searrow} & \\ Q_1 & & \cdots & & Q_n \end{array}$$

on a
$\exists x A(x) \vee c_1 \vee \cdots \vee c_n \in cc(Q)$ pour toutes
les clauses complètes c_i de Q_i,
et $\wedge\!\!cc(Q) = \exists x A(x) \vee_{i=1}^n (\wedge\!\!cc(Q_i))$

▷ Soit Q_0 un arbre d'expansion tel que $Q_0^s = A(a)$. Pour Q, tel que

$$Q = \begin{array}{c} \forall x A(x) \\ \Big| \; X \\ Q_0 \end{array}$$

on a
$\forall x A(x) \vee cc(Q_0) \in cc(Q)$ pour toute clause complète de Q_0,
et $\wedge\!\!cc(Q) = \exists x A(x) \vee (\wedge\!\!cc(Q_0))$

Arbres d'expansion

▷ Une formule A est un arbre d'expansion et $Q^s = A$.

▷ Soient $Q_1, \ldots, Q_n$ des arbres d'expansion. Q est un arbre d'expansion si

$$Q = \begin{array}{ccccc} & & \wedge & & \\ & \swarrow & & \searrow & \\ Q_1 & & \cdots & & Q_n \end{array} \qquad \text{avec } Q^s = \wedge_{i=1}^n Q_i^s$$

▷ Soient $Q_1, \ldots, Q_n$ des arbres d'expansion. Q est un arbre d'expansion si

$$Q = \begin{array}{ccccc} & & \vee & & \\ & \swarrow & & \searrow & \\ Q_1 & & \cdots & & Q_n \end{array} \qquad \text{avec } Q^s = \vee_{i=1}^n Q_i^s$$

▷ Soient $Q_1, \ldots, Q_n$ des arbres d'expansion tels que $Q_i^s = A(t_i)$. Q est un arbre d'expansion si

$$Q = \begin{array}{ccccc} & & \exists x A(x) & & \\ & \overset{t_1}{\swarrow} & & \overset{t_n}{\searrow} & \\ Q_1 & & \cdots & & Q_n \end{array} \qquad \text{avec } Q^s = \exists x A(x)$$

Q est alors un *nœud d'expansion*; x est une *variable expansée*; les t_i sont des *termes d'expansion.*

▷ Soit Q_0 un arbre d'expansion tel que $Q_0^s = A(a)$. Q est un arbre d'expansion si

$$Q = \begin{array}{c} \forall x A(x) \\ X \big| \\ Q_0 \end{array} \qquad \text{avec } Q^s = \forall x A(x)$$

Q est alors un *nœud de sélection*; x est une *variable de sélection*; X est un paramètre (variable libre) sélectionné. Dans un arbre d'expansion, un paramètre ne peut être sélectionnée qu'une seule fois.

Figure I.3: Arbres d'expansion

□

Définition 10 Une relation symétrique $\mathcal{M}$ sur les formules superficielles associées aux nœuds est un *ensemble de connexions* si c'est un ensemble de couples non-ordonnés (l^s, k^s) tels que $l^s = \neg k^s$ pour chacun de ces couples. Dans la suite, on pourra dire qu'un nœud appartient à l'ensemble des connexions, alors qu'en toute rigueur, il s'agit de la formule superficielle qui lui est associée. De la même façon, nous parlerons d'une *connexion*, plutôt que d'un ensemble de

connexions. □

Les connexions sont habituellement notées $\mathcal{M}$ pour faire référence au "*mating*" de Andrews. Une connexion est une composante d'une preuve. Deux connexions différentes donnent deux preuves distinctes.

Définition 11 Une connexion $\mathcal{M}$ *couvre* une clause complète c, s'il existe un couple $(l,k) \in \mathcal{M}$ tel que $c = \cdots \vee l \vee \cdots \vee k \vee \cdots$. Une connexion $\mathcal{M}$ est une *couverture* pour (ou couvre) un arbre d'expansion Q si $\mathcal{M}$ couvre toutes les clauses complètes c de Q. Autrement dit, $\mathcal{M}$ couvre un arbre d'expansion Q si elle couvre chaque élément de la conjonction $\wedge\mathrm{cc}(Q)$. Dans ce cas, $\wedge\mathrm{cc}(Q)$ est une tautologie. Ce qui implique – par la correction des preuves par expansion [Pfe87] – que la formule Q^s représentée par Q est valide. □

Définition 12 Une connexion $\mathcal{M}_{min}$ couvrant une conjonction de clauses complètes $\wedge\mathrm{cc}(Q)$, est dite *minimale* si pour tout couple $(l,k) \in \mathcal{M}_{min}$, $\mathcal{M}_{min} - (l,k)$ ne couvre plus $\wedge\mathrm{cc}(Q)$. Notons que si une couverture existe, il en existe une minimale; mais elle n'est pas nécessairement unique. □

Si une connexion est minimale, il peut toutefois en exister une autre contenant moins d'éléments. Une connexion est minimale, simplement si aucun de ses couples n'est inutile.

Définition 13 Soit Q un arbre d'expansion. La relation $<_Q^0$ sur les occurrences des termes expansés est définie par : $t <_Q^0 s$ s'il existe une variable libre dans s, qui est sélectionnée sous t. La *relation de dépendance* $<_Q$ est la fermeture transitive de $<_Q^0$. □

Cette relation de dépendance joue le rôle de la skolémisation, mais nous aurons l'occasion d'en reparler. La définition des preuves par expansion contient une contrainte sur cette précédence qui assure la correction de la preuve sous cette forme analytique. Nous pouvons à présent donner la définition des preuves par expansion. Rappelons qu'elles ont été définies pour un ordre quelconque, mais que nous nous restreignons au premier ordre. L'extension à l'ordre supérieur constitue un objectif de travail futur.

Définition 14 Preuves par expansion

Un couple $(Q,\mathcal{M})$ est une *preuve par expansion* d'une formule A si,

- $Q^s = A$ (où A est une formule d'ordre un),
- aucune variable sélectionnée n'est libre dans Q^s,
- $<_Q$ est acyclique,
- et $\mathcal{M}$ est une connexion qui couvre $\wedge\mathrm{cc}(Q)$.

□

La première condition affirme que la formule représentée par l'arbre d'expansion est bien celle qui doit être prouvée; la seconde stipule que l'on ne peut pas utiliser de variable libre de la formule à prouver comme paramètre de la preuve; la troisième condition joue le rôle de la

skolémisation; la dernière signifie que la connexion $\mathcal{M}$ doit vérifier que la conjonction des clauses complètes est une tautologie. Nous noterons $(Q, \mathcal{M}) \vdash A$ (ou simplement $\vdash A$), le fait que $(Q\mathcal{M})$ est une preuve par expansion de la formule A. La représentation par les preuves par expansion est correcte et complète ([Pfe87]), nous pourrons donc utiliser l'équivalence $\vdash A \Leftrightarrow \models A$. Notons que la recherche d'une connexion couvrant un arbre d'expansion consiste à comparer les nœuds de cet arbre. Cette comparaison s'effectue modulo le renommage des variables liées. Or, la comparaison de deux formules est décidable. Cette recherche est, par conséquent effectuée en un temps fini et son résultat est une connexion couvrant l'arbre, ou l'assurance qu'il n'existe pas de telle connexion. Le fait que les preuves par expansion utilisent la forme normale négative des formules nuit à la lisibilité. Aussi, dans nos exemples, nous utilisons le connecteur $\Rightarrow$. En effet, $A \Rightarrow B$ est équivalent à $\neg A \vee B$. Il suffit alors de considérer les nœuds marqués par ce connecteur comme des nœuds de disjonction, et de repérer la polarité de chaque nœud . Les nœuds situés à des occurrences négatives devront être pris pour leur négation.

Définition 15 Un nœud Q_0 d'un arbre Q figure à une *occurrence positive* s'il existe un nombre pair de négation sur le chemin menant de la racine de Q jusqu'à Q_0. Inversement, un nœud Q_0 d'un arbre Q figure à une *occurrence négative* s'il existe un nombre impair de négation sur le chemin menant de la racine de Q jusqu'à Q_0. □

Ceci ne perturbe pas les preuves par expansion, parce que la structure de l'arbre est préservée. Par contre, il n'est pas possible d'utiliser l'équivalence $\Leftrightarrow$ puisqu'il faudrait l'interpréter comme une conjonction de deux implications. Le structure de l'arbre serait alors modifiée. Néanmoins, avec l'implication, des nœuds étiquetés par le quantificateur $\forall$ peuvent être en réalité des nœuds de sélection, et réciproquement, des nœuds étiquetés par le quantificateur $\exists$ peuvent être des nœuds de sélection. Cela demande parfois une certaine gymnastique, mais est préférable à l'utilisation systématique des FNN. Nous développons maintenant un exemple complet, afin d'éclaircir les notions et définitions introduites par cette représentation. Pour cet exemple, le signe des occurrences est indiqué dans l'arbre d'expansion.

Exemple 16 Considérons la formule suivante inspirée de [BdlTK92].

$$A = \forall xy\ (q(x,y) \Rightarrow \forall z\ (p(x,z) \Rightarrow p(y,z))) \wedge q(a,b) \wedge (p(a,a) \vee p(b,b))$$
$$\Rightarrow \exists x\ p(b,x)$$

On voit ici, de façon évidente, la similitude entre la formule A et l'arbre d'expansion qui lui est associé dans la preuve. Pour aider la lecture, nous avons étiqueté les branches par le nombre d'occurrences de la négation – modulo deux – au dessus de la branche en question. On peut ainsi remarquer que tous les quantificateurs universels se trouvent sous un nombre impair de négations. Ce sont donc des nœuds d'expansion. La connexion est la relation simulée par les pointillés joignant des nœuds de l'arbre – dans le cas présent, cela ne concerne que des feuilles. Donnons maintenant un exemple de clause complète associée à cette preuve:

$$c = A \vee \exists x \exists y\ (q(x,y) \wedge \exists z\ (p(x,z) \wedge \neg p(y,z)) \vee q(a,b) \vee \neg q(a,b)$$
$$\vee \underline{\neg p(b,b)} \vee \exists x\ q(b,x) \vee p(b,a) \vee \underline{p(b,b)})$$

■

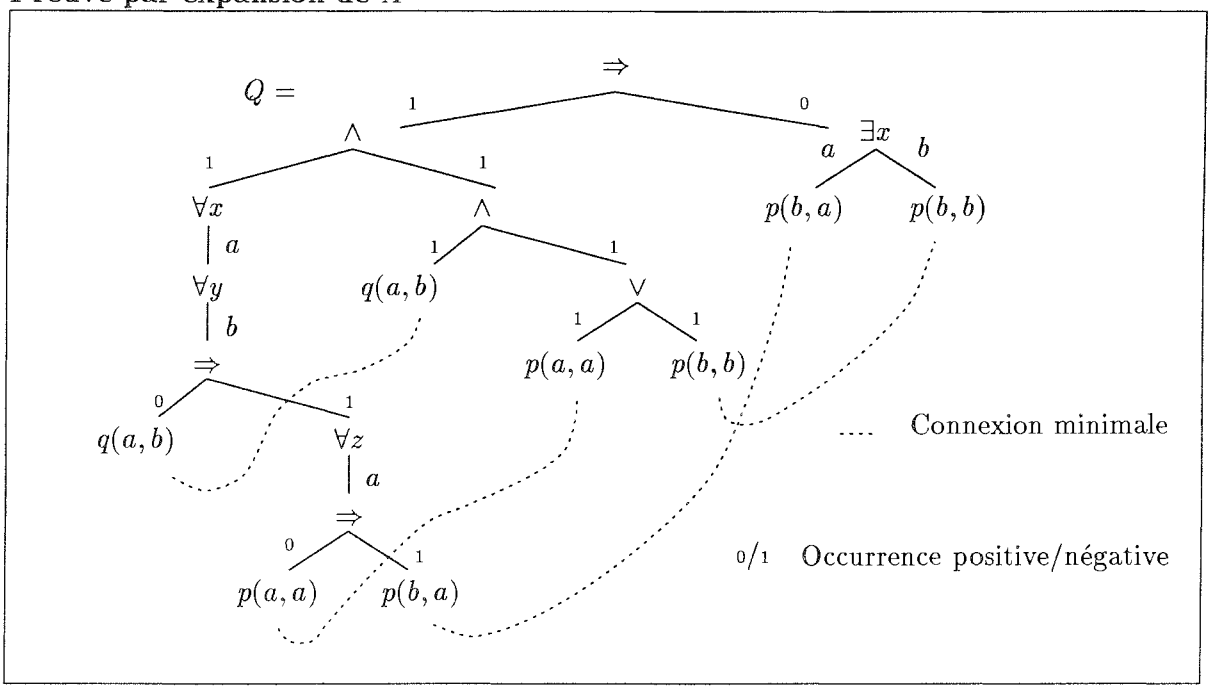


Figure I.4: Preuve par expansion

Cette clause contient une occurrence positive du littéral $p(b,b)$, ainsi qu'une occurrence négative. C'est donc une tautologie, et ces deux occurrences sont bien liées par la connexion. Ainsi, on peut écrire la forme expansée de A qui est contenue dans l'arbre, sous forme d'une conjonction de telles clauses. Si chacune de ces clauses contient au moins un couple de la connexion, c'est une tautologie. On voit donc bien ici, l'équivalence entre le fait qu'il existe une connexion qui couvre la conjonction des clauses complètes et le fait que la forme expansée de A est tautologique. Il faut toutefois remarquer que c n'est pas exactement une clause complète associée à cet arbre, mais seulement une partie de la disjonction. Elle a été simplifiée par souci de clarté. En particulier, telle qu'elle est présentée ici, elle expanse les trois variables existentielles en une seule étape, alors qu'en réalité, ces trois étapes doivent être distinctes. Notons également au passage que la connexion donnée est minimale.

Maintenant que nous avons introduit la représentation des preuves que nous allons manipuler et la relation $\preceq$ qui va être notre première définition de l'analogie, nous pouvons passer au processus d'analogie proprement dit.

I.5 Preuve par analogie via la relation $\preceq$

Dans cette section, nous montrons comment, dans un cas simple, l'analogie peut être utilisée pour prouver automatiquement des formules proches de celles déjà prouvées. Bien que la similitude entre les formules soit relativement simple, l'intégration d'un tel processus dans un démonstrateur doit permettre d'éviter un nombre considérable de preuves redondantes les unes par rapport aux autres. Nous donnons donc ici les outils techniques nécessaires, puis les algorithmes

de transformation. Enfin, nous donnons les propriétés théoriques de ces transformations.

I.5.1 Preuves par expansion réduites

Nous avons défini les preuves par expansion, et nous allons définir deux transformations à leur appliquer. D'une part, l'élagage de preuves afin de les simplifier, et de simplifier la formule prouvée par la même occasion. Pour cette opération, nous avons besoin de définir les formules élaguées – ce qui est très proche de la relation $\preceq$ – et les preuves par expansion réduites. Ces preuves sont, approximativement, des preuves par expansion auxquelles nous avons intégré la relation $\preceq$. Mais les preuves par expansion réduites doivent être retransformées en preuves par expansion classiques. C'est le but de la seconde manipulation. Celle-ci permet, non seulement de transformer une preuve par expansion réduite d'une formule A en une preuve par expansion de cette même formule, mais aussi de la transformer en preuve par expansion de toute formule B vérifiant $A \preceq B$.

Comme nous l'avons dit auparavant, d'une formule choisie comme référence pour faire de l'analogie, on peut extraire une formule plus simple et sa preuve, qui contienne toute l'information essentielle. C'est cette opération – l'élagage – qui introduit la nécessité des preuves réduites.

A Formules élaguées

La transformation de formules qu'est l'élagage est en réalité une reformulation de la relation $\preceq$. Elle permet d'obtenir toutes les formules plus petites, comme le montre le lemme 18.

Définition 17 Soit A une formule quelconque. B est une *formule élaguée* de A, notée $El(A)$, si et seulement si B peut être dérivée par une séquence d'applications des règles suivantes :

a- $El(A) \rightarrow A$

b- $El(A_1 \vee A_2) \rightarrow El(A_1)$

c- $El(A_1 \vee A_2) \rightarrow El(A_2)$

d- $El(A_1 \vee A_2) \rightarrow El(A_1) \vee El(A_2)$

e- $El(A_1 \wedge A_2) \rightarrow El(A_1) \wedge El(A_2)$

f- $El(\exists x\ A) \rightarrow \exists x\ (El(A))$

g- $El(\forall x\ A) \rightarrow \forall x\ (El(A))$

□

Chacune de ces règles fait décroître le multi-ensemble des tailles des termes appliqués à El. Toute séquence d'applications de ces règles est donc finie.

B Relation entre formules élaguées et $\preceq$

Le lemme suivant met en évidence le fait qu'en élagant une formule A, on obtient une formule B qui vérifie $B \preceq A$, et que, inversement, si $B \preceq A$, il existe une façon d'élaguer A pour obtenir B.

Lemme 18 *Pour toutes formules A et B,*

$$B = El(A) \quad \Leftrightarrow \quad B \preceq A.$$

Preuve
$\Leftarrow$ Ce cas est simple, puisque chaque règle de El correspond à une règle de la définition de $\preceq$.
$\Rightarrow$ par induction structurelle sur A :

Cas de base : A est un littéral. Par conséquent, $A \preceq A$ et $A = El(A)$.
Cas inductifs : nous passons en revue tous les cas où $B \preceq A$.

- $B = A$: $B \preceq A$ est vérifié par application de la règle **a-**
- $A = A_1 \vee A_2$ avec $B \preceq A_1$: est vérifié par application de la règle **b-** puis, par application de l'hypothèse d'induction.
- $A = A_1 \vee A_2$ avec $B \preceq A_2$: est vérifié par application de la règle **c-** puis, par application de l'hypothèse d'induction.
- $A = A_1 \wedge A_2$ avec $B \preceq A_1$ et $B \preceq A_2$: est vérifié par application de la règle **e-**, puis, par hypothèse d'induction.
- $A = A_1 \vee A_2$ et $B = B_1 \vee B_2$ avec $B_1 \preceq A_1$ et $B_2 \preceq A_2$: est vérifié par application de la règle **d-** puis par induction.
- $A = A_1 \wedge A_2$ et $B = B_1 \wedge B_2$ avec $B_1 \preceq A_1$ et $B_2 \preceq A_2$: est vérifié par application de la règle **e-** puis par induction.
- $A = \exists x\ A'$ et $B = \exists x\ B'$ avec $B' \preceq A'$: est vérifié par application de la règle **f-** puis par induction.
- $A = \forall x\ A'$ et $B = \forall x\ B'$ avec $B' \preceq A'$: est vérifié par application de la règle **g-** puis par induction.

□

Pour les manipulations qui suivent, nous allons avoir besoin de la notion de formule commune. Il s'agit simplement de définir une formule C telle que l'on ait à la fois $A \preceq C$ et $B \preceq C$. C est alors à la fois plus générale que A et plus générale que B. Mais ce genre de formule n'est pas unique, et d'autre part, nous voudrions en garder une, aussi simple que possible. C'est pourquoi nous définissons la notion de formule commune minimale respectivement à C, qui correspond à la formule la plus élaguée, qui soit plus grande que A et B, et plus petite que C par la relation $\preceq$. Voyons les définitions formelles.

Définition 19 Soient A, B et C trois formules vérifiant $A \preceq C$ et $B \preceq C$. C' est une formule commune à A et B respectivement à C si les trois conditions suivantes sont vérifiées simultanément

- $A \preceq C'$
- $B \preceq C'$

- $C' \preceq C$

De plus, C' est une *formule commune minimale* (fcm) à A et B respectivement à C s'il n'est plus possible de l'élaguer en conservant ces trois propriétés. Notons qu'une telle formule existe toujours, puisque C est une formule commune à A et B respectivement à C. □

Il n'existe donc pas de formule élaguée de C plus petite que C' et à la fois plus grande que A et que B. Par conséquent, dire que C' est la plus petite formule commune à A et B respectivement à C est équivalent à dire que pour toute formule D telle que $A \preceq D$, $B \preceq D$ et $D \preceq C$, alors $C' \preceq D$.

Exemple 20 Prenons par exemple les formules

$$A = \forall xy\ (p(x,y) \wedge q(y)) \Rightarrow \exists z\ p(a,z)$$

$$\text{et } B = \forall xy\ p(x,y) \Rightarrow \exists z\ (p(a,z) \vee p(z,a))$$

Alors, $C' = \forall xy\ (p(x,y) \wedge q(y)) \Rightarrow \exists z\ (p(a,z) \vee p(z,a))$ est une fcm pour A et B respectivement à C, pour toute formule C telle que $C' \preceq C$. Notons au passage que lorsque $A \preceq B$, B est une fcm respectivement à toute formule C telle que $B \preceq C$. ■

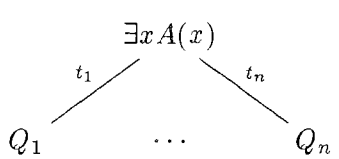
La notation $A = fcm(\overline{B_n})$ respectivement à C signifie que A est une fcm pour chacun des couples $(B_i, B_j)_{i,j \in \{1,..,n\}}$ respectivement à C. Nous pouvons maintenant donner la définition des preuves par expansion réduites.

C Définition des preuves par expansion réduites

La définition des arbres d'expansion réduits est équivalente à celle des arbres d'expansion, excepté pour les nœuds d'expansion et de sélection. Nous ne donnons ici que ces deux parties de la définition, le reste étant inchangé. Il suffit d'y substituer "arbre d'expansion" par "arbre d'expansion réduit".

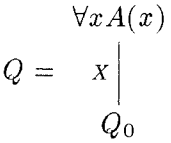
Définition 21 Arbres d'expansion réduits

▷ Soient $Q_1, \ldots, Q_n$ des arbres d'expansion réduits tels que $\boxed{Q_i^s \preceq A(t_i)}$. Q est un arbre d'expansion réduit si

$Q =$ 

avec $Q^s = \exists x A(x)$
$\exists x A(x) \vee cc(Q_i) \in cc(Q)$ pour toutes les clauses complètes de Q_i.
Alors, $\wedge cc(Q) = \exists x A(x) \vee_{i=1}^{n} (\wedge cc(Q_i))$

▷ Soit Q_0 un arbre d'expansion réduit tel que $\boxed{Q_0^s \preceq A(X)}$. Q est un arbre d'expansion réduit si

$Q =$ 

avec $Q^s = \forall x A(x)$
$\forall x A(x) \vee cc(Q_0) \in cc(Q)$ pour toutes les clauses complètes de Q_0.
Alors, $\wedge cc(Q) = \exists x A(x) \vee (\wedge cc(Q_0))$

□

La différence entre les arbres d'expansion et les arbres d'expansion réduits, est que nous avons remplacé la condition d'égalité syntaxique entre la formule superficielle d'un nœud et de ses fils par la relation $\preceq$. Cela correspond à ce que l'on peut obtenir après élagage d'un arbre d'expansion. Cette opération d'élagage d'arbres est définie plus loin. Nous pouvons maintenant donner la définition des preuves par expansion réduites.

Définition 22 Preuves par expansion réduites
Un couple $(Q, \mathcal{M})$ est une **preuve par expansion réduite** d'une formule A si

- Q est un arbre d'expansion réduit vérifiant $Q^s = A$,
- aucune variable sélectionnée n'est libre dans Q^s,
- $<_Q$ est acyclique,
- et $\mathcal{M}$ est une connexion qui couvre $\wedge\mathrm{cc}(Q)$.

□

Nous écrirons $(Q, \mathcal{M}) \vdash_R A$ lorsque $(Q, \mathcal{M})$ est une preuve par expansion réduite de A, ou $\vdash_R A$ pour signifier qu'il en existe une. Comme nous avons décrit précédemment l'élagage des formules, nous allons voir maintenant comment élaguer des preuves par expansion – pour obtenir des preuves par expansion réduites. Cela correspond concrètement à une simplification de la preuve et de la formule prouvée, puisque nous verrons que nous obtenons une preuve réduite d'une formule élaguée de la formule de départ.

I.5.2 Elagage des preuves (E)

Il s'agit d'utiliser l'information contenue dans la connexion liée à la preuve, afin de l'affiner. Nous allons ainsi débarrasser l'arbre d'expansion de ses branches inutiles du point de vue de la preuve. L'arbre d'expansion réduit obtenu correspondra à une formule plus petite que la formule de départ selon la relation $\preceq$. Mais nous donnons par la suite un algorithme de transformation d'une preuve par expansion réduite en une preuve par expansion. C'est en cela que les preuves réduites ne sont en fait qu'un intermédiaire dans la méthode.

Pour transformer les preuves par expansion en déduction naturelle, Pfenning [Pfe87] introduit des notions de "nœuds non-essentiels" et d'"effacement". Ce que nous utilisons dans ce qui suit est un peu similaire. Par contre, pour ses transformations, Pfenning travaille toujours à la racine de l'arbre d'expansion. Ainsi, définit-il comme "inaccessible" tout nœud se trouvant sous un nœud de sélection ou d'expansion. Or, dans notre cas, nous avons besoin de parcourir tout l'arbre. Nous permettons donc de travailler sur ces nœuds dit inaccessibles, mais en n'appliquant que des transformations simples. L'algorithme de transformation des preuves réduites en preuves par expansion montrera que nous n'avons pas altéré la correction des preuves. En premier lieu, définissons les nœuds d'un arbre d'expansion qui ne sont pas effectivement utiles à la preuve.

Définition 23 Un nœud Q d'un arbre d'expansion est *superflu* si

- ni lui ni aucun de ses descendants n'appartient à la connexion,

- pour Q^s, la formule associée au nœud Q, Q n'est pas un descendant d'un nœud dont la formule associée est $A(t)$ où t est un terme d'expansion, et tel que Q^s est une sous-formule de t.

□

En clair, la seconde condition de la définition signifie que l'on n'autorise pas l'élagage de morceaux de formules qui ont été introduits par un terme d'expansion. On ne peut élaguer que des morceaux de la formule de départ.

L'algorithme d'élagage de preuves se déroule en deux étapes (figure I.5).

Intuitivement, après la première étape, nous avons un arbre épuré de ses branches inutiles. Mais la formule superficielle associée à l'arbre de départ peut ne plus correspondre à ce nouvel arbre. En effet, si sous un nœud d'expansion, nous avons pratiqué l'élagage, nous n'avons plus des formules superficielles égales à $A(t_i)$ où t_i est le terme d'expansion, mais nous avons des formules superficielles $B_i(t_i)$. Or, l'un de nos buts est de simplifier au mieux la formule superficielle associée à l'arbre. Par conséquent, plutôt que de garder $\exists x\ A(x)$ comme père de tous ces nœuds nous prenons la plus petite formule commune, notée $B(x)$, qui remplacera $A(x)$, et qui est calculée par $B(x) = fmc(\overline{B_i(x)})$. Ce qui garantit que nous conservons la propriété $B_i(x) \preceq B(x)$, mais aussi $B(x) \preceq A(x)$. Par ailleurs, le lemme 7 nous garantit que l'on peut faire abstraction des instances t_i pour calculer un tel $B(x)$. Celui-ci existe donc toujours, puisque, dans le pire des cas, $B(x)$ sera simplement $A(x)$.

Avant de donner une méthode de reconstruction d'une preuve par expansion, observons les effets de l'élagage sur une preuve simple.

Exemple 24 La figure I.6 contient deux preuves. La première est une preuve par expansion de la formule $\forall x(A(x) \wedge B(x)) \Rightarrow \exists x A(x)$. La preuve de droite est le résultat de l'élagage de la preuve de gauche. La formule superficielle correspondante est $\forall x A(x) \Rightarrow \exists x A(x)$. Nous avons donc bien épuré cette preuve pour n'en garder que l'information essentielle, et simplifié par là même la formule prouvée.

■

Nous savons maintenant comment extraire l'information essentielle d'une preuve par expansion, grâce à la connexion – c'est cette connexion qui permet de déterminer les nœuds superflus – ce qui renforce l'idée selon laquelle la connexion contient la raison pour laquelle la formule est valide. On voit par exemple que le processus d'élagage d'une preuve supprimera toutes les expansions inutiles produites durant la construction de la preuve originelle. Un problème subsiste, puisque nous n'obtenons pas des preuves par expansion, mais des preuves réduites. Nous allons donc nous intéresser à la transformation des preuves réduites en preuves par expansion.

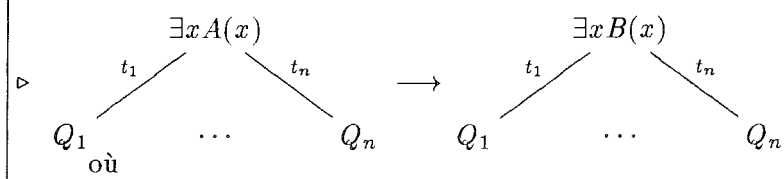
I.5.3 Transformation : preuve par expansion réduite → preuve par expansion (TR)

Maintenant que nous avons épuré la preuve de ce qui lui était inutile, nous allons reconstituer une preuve par expansion classique. Pour cela, il suffit de reconstituer les formules superficielles de chaque nœud de l'arbre d'expansion, afin de retrouver la correspondance entre la formule et

$E(Q)$: élagage des preuves par expansion

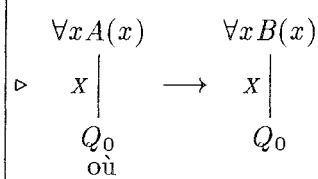
1- effacement de tous les nœuds superflus dans Q.

2- application des règles suivantes de bas en haut de l'arbre.



où

- $Q_i^s = B_i(t_i) \preceq A(t_i)$,
- $\exists x A(x) \notin \mathcal{M}$,
- et $B(x) = fcm(\overline{B_i(x)})_{i=1,\ldots,n}$ respectivement à $A(X)$.



où

- $B(X) = Q_0^s \preceq A(X)$,
- et $\forall x A(x) \notin \mathcal{M}$.

Figure I.5: Elagage des preuves par expansion

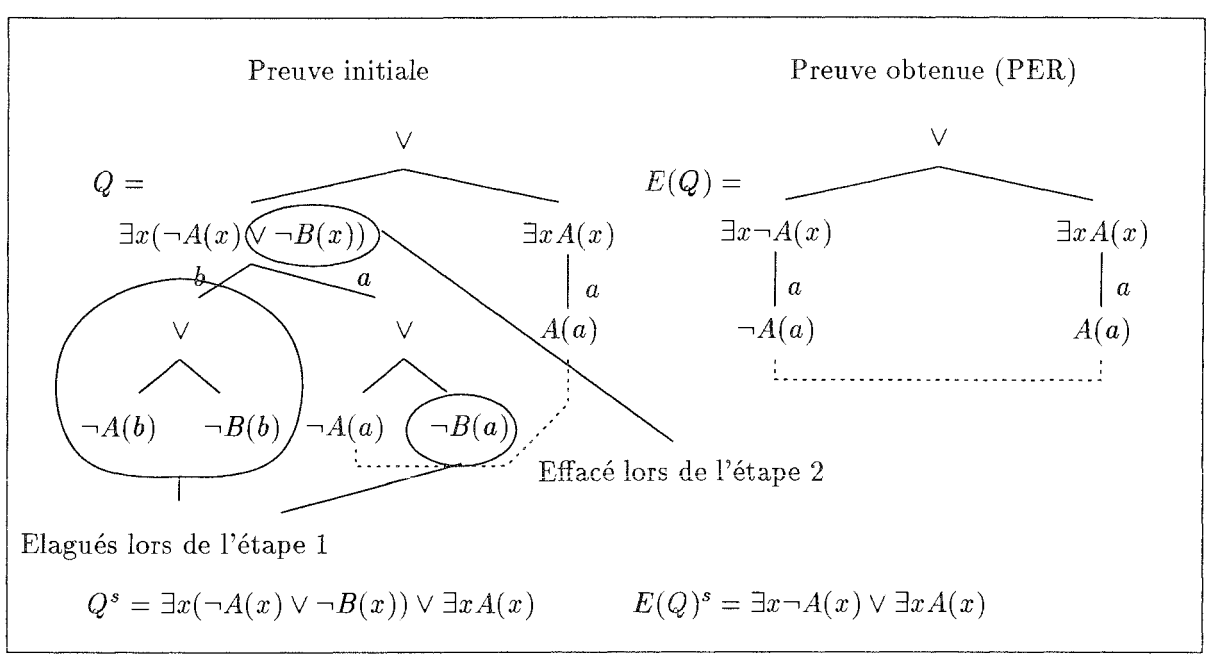


Figure I.6: Elagage de la preuve de départ

l'arbre. Mais en fait, cette transformation a un autre intérêt. Elle permet de transformer toute preuve par expansion – ou par expansion réduite – d'une formule A en preuve par expansion pour toute formule respectant la condition $A \preceq B$. La raison en est très simple. Partant du principe que le cœur de la preuve se trouve dans la connexion, il nous suffit de modifier la structure de l'arbre afin qu'il reflète B au lieu de A. La relation $A \preceq B$ nous garantit que si la connexion $\mathcal{M}$ couvre l'arbre représentant A, elle couvrira également l'arbre représentant B. Mais en particulier, bien sûr, nous pouvons prendre $A = B$. Ce qui permet de passer d'une preuve réduite à une preuve par expansion pour A.

Définition 25 La transformation TR prend donc trois arguments :
$TR(Q, A, B)$ avec

- Q est un arbre d'expansion réduit (ou arbre d'expansion) tel que $Q^s = A$,
- A est une formule
- B est une formule vérifiant $A \preceq B$.

La transformation est définie par les règles 0 à 6. La numérotation des règles correspond à celles de la définition de la relation $\preceq$. □

Un théorème est de référence, lorsque l'élagage a été appliqué à sa preuve relativement à une connexion minimale, puis transformée à nouveau en une preuve par expansion. Lui appliquer à nouveau l'élagage, puis la transformation TR, n'aurait plus d'effet.

Définition 26 Un théorème A et sa preuve $(Q, \mathcal{M})$ sont dit *de référence* si $Q = TR(E(Q), A, A)$, lorsque l'élagage $E(Q)$ est relatif à la connexion minimale $\mathcal{M}$. □

Nous donnons maintenant un exemple d'application de la transformation TR. Il s'agit de la prolongation de l'exemple d'élagage de preuve.

Exemple 27 Nous avions dans l'exemple précédent, une preuve par expansion pour la formule $A = \forall x(A(x) \wedge B(x)) \Rightarrow \exists x A(x)$. Après élagage, nous avons obtenu une preuve réduite de la formule $B = \forall x A(x) \Rightarrow \exists x A(x)$. Or, comme nous l'avons dit, la formule obtenue est plus petite au regard de la relation $\preceq$ que la formule de départ. Nous allons maintenant appliquer la transformation TR afin d'obtenir à nouveau une preuve par expansion de A. La transformation se fait par $TR(Q, B, A)$. On obtient la figure IV.3.

Encore une fois, la connexion reste intacte. Elle constitue le cœur de la preuve, et une raison suffisante pour toute formule plus grande que B par la relation $\preceq$, d'être valide. ■

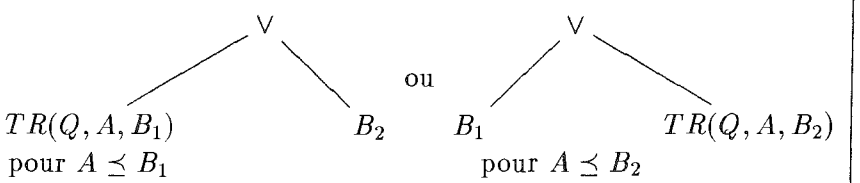
Enfin, remarquons que l'élagage d'une preuve suivi de la reconstruction d'une preuve par expansion de la formule originelle ne produit pas forcément le même arbre d'expansion. En effet, l'élagage a pu supprimer des expansions inutiles qui ne seront pas reconstruite par application de la transformation TR. Ainsi, sur l'exemple ci-dessus, l'expansion par b qui figurait dans la preuve originelle a été effacée. Ce qui signifie également qu'indépendamment de l'utilisation que nous en faisons pour l'analogie, l'élagage peut être utilisé pour simplifier les preuves par expansion, ce qui peut constituer une application en soi.

En résumé, le rôle des transformations que nous avons introduites peut être défini ainsi :

Règles de transformation : preuve par expansion réduite → preuve par expansion

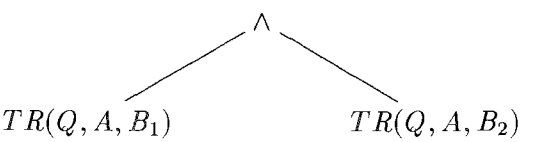
0- Q est une feuille. $TR(A, A, B) \longrightarrow B$

1- $TR(Q, A, B_1 \vee B_2) \longrightarrow$



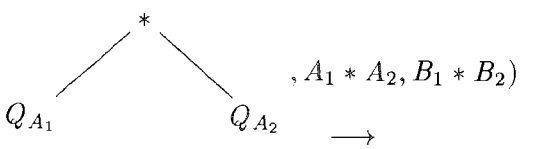
avec renommage des variables libres de B_1 (ou B_2) sélectionnées dans Q.

2- $TR(Q, A, B_1 \wedge B_2) \longrightarrow$

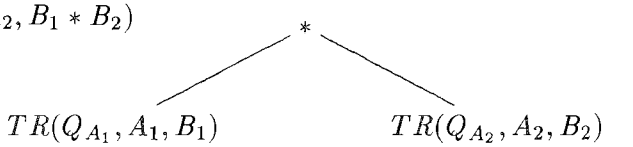


avec $A \preceq B_1$ et $A \preceq B_2$

3-& 4- $TR($

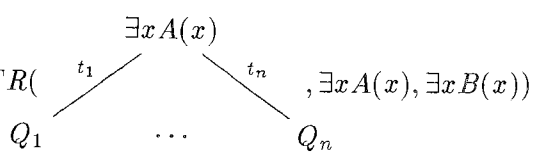


$, A_1 * A_2, B_1 * B_2) \longrightarrow$

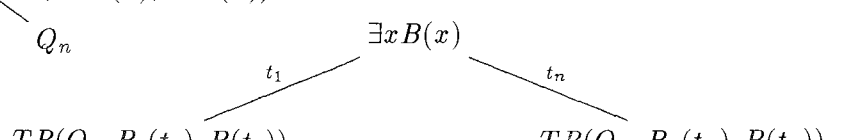


avec $Q^s_{A_1} = A_1$; $A_1 \preceq B_1$; $Q^s_{A_2} = A_2$ et $A_2 \preceq B_2$
Pour $* = \vee$ pour la règle **3-**, et $\wedge$ pour la règle **4-**.

5- $TR($

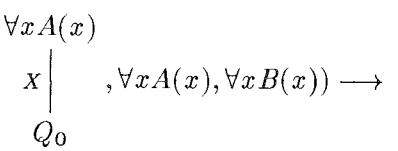


$, \exists x A(x), \exists x B(x)) \longrightarrow$

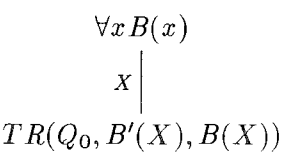


où $Q^s_i = B_i(t_i) \preceq A(t_i)$

6- $TR($



$, \forall x A(x), \forall x B(x)) \longrightarrow$



où $Q^s_0 = B'(X) \preceq A(X)$

Figure I.7: Transformation preuve par expansion réduite → preuve par expansion

- Pour toute formule A munie d'une preuve par expansion $(Q, \mathcal{M})$ où $\mathcal{M}$ est une connexion minimale, on peut calculer une formule A_0 munie d'une preuve par expansion $(Q_0, \mathcal{M})$ telles que $A_0 \preceq A$.

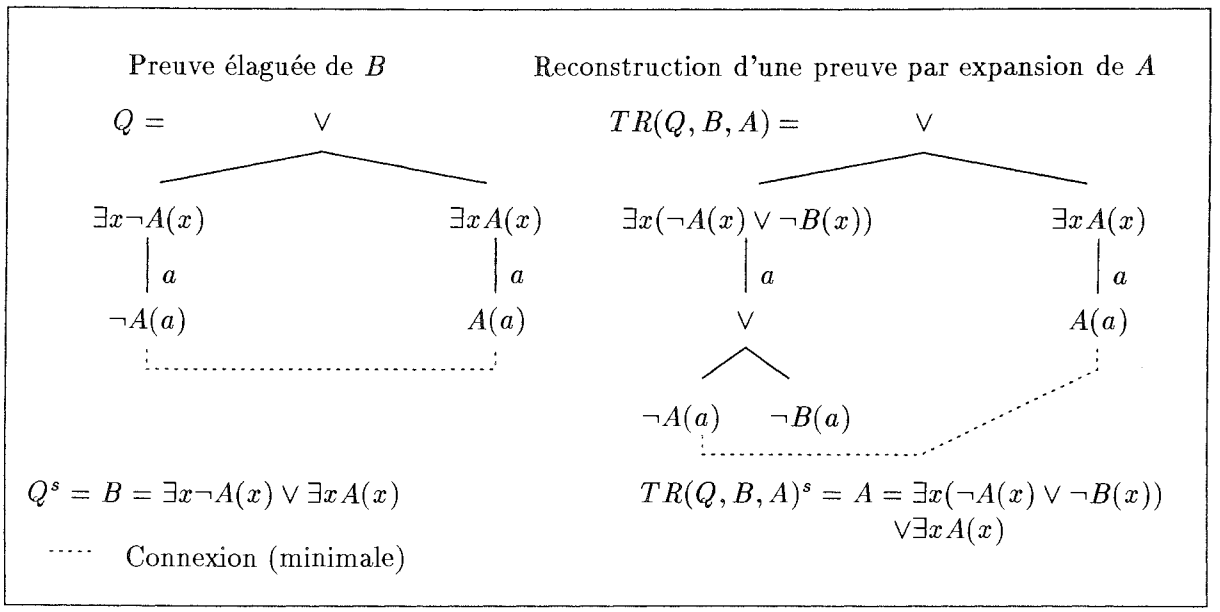


Figure I.8: Reconstruction de la preuve par expansion de A

- Pour toute formule B vérifiant $A_0 \preceq B$, on peut construire une preuve par expansion $(Q', \mathcal{M}) \vdash B$.

L'objectif de la section suivante est de prouver la correction, complétude et terminaison de ces transformations.

I.5.4 Propriétés des transformations E et TR

Nous allons montrer maintenant que les transformations présentées réalisent correctement, complètement et en un temps fini ce qui était annoncé par la figure I.2.

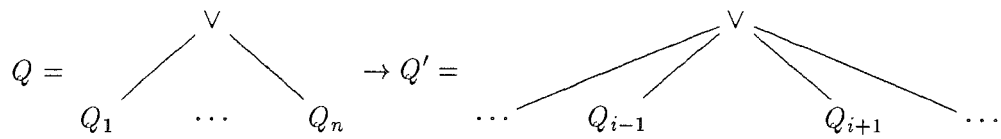
A Propriétés de l'élagage

Il va de soi que l'élagage est un processus qui termine. En effet, le nombre de nœuds superflus est fini, ainsi que le nombre de nœuds de sélection et d'expansion. Tout d'abord, nous allons montrer que l'élagage d'une preuve par expansion est une preuve par expansion réduite. Pour ce faire, nous distinguons les deux étapes de cette transformation.

Lemme 28 *Soit Q un arbre d'expansion. Si $Q' = E(Q)$, respectivement à la connexion minimale $\mathcal{M}$, alors, après la première étape de cette transformation, Q' est un arbre d'expansion réduit tel que $Q'^s \preceq Q^s$.*

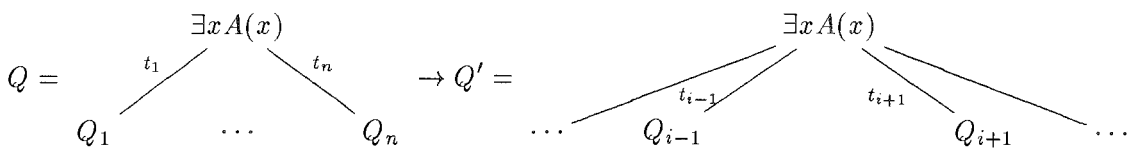
Preuve Par induction sur le nombre de branches coupées.
Cas de base : une seule branche a été coupée. On considère son père.

- Disjonction



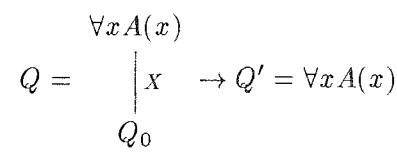
On a donc $Q'^s = Q_1^s \vee \cdots \vee Q_{i-1}^s \vee Q_{i+1}^s \vee \cdots \vee Q_n^s \preceq Q_1^s \vee \cdots \vee Q_i^s \vee \cdots \vee Q_n^s = Q^s$ et Q' qui est un arbre d'expansion réduit puisque tous les Q_j sont des arbres d'expansion.

- Quantificateur existentiel



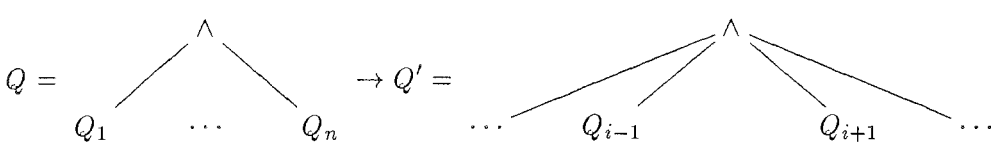
On a donc Q' qui est un arbre d'expansion réduit puisque tous les Q_j sont des arbres d'expansion. D'autre part, $Q'^s = Q^s$. Notons que si $n = i = 1$, $\exists x A(x)$ est un arbre d'expansion réduit.

- Quantificateur universel



Identique au cas précédent, avec $n = i = 1$.

- Conjonction



Ce cas ne peut jamais survenir, puisque si l'un des Q_i est superflu, soit Q est globalement superflu, soit la connexion n'est pas minimale.

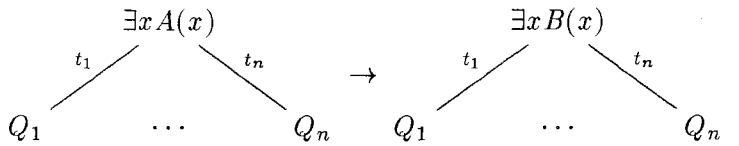
Cas inductif : on peut tenir exactement le même raisonnement, excepté que les Q_j sont des arbres d'expansion réduits au lieu d'arbres d'expansion. □

Il reste à vérifier la correction de la seconde partie de la transformation.

Lemme 29 *Soit Q un arbre d'expansion. Si $Q' = E(Q)$, respectivement à la connexion minimale $\mathcal{M}$, alors, Q' est un arbre d'expansion réduit tel que $Q'^s \preceq Q^s$.*

Preuve Il ne reste à prouver que la deuxième étape. Cela correspond à la correction des deux règles appliquées.

- Règle 1 :

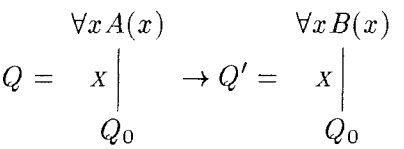


avec $Q^s = \exists x A(x)$, et $Q'^s = \exists x B(x)$. Les Q_j sont des arbres d'expansion réduits par le lemme 28. De plus, $B(x) = fcm(\overline{B_i(x)})$ respectivement à $A(x)$. D'où, par définition, par définition

$$B_j(x) \preceq B(x)$$

pour $j = 1,..n$. Et comme, par définition, $B(x) \preceq A(x)$, Q' est un arbre d'expansion réduit vérifiant $Q'^s \preceq Q^s$.

- Règle 2 :



avec $B(X) = Q_0^s$. Donc, Q' est un arbre d'expansion réduit puisque Q_0 en est un. En effet, par le lemme 28, les Q_j sont des arbres d'expansion réduits. Or, $Q_0^s \preceq A(X)$. D'où $B(X) \preceq A(X)$ et donc, $Q'^s \preceq Q^s$. Les conditions pour que Q' soit un arbre d'expansion réduit sont vérifiées.

□

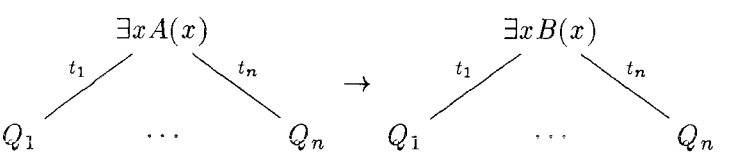
Nous savons maintenant que l'élagage d'une preuve par expansion donne une preuve réduite d'une formule plus petite par la relation $\preceq$, que la formule originelle. Il reste à montrer que la connexion qui couvrait $\wedge cc(Q)$ couvre toujours $\wedge cc(Q')$.

Lemme 30 *Soit $\mathcal{M}$ une connexion minimale, Q un arbre d'expansion et Q' tel que $Q' = E(Q)$. Si $\mathcal{M}$ couvre $\wedge cc(Q)$, alors $\mathcal{M}$ couvre $\wedge cc(Q')$.*

Preuve Si $\mathcal{M}$ couvre $\wedge cc(Q)$, le fait que $\mathcal{M}$ couvre Q' où Q' est l'élagage de Q après la première étape et respectivement à $\mathcal{M}$ est évident, puisque l'on n'a ôté que les branches n'intervenant pas dans la connexion.

Après la seconde étape,

- Règle 1 :



avec $Q_i^s = B_i(t_i) \preceq A(t_i)$, et $\exists x A(x) \notin \mathcal{M}$. Par conséquent, si $\mathcal{M}$ couvre $\wedge cc(Q) = \exists x A(x) \vee_{j=1}^n \wedge cc(Q_i)$, elle couvre aussi $\wedge cc(Q') = \exists x B(x) \vee_{j=1}^n \wedge cc(Q_i)$.

- Règle 2 :

$$\begin{array}{ccc} \forall x A(x) & & \forall x B(x) \\ \Big| X & \rightarrow & \Big| X \\ Q_0 & & Q_0 \end{array}$$

avec $Q_0^s = B(X)$ et $\forall x A(x) \notin \mathcal{M}$. Par conséquent, si $\mathcal{M}$ couvre $\wedge cc(Q) = \forall x A(x) \vee \wedge cc(Q_0)$, elle couvre aussi $\wedge cc(Q') = \forall x B(x) \vee \wedge cc(Q_0)$.

□

Lors de cette manipulation, nous n'avons pu qu'ôter des sélections ou des expansions. Par conséquent, nous n'avons pas pu rendre cyclique la relation de dépendance si elle ne l'était pas auparavant. Il en est de même pour la condition sur les variables sélectionnées libres dans Q^s. On peut donc en déduire le lemme suivant.

Lemme 31 *Soit Q un arbre d'expansion tel que la relation de dépendance $<_Q$ qui lui est associée soit acyclique, et qu'aucune variable sélectionnée n'apparaisse libre dans Q^s. Alors, $E(Q)$ satisfait la condition $<_{P(Q)}$ acyclique et il n'existe pas de variable sélectionnée dans Q' qui soit libre dans Q'^s.*

Nous avons alors le théorème suivant qui donne la correction de l'élagage, sachant que cette transformation peut s'appliquer à une quelconque preuve par expansion, pourvu qu'elle soit munie d'une connexion minimale.

Théorème 32 *$(Q, \mathcal{M}) \vdash A$ implique $(P(Q), \mathcal{M}) \vdash_R B$ avec $B = P(Q)^s \preceq A$.*

B Propriétés de la transformation TR

Là encore, la terminaison du processus est évidente puisque chaque application de règle fait descendre la transformation sur les fils, et finit par conséquent par s'arrêter puisque les arbres d'expansion sont finis. Montrons que pour toute preuve par expansion ou preuve réduite $(Q, \mathcal{M})$ d'une formule A, $(TR(Q, A, B), \mathcal{M})$ est une preuve par expansion pour B lorsque $A \preceq B$. Pour cela, commençons par montrer que l'on obtient bien un arbre d'expansion et que la formule superficielle qui lui est associée est bien B.

Lemme 33 *Soit Q un arbre d'expansion réduit (ou arbre d'expansion) tel que $Q^s = A$, et B une formule vérifiant $A \preceq B$. $Q' = TR(Q, A, B)$ est un arbre d'expansion et $Q'^s = B$.*

Preuve Par induction sur la hauteur de l'arbre Q.
Cas de base : $|Q| = 1$. Ce cas correspond à la règle 0 et on a $TR(Q, A, B) = B$ qui est un arbre d'expansion dont la formule superficielle associée est B, par définition.
Cas inductif : l'hypothèse d'induction (HI) s'énonce comme ceci : pour tout Q tel que $|Q| \leq n$, $Q' = TR(Q, A, B)$ est un arbre d'expansion vérifiant $Q'^s = B$. On regarde alors chaque cas où A et B peuvent être comparés.

1- $B = B_1 \vee B_2$. Par la règle 1, lorsque $A \preceq B_1$

$$Q' = TR(Q, A, B_1 \vee B_2) = \begin{array}{ccc} & \vee & \\ TR(Q, A, B_1) & & B_2 \end{array}$$

et, lorsque $A \preceq B_2$

$$Q'' = TR(Q, A, B_1 \vee B_2) = \quad \begin{array}{ccc} & \vee & \\ B_1 & & TR(Q, A, B_2) \end{array}$$

Par (HI), $TR(Q, A, B_1)$ est un arbre d'expansion vérifiant $TR(Q, A, B_1)^s = B_1$. D'où, $Q''^s = TR(Q, A, B_1)^s \vee B_2 = B_1 \vee B_2$.
(de la même façon pour $Q''^s = B_1 \vee TR(Q, A, B_2)^s = B_1 \vee B_2$).

2- $B = B_1 \wedge B_2$ avec $A \preceq B_1$ et $A \preceq B_2$.

$$Q' = TR(Q, A, B) = \quad \begin{array}{ccc} & \wedge & \\ TR(Q, A, B_1) & & TR(Q, A, B_2) \end{array}$$

Par (HI), $TR(Q, A, B_i)^s = B_i$ $(i = 1, 2)$ et $TR(Q, A, B_i)$ sont des arbre d'expansion. Donc, Q' est un arbre d'expansion vérifiant $Q'^s = B_1 \wedge B_2$.

3- $A = A_1 \vee A_2$ et $B = B_1 \vee B_2$ avec $A_1 \preceq B_1$ et $A_2 \preceq B_2$.

$$Q' = TR\left(\begin{array}{ccc} & \vee & \\ Q_{A_1} & & Q_{A_2} \end{array}, A, B\right) = \quad \begin{array}{ccc} & \vee & \\ TR(Q_{A_1}, A_1, B_1) & & TR(Q_{A_2}, A_2, B_2) \end{array}$$

On peut appliquer (HI) à $TR(Q_{A_1}, A_1, B_1)$ et $TR(Q_{A_2}, A_2, B_2)$. Alors, Q' est un arbre d'expansion vérifiant $Q'^s = TR(Q_{A_1}, A_1, B_1)^s \vee TR(Q_{A_2}, A_2, B_2)^s = B_1 \vee B_2$.

4- $A = A_1 \wedge A_2$ et $B = B_1 \wedge B_2$ avec $A_1 \preceq B_1$ et $A_2 \preceq B_2$.

$$Q' = TR\left(\begin{array}{ccc} & \wedge & \\ Q_{A_1} & & Q_{A_2} \end{array}, A, B\right) = \quad \begin{array}{ccc} & \wedge & \\ TR(Q_{A_1}, A_1, B_1) & & TR(Q_{A_2}, A_2, B_2) \end{array}$$

On peut appliquer (HI) à $TR(Q_{A_1}, A_1, B_1)$ et $TR(Q_{A_2}, A_2, B_2)$. On en déduit que Q' est une arbre d'expansion vérifiant $Q'^s = TR(Q_{A_1}, A_1, B_1)^s \wedge TR(Q_{A_2}, A_2, B_2)^s = B_1 \wedge B_2$.

5- $A = \exists x A(x)$ et $B = \exists x B(x)$.

$$Q' = TR\left(\begin{array}{ccccc} & & \exists x A(x) & & \\ & {\scriptstyle t_1} \swarrow & & \searrow {\scriptstyle t_n} & \\ Q_1 & & \cdots & & Q_n \end{array}, \exists x A(x), \exists x B(x)\right) =$$

$$\begin{array}{ccccc} & & \exists x B(x) & & \\ & {\scriptstyle t_1} \swarrow & & \searrow {\scriptstyle t_n} & \\ TR(Q_1, B_1(t_1), B(t_1)) & & \cdots & & TR(Q_n, B_n(t_n), B(t_n)) \end{array}$$

Où $B_i(t_i) \preceq A(t_i)$ pour $i = 1, \ldots, n$. Là encore, (HI) peut être appliquée aux fils. La

condition nécessaire $TR(Q_i, B_i(t_i), B(t_i))^s = B(t_i)$ pour que Q' soit un arbre d'expansion en découle. De plus, nous avons $Q'^s = \exists x B(x)$.

6- $A = \forall x A(x)$ et $B = \forall x B(x)$.

$$TR\left(\begin{array}{c}\forall x A(x)\\ \Big|_X\\ Q_0\end{array}, \forall x A(x), \forall x B(x)\right) = \begin{array}{c}\forall x B(x)\\ \Big|_X\\ TR(Q_0, B'(X), B(X))\end{array}$$

Où $Q_0^s = B'(X) \preceq A(X)$. Une fois de plus, (HI) peut être appliquée à $TR(Q_0, B'(X), B(X))$. La condition $TR(Q_0, B'(X), B(X))^s = B(X)$ est donc vérifiée. On en conclu que Q' est un arbre d'expansion et il vérifie $Q'^s = \forall x B(x)$.

□

La conclusion est qu'à partir d'une formule donnée A et d'un arbre d'expansion ou d'un arbre d'expansion réduit dont elle est la formule associée, nous sommes capables de construire un arbre d'expansion dont B est la formule associée pour toute formule B vérifiant $A \preceq B$. Nous vérifions maintenant que la condition nécessaire sur la relation de dépendance n'est pas affectée, ainsi que la condition sur les variables sélectionnées.

Lemme 34 *Soient Q un arbre d'expansion tel que $Q^s = A$ et B tel que $A \preceq B$. Si $<_Q$ est acyclique, il en est de même pour $<_{TR(Q,A,B)}$.*

Preuve Cette propriété est préservée du simple fait que l'on n'ajoute ni n'efface aucune variable sélectionnée, ni de terme expansé. On a même, en réalité $<_Q = <_{TR(Q,A,B)}$. □

Lemme 35 *Soient Q un arbre d'expansion tel que $Q^s = A$ et B tel que $A \preceq B$. S'il n'y a pas de variable sélectionnée dans Q qui soit libre dans Q'^s, il n'y a pas de variable sélectionnée dans $TR(Q, A, B)$ qui soit libre dans B.*

Preuve Le seul cas où nous sommes susceptibles d'introduire de nouvelles variables libres dans B qui ne figureraient pas dans A, nous prenons la précaution – voir règle 1 de la transformation – de les renommer pour éviter ce genre de conflit. D'autre part, nous n'ajoutons aucune variable sélectionnée. □

Nous montrons maintenant que la conjonction des clauses complètes de l'arbre d'expansion représentant A est plus petite respectivement à $\preceq$ que la conjonction de clauses complètes de l'arbre obtenu pour toute formule B telle que $A \preceq B$. Or, si c'est le cas, $\wedge cc(Q_B)$ est une tautologie, et de plus, $\mathcal{M}$ couvre Q_B.

Lemme 36 *Soient Q un arbre d'expansion tel que $Q^s = A$ et B telle que $A \preceq B$. Si $Q' = TR(Q, A, B)$, alors $\wedge cc(Q) \preceq \wedge cc(Q')$.*

Preuve Par induction sur la hauteur de Q.
Cas de base : $|Q| = 1$. Alors, $\wedge cc(Q) = A$ et $\wedge cc(Q') = B$. On a donc $A \preceq B$, puisqu'on ne peut appliquer TR que si cette condition est vérifiée.
Cas inductif : $|Q| > 1$. Nous notons l'hypothèse d'induction (HI).

1-

$$Q' = \begin{array}{ccc} & \vee & \\ TR(Q,A,B_1) & & B_2 \end{array} \qquad \wedge cc(Q') = Q'^s \vee \wedge cc(TR(Q,A,B_1)) \vee B_2$$

lorsque $A \preceq B_1$.

$$Q'' = \begin{array}{ccc} & \vee & \\ B_1 & & TR(Q,A,B_2) \end{array} \qquad \wedge cc(Q'') = Q''^s \vee B_1 \vee \wedge cc(TR(Q,A,B_2))$$

lorsque $A \preceq B_2$.
On a donc $\wedge cc(Q) \preceq \wedge cc(Q')$ puisque par (HI), $\wedge cc(Q) \preceq \wedge cc(TR(Q,A,B_1))$.
De la même façon, $\wedge cc(Q) \preceq \wedge cc(Q'')$ puisque par (HI), $\wedge cc(Q) \preceq \wedge cc(TR(Q,A,B_2))$.

2-

$$Q' = \begin{array}{ccc} & \wedge & \\ TR(Q,A,B_1) & & TR(Q,A,B_2) \end{array} \qquad \begin{array}{l} \wedge cc(Q') = Q'^s \vee \\ (\wedge cc(TR(Q,A,B_1))) \wedge \\ (\wedge cc(TR(Q,A,B_2))) \end{array}$$

Par (IH) $\wedge cc(Q) \preceq \wedge cc(TR(Q,A,B_1))$ et $\wedge cc(Q) \preceq \wedge cc(TR(Q,A,B_2))$.
D'où, $\wedge cc(Q) \preceq \wedge cc(Q')$.

3 & 4-

$$Q' = \begin{array}{ccc} & \vee & \\ TR(Q_{A_1},A_1,B_1) & & TR(Q_{A_2},A_2,B_2) \end{array} \qquad \begin{array}{l} \wedge cc(Q') = B_1 \vee B_2 \vee \\ (\wedge cc(TR(Q_{A_1},A_1,B_1))) \vee \\ \wedge cc(TR(Q_{A_2},A_2,B_2))) \end{array}$$

Toujours par (HI), on a $\wedge cc(Q_{A_1}) \preceq \wedge cc(TR(Q_{A_1},A_1,B_1))$ et $\wedge cc(Q_{A_2}) \preceq \wedge cc(TR(Q_{A_2},A_2,B_2)))$. On en déduit, $\wedge cc(Q) = Q^s \vee \wedge cc(Q_{A_1}) \vee \wedge cc(Q_{A_2}) \preceq \wedge cc(Q')$.
De la même façon pour la conjonction, l'(HI) donne $\wedge cc(Q_{A_1}) \preceq \wedge cc(TR(Q_{A_1},A_1,B_1))$ et $\wedge cc(Q_{A_2}) \preceq \wedge cc(TR(Q_{A_2},A_2,B_2)))$. On en déduit, $\wedge cc(Q'') = B_1 \wedge B_2 \wedge \wedge cc(TR(Q_{A_1},A_1,B_1)) \wedge \wedge cc(TR(Q_{A_2},A_2,B_2))$.
D'où, $\wedge cc(Q) = Q^s \vee (\wedge cc(Q_{A_1}) \wedge \wedge cc(Q_{A_2})) \preceq \wedge cc(Q'')$.

5-

$$Q' = \begin{array}{ccc} & \exists x B(x) & \\ t_1 \swarrow & & \searrow t_n \\ Q'_1 & \cdots & Q'_n \end{array} \qquad \text{avec } Q'_i = TR(Q_i, B_i(t_i), B(t_i))$$

On a alors, $\wedge cc(Q') = \exists x B(x) \vee_{i=1}^{n} (\wedge cc(Q'_i))$. D'autre part, $\wedge cc(Q) = \exists x A(x) \vee_{i=1}^{n} (\wedge cc(Q_i))$, et par (HI), $\wedge cc(Q_i) \preceq \wedge cc(Q'_i)$ pour $1 \leq i \leq n$. Or, $A(x) \preceq B(x)$ est nécessaire pour l'application de la règle. Donc, $\exists x A(x) \vee_{i=1}^{n} (\wedge cc(Q_i)) \preceq \exists x B(x) \vee_{i=1}^{n} (\wedge cc(Q'_i))$. i.e. $\wedge cc(Q) \preceq \wedge cc(Q')$.

6-

$$Q' = \begin{array}{c} \forall x B(x) \\ X \,\Big|\, \\ TR(Q_0, B'(X), B(X)) \end{array} \qquad \begin{array}{ll} \wedge cc(Q') = & \forall x B(x) \vee \wedge cc(TR(Q_0, B'(X), B(X))) \\ \text{et} & \wedge cc(Q) = \forall x A(x) \vee \wedge cc(Q_0) \end{array}$$

Là encore, $A(x) \preceq B(x)$ est nécessaire pour l'application de la règle. Par (HI), nous avons $\wedge cc(Q_0) \preceq \wedge cc(TR(Q_0, B'(X), B(X)))$. Donc, $\wedge cc(Q) \preceq \wedge cc(Q')$.

□

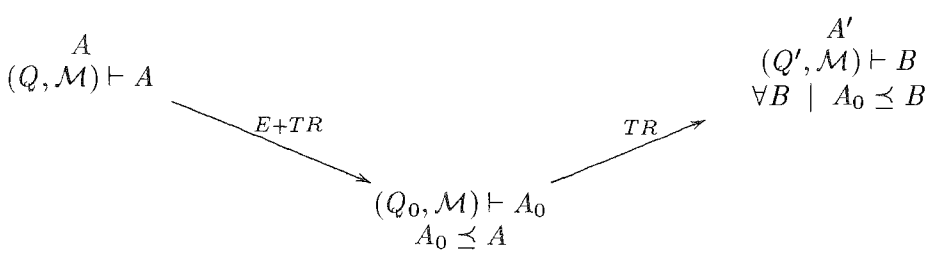
L'intérêt de ce résultat est qu'il nous permet d'affirmer que la transformation TR n'affecte pas le fait que $\mathcal{M}$ couvre la conjonction des clauses complètes.

Corollaire 1 *Soit Q un arbre d'expansion ou un arbre d'expansion réduit vérifiant $Q^s = A$, soit B une formule telle que $A \preceq B$ et soit $Q' = TR(Q, A, B)$. Si $\mathcal{M}$ couvre $\wedge cc(Q)$, alors $\mathcal{M}$ couvre $\wedge cc(Q')$.*

Ce qui nous permet de conclure avec le théorème de correction – et de complétude puisqu'il considère toute formule A telle que $A \preceq B$ – pour la transformation TR. Ce théorème implique également que la définition des preuves réduites est correcte puisque chacune de celles-ci peut être transformée en preuve par expansion.

Théorème 37 $(Q, \mathcal{M}) \vdash_R A$ *implique* $(TR(Q, A, B), \mathcal{M}) \vdash B$ *pour toute formule B vérifiant $A \preceq B$.*

Les théorèmes 32 et 37 valident les résultats annoncés au début de ce chapitre. Rappelons que ces transformations sont automatiques et terminent. Nous pouvons reprendre le schéma donné en début de chapitre en le complétant :



Ceci constitue les manipulations de base sur les preuves par expansion, que nous réutiliserons dans la suite. La relation $\preceq$ que nous avons introduite peut être comparée à l'abstraction de Plaisted [Pla81] qui consiste à effacer certains arguments de fonctions ou de prédicats. Toutefois, nous permettons d'aller plus loin, puisque la relation $\preceq$ permet par exemple d'effacer des hypothèses inutiles, ou d'ajouter une disjonction dans le but de la formule.

Avant d'aller plus loin dans les processus d'analogie, nous avons besoin d'un outil fondamental qui est le filtrage. C'est l'objet du chapitre II. En première lecture, ce chapitre peut éventuellement être omis. Toutefois, il contient la définition des λ-termes simplement typés dont nous avons besoin pour représenter les formules.

II
Filtrage AC d'ordre deux

Dans ce chapitre, nous abordons un sujet à priori assez distant de la preuve par analogie. En fait, le filtrage d'ordre deux constitue un outil très important pour la reconnaissance des similitudes, puis des différences entre les formules. D'ailleurs, certains l'utilisent déjà pour la preuve par analogie [BdlTC87, BdlTC88, BdlTK92, KW94] En utilisant des termes d'ordre deux, il est possible de représenter des schémas de formules d'ordre un. Le filtrage permet alors de savoir si une formule donnée est une instance d'un certain schéma. C'est une information primordiale pour pratiquer l'analogie.

Le filtrage d'ordre deux a été introduit par G. Huet dans sa thèse [Hue76], puis appliqué à la transformation de programmes par Huet et Lang [HL78]. Pour notre part nous l'utiliserons pour étudier les analogies entre les formules. Or il s'avère que l'algorithme classique, pourtant puissant, possède un handicap important pour l'utilisation que nous voulons en faire; en effet, il n'est pas possible, avec cet algorithme, de tenir compte des propriétés algébriques des opérateurs. Pourtant, il est essentiel pour nous de savoir reconnaître des similitudes modulo certaines propriétés élémentaires des connecteurs logiques, comme l'associativité et la commutativité. Ceci accroît considérablement la puissance de l'outil de reconnaissance. C'est pourquoi nous avons proposé un algorithme qui combine le filtrage d'ordre deux et le filtrage dans une théorie d'ordre un. La théorie qui nous intéresse particulièrement est la théorie associative-commutative (AC). Pour ce faire, nous nous sommes tout d'abord inspirés des travaux qui ont été réalisés dans un cadre plus général, à savoir l'unification d'ordre supérieur dans une théorie régulière [NQ91], puis quelconque [QW92] (bien que pour le filtrage une théorie non régulière ne puisse pas convenir). L'algorithme résultant possède les propriétés nécessaires de correction, complétude et terminaison, mais un certain nombre de mécanismes trop lourds le rendent inefficace, et fournissent un grand nombre de solutions superflues. En conséquence, en collaboration avec Zhenyu Qian, nous proposons un algorithme remédiant aux inconvénients les plus marqués. Cet algorithme s'applique aussi bien au cas syntaxique qu'au cas AC. Il élimine presque totalement les redondances de solutions, réutilise au maximum les calculs déjà effectués, et allège largement les mécanismes internes de l'algorithme.

II.1 Représentation des formules par des λ-termes simplement typés

Dans tout ce document, nous utilisons la logique du premier ordre. Il est donc a priori inutile d'introduire les λ-termes simplement typés pour les représenter. En réalité, un certain nombre de

raisons indique que cette représentation est la mieux indiquée : tout d'abord, dans ce chapitre, nous allons définir des schémas de formules, ce qui nous oblige à utiliser l'ordre deux. De plus, nous allons définir un algorithme de filtrage qui s'applique à des λ-termes d'ordre deux. Enfin, les λ-termes simplement typés sont utilisés par Miller et Pfenning pour la représentation des formules dans les preuves par expansion, ce qui leur permet de traiter l'ordre supérieur. Il est donc nécessaire pour nous d'introduire un tel langage. De plus, nous envisageons, dans un avenir proche d'étendre certaines des techniques présentées ici à l'ordre supérieur. Nous pourrons alors réutiliser directement le formalisme introduit ici. Notons également que cette représentation est très largement utilisée dans les systèmes de déduction automatique. Citons par exemple Elf. Par conséquent, l'utilisation de cette représentation des formules nous permet à la fois d'utiliser l'ordre deux dont nous avons besoin ici, et de nous situer dans un cadre qui nous permettra d'étendre nos techniques sans problèmes liés à la représentation.

II.1.1 Types

Nous utilisons la définition standard des types simples.

Définition 1 Soit T_0, un ensemble fini de types élémentaires (ou simples). L'ensemble des types T est défini récursivement par :

$$\begin{cases} \tau \in T_0 & \Rightarrow \tau \in T \\ \tau_1, \tau_2 \in T & \Rightarrow \tau_1 \to \tau_2 \in T \end{cases}$$

□

La relation $\to$ associe à droite. Dans la suite, nous utiliserons la notation $\tau_1 \times \tau_2 \times \cdots \times \tau_n \to \tau$ pour $(\tau_1 \to \cdots (\tau_n \to \tau) \cdots)$, et même, sous forme plus synthétique $\overline{\tau_n} \to \tau$.

Définition 2 L'ordre d'un type, noté $O(\tau)$ est défini par :

$$\begin{cases} \tau \in T_0 \Rightarrow O(\tau) = 1 \\ O(\tau_1 \to \tau_2) = Max\{1 + O(\tau_1); O(\tau_2)\} \end{cases}$$

□

τ, ι et o seront utilisés comme symboles de types. Avec ι le type individuel, et o le type proposition (ou de vérité).

II.1.2 λ-termes simplement typés

Pour plus de détails sur les résultats classiques du λ-calcul, voir par exemple [HS86].

Définition 3 *Signature* : on suppose qu'il existe un ensemble dénombrable $\mathcal{V}_\tau$ de symboles de variables pour chacun des types $\tau \in T$. L'ensemble des symboles de variables $\mathcal{V}$ est défini par $\mathcal{V} = \cup_{\tau \in T} \mathcal{V}_\tau$. De même, on suppose qu'il existe un ensemble dénombrable $\mathcal{C}_\tau$ de symboles de constantes pour chacun des types $\tau \in T$ et l'ensemble $\mathcal{C}$ des symboles de constantes est défini par $\mathcal{C} = \cup_{\tau \in T} \mathcal{C}_\tau$. □

Définition 4 *λ-termes simplement typés* :
On note $\mathcal{T}_\tau$ l'ensemble des termes de type τ.

Atomes	$u \in \mathcal{V}_\tau \cup \mathcal{C}_\tau$	$\Rightarrow$	$u \in \mathcal{T}_\tau$
Application	$u \in \mathcal{T}_{\tau_1 \to \tau_2}$ et $t \in \mathcal{T}_{\tau_1}$	$\Rightarrow$	$(u\ t) \in \mathcal{T}_{\tau_2}$
Abstraction	$x \in \mathcal{V}_{\tau_1}$ et $t \in \mathcal{T}_{\tau_2}$	$\Rightarrow$	$\lambda x.t \in \mathcal{T}_{\tau_1 \to \tau_2}$

□

Définition 5 L'ordre d'un λ-terme est défini par l'ordre de son type. □

Nous utiliserons fréquemment les notations suivantes :

$u(t_1, \ldots, t_n)$	pour	$((\cdots((u\ t_1)t_2)\cdots)t_n)$
$\lambda x_1, \ldots, x_n.t$	pour	$\lambda x_1.(\lambda x_2(\cdots(\lambda x_n.t)\cdots))$
et même $\lambda\overline{x_n}.t$	pour	$\lambda x_1, \ldots, x_n.t$

Nous pourrons également indicer les symboles ou les termes par leur type lorsque cela est nécessaire. Ainsi, x_ι signifie que x est de type ι.

Les termes que nous utiliserons dans la suite seront tels que l'ordre des constantes sera inférieur ou égal à trois, et l'ordre des variables, inférieur ou égal à deux. Remarquons au passage que l'ordre choisi pour le type o est un, mais il est parfois considéré d'ordre deux.

Définition 6 Dans le λ-terme $\lambda x.t$, les occurrences de la variable x dans t sont dites liées. Toute occurrence non liée d'une variable est dite libre. Plus généralement, nous utiliserons les dénominations suivantes pour les différentes parties des λ-termes :

$$\overbrace{\lambda\overline{x_n}}^{\textit{lieur}} . \underbrace{\overbrace{u}^{\textit{tête}}\ \overbrace{(t_1, \ldots, t_n)}^{\textit{arguments}}}_{\textit{matrice}}$$

L'ensemble des variables libres d'un λ-terme t est noté $\mathcal{FV}(t)$ et sa tête $\mathcal{H}(t)$. □

Définition 7 Une substitution σ est une fonction notée $\{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ ou $\{\overline{X_n \mapsto t_n}\}$ où $X_i \in \mathcal{V}_{\tau_i}$ et $t_i \in \mathcal{T}_{\tau_i}$ pour $i = 1, \ldots, n$. □

Définition 8 L'application d'une substitution $\sigma = \{\overline{X_n \mapsto t_n}\}$ à un terme s notée $\sigma(s)$ est le terme s où l'on a remplacé, pour chaque élément $X_i \mapsto t_i$ de σ, toutes les occurrences libres de X_i dans s par t_i. Si des variables libres dans t_i sont liées dans s, elles sont renommées avant application de la substitution. $\sigma(s)$ peut également être notée $s[\overline{X_n \mapsto t_n}]$. □

Définissons les opérations classiques du λ-calcul.

α-conversion

$$\lambda x.t \ \to_\alpha \ \lambda y.t[x \mapsto y]$$

si y n'est pas libre dans t, et où $t[x \mapsto y]$ signifie que toutes les occurrences libres de x dans t sont remplacées par y. L'α-conversion est aussi appelée renommage des variables liées.

β-réduction

$$(\lambda x.t)\ u \ \to_\beta \ t[x \mapsto u]$$

La β-réduction correspond à l'application d'une fonction à un argument, ou, pour la programmation, au remplacement d'un paramètre formel, par un paramètre effectif.

η-réduction

$$\lambda x.(t\ x) \ \rightarrow_{\eta\downarrow} \ t$$

Lorsque x n'a pas d'occurrence libre dans t. Cette opération correspond à une contraction. Dans la suite, nous utiliserons plutôt l'opération inverse de η-expansion définie ci-dessous.

η-expansion

$$t \ \rightarrow_{\eta\uparrow} \ \lambda x.(t\ x)$$

Pour ce qui nous intéresse, la η-expansion est utilisée pour obtenir des λ-termes dont la matrice est toujours de type simple.

Nous évoquerons l'α-conversion lorsque cela sera nécessaire, mais en général, les termes seront considérés comme équivalents modulo cette opération. C'est à dire que nous assimilerons l'égalité syntaxique $=$ et l'équivalence modulo le renommage des variables liées $=_\alpha$.

Définition 9 Un terme est dit en β-forme normale, lorsqu'aucune β-réduction ne peut plus lui être appliquée. □

Il est important de remarquer que tous les termes possèdent une β-forme normale unique. En effet, une des principales propriétés des λ-termes simplement typés est d'être fortement normalisables. Donc, comme la β-réduction est confluente et qu'il n'existe pas de dérivation infinie par la règle $\rightarrow_\beta$, tout λ-terme simplement typé possède une β-forme normale unique. L'intérêt, est que cela permet de comparer les termes syntaxiquement, lorsqu'ils sont en forme normale. Ce qui n'est pas possible en λ-calcul pur (non typé).

Définition 10 Un terme en β-forme normale est dit en $\beta\eta$-forme normale longue ($\beta\eta fnl$), lorsqu'il n'est plus possible de lui appliquer la η-expansion sans créer de β-redex. □

Sauf indication contraire, nous ne manipulerons que des termes en $\beta\eta fnl$. Les termes que nous avons définis vont être utilisés pour représenter les formules.

II.1.3 Représentation de formules

Nous pouvons maintenant représenter les formules par les λ-termes simplement typés. Il suffit de voir les connecteurs comme des fonctions particulières. Pour les connecteurs, une formule $\forall x\ A(x)$ est représentée par le $\lambda-$terme $\forall(\lambda x.A(x))$. De la même façon, une formule $\exists x\ A(x)$ est représentée par le $\lambda-$terme $\exists(\lambda x.A(x))$. Pour les λ-termes $\forall$ et $\exists$ sont donc des fonctions particulières.

On peut remarquer qu'à une variable liée au sens des formules correspond une variable liée au sens des λ-termes. L'ordre d'une formule est l'ordre du terme qui la représente. Nous nous intéressons aux formules du premier ordre. Par conséquent, les termes qui les représentent sont également du premier ordre. Par contre, pour représenter des schémas de formules, nous utiliserons des termes d'ordre deux, ce qui signifie simplement que nous autorisons des variables libres d'ordre deux. Une illustration de l'utilisation de l'ordre deux se trouve dans l'exemple 11.

Dans la suite, nous utiliserons les symboles x, y, z pour représenter des variables liées.

II.2 Filtrage d'ordre deux

Nous allons, dans un premier temps, présenter une version de l'algorithme classique de filtrage d'ordre deux, dû à Huet [Hue76]. Ceci va nous permettre de nous familiariser avec l'ordre supérieur et de comprendre les mécanismes de base du filtrage. Mais avant de rentrer dans les détails techniques, regardons sur un exemple (tiré de [HL78]), ce qu'il est possible de faire, de façon intuitive, avec le filtrage d'ordre deux.

Exemple 11 Il s'agit de reconnaître une fonction récursive, afin, éventuellement, d'en donner une version non récursive. Un schéma représentant les fonctions récursives peut être :

$$f(x) = si\ a(x)\ alors\ b(x)\ sinon\ h(d(x), f(e(x)))$$

où l'on peut voir que l'appel récursif se situe dans la partie "sinon". Par ailleurs, il ne s'agit pas nécessairement de récursivité terminale puisque $f(e(x))$ est argument de h. Ce schéma peut être représenté par le λ-terme suivant :

$$S = \lambda u\ .fixpt(\lambda f\ x.cond(A(x), B(x), H(D(x), f(E(x)))), u)$$

où $fixpt$ est un opérateur point fixe, $cond$ est la conditionnelle, et A, B H et E des variables libres. Pour les types, nous avons :

$\tau(fixpt) = ((\iota \to \iota) \times \iota \to \iota) \times \iota \to \iota$ $\tau(cond) = bool \times \iota \times \iota \to \iota$ $\tau(H) = \iota \times \iota \to \iota$ $\tau(B) = \tau(D) = \tau(E) = \tau(f) = \iota \to \iota$ $\tau(A) = \iota \to bool$ $\tau(x) = \iota$ $\tau(u) = \iota$.

Prenons à présent la fonction rev qui renverse une liste. Elle s'écrira :

$$rev(x) = si\ null(x)\ alors\ nil \\ sinon\ append(rev(cdr(x)), cons(car(x), nil))$$

Ce qui s'exprime par le λ-terme :

$$\lambda u\ .fixpt(\lambda rev\ x.cond(null(x), nil, append(rev(cdr(x)), cons(car(x), nil))), u)$$

Le filtrage entre le schéma définissant les fonctions récursives et la fonction rev donne trois solutions :

$$\begin{cases} A & \mapsto & \lambda x.null(x) \\ B & \mapsto & \lambda x.nil \\ E & \mapsto & \lambda x.cdr(x) \\ H & \mapsto & \lambda x\ y.append(y, x) \\ D & \mapsto & \lambda x.cons(car(x), nil) \end{cases} \qquad \begin{cases} A & \mapsto & \lambda x.null(x) \\ B & \mapsto & \lambda x.nil \\ E & \mapsto & \lambda x.cdr(x) \\ H & \mapsto & \lambda x\ y.append(y, cons(x, nil)) \\ D & \mapsto & \lambda x.car(x) \end{cases}$$

$$\begin{cases} A & \mapsto & \lambda x.null(x) \\ B & \mapsto & \lambda x.nil \\ E & \mapsto & \lambda x.cdr(x) \\ H & \mapsto & \lambda x\ y.append(y, cons(car(x), nil)) \\ D & \mapsto & \lambda x.x \end{cases}$$

Ainsi, on peut envisager d'appliquer une opération de dérécursivation à la fonction, puisqu'il existe des algorithmes, qui, appliqués à une fonction dont le schéma est S, la transforme en fonction itérative. ■

Intuitivement, nous voulons utiliser ce qui est proposé ici pour des fonctions, sur des formules et leurs preuves. C'est-à-dire qu'à partir du filtrage d'une formule avec un schéma de formule utilisé comme référence, nous allons appliquer une transformation à la preuve de la formule de référence pour retrouver une preuve de la formule objet. Bien sûr, il faut pour cela qu'il existe des solutions au filtrage. Intéressons nous maintenant à l'algorithme. Pour ce faire, nous avons besoin d'introduire quelques définitions supplémentaires.

Dans ce chapitre, nous utilisons les λ-termes simplement typés d'ordre deux. Cela signifie, en particulier, que nous autorisons des variables d'ordre deux. Par exemple, la variable F dont le type est $\tau(F) = \mathit{entier} \times \mathit{entier} \to \mathit{entier}$ est une variable d'ordre deux. Elle peut par exemple recevoir la valeur $\lambda x_1 x_2.x_1 + x_2$, c'est-à-dire l'addition. Nous donnons maintenant les définitions fastidieuses, mais classiques pour les substitutions, filtres et unificateurs qui nous seront utiles par la suite.

Définition 12 Le domaine d'une substitution $\sigma = \{\overline{X_n \mapsto t_n}\}$ noté $\mathcal{D}om(\sigma)$ est l'ensemble des variables $\{\overline{X_n}\}$. Son rang, noté $\mathcal{R}g(\sigma)$ est l'ensemble des termes $\{\overline{t_n}\}$. On note $\mathcal{I}(\sigma)$ les variables libres introduites par la substitution, c'est-à-dire $\cup_{i=1}^{n} \mathcal{FV}(t_i)$. □

Pour éviter les conflits de variables, nous supposerons que toutes les variables liées de s sont renommées de sorte qu'il n'y ait pas de variable commune avec $\mathcal{I}(\sigma)$.

Définition 13 La composition de deux substitutions σ et θ est définie par $(\sigma\theta)(X) = \sigma(\theta(X))$ pour tout $X \in \mathcal{V}$. □

Définition 14 Pour un sous-ensemble $\mathcal{W}$ de $\mathcal{V}$, la restriction $\sigma_{|\mathcal{W}}$ est la substitution définie par $\sigma_{|\mathcal{W}}(X) = \sigma(X)$ si $X \in \mathcal{W}$, et $\sigma_{|-\mathcal{W}}(X) = X$ sinon. D'autre part, $\sigma_{|-\mathcal{W}}$ représente la restriction $\sigma_{|\mathcal{D}om(\sigma)-\mathcal{W}}$. □

Définition 15 Une substitution σ est normalisée si t est en $\beta\eta$-forme normale longue pour tout $t \in \mathcal{R}ang(\sigma)$. Une substitution normalisée satisfait le condition $(\sigma(t)_{\beta\downarrow})_{\eta\uparrow} = \sigma(t)_{\beta\downarrow}$. □

Nous ne considérerons que les substitutions normalisée. Dans la suite, nous utiliserons σ, θ et ρ pour représenter des substitutions.

Définition 16 Pour toutes substitutions σ et θ, $\sigma = \theta[\mathcal{W}]$ signifie que $\sigma(X) = \theta(X)$ pour tout $X \in \mathcal{W}$. De plus, $\sigma \leq \theta[\mathcal{W}]$ s'il existe une substitution ρ telle que $\rho\sigma = \theta[\mathcal{W}]$. $[\mathcal{W}]$ peut être omis lorsque $\mathcal{W} = \mathcal{V}$. D'autre part, pour toute substitution σ, et $\mathcal{W} \in \mathcal{D}om(\sigma)$, il existe toujours une substitution σ' telle que $\mathcal{D}om(\sigma') \cap \mathcal{I}(\sigma') = \emptyset$, $\mathcal{D}om(\sigma) = \mathcal{D}om(\sigma')$, $\sigma \leq \sigma'[\mathcal{W}]$ et $\sigma' \leq \sigma[\mathcal{W}]$. Puisqu'il n'y a pas, généralement d'ambiguïté sur $\mathcal{W}$, nous nous restreindrons à σ'. L'intérêt étant que $\mathcal{D}om(\sigma') \cap \mathcal{I}(\sigma') = \emptyset$ implique $\sigma'\sigma' = \sigma'$. □

Définition 17 Une paire $s =^? t$ est une paire d'unification si s et t sont deux termes d'ordre deux ayant même type. De plus, puisque les termes sont en forme longue, nous pouvons supposer qu'ils possèdent le même lieur. Une paire $s \stackrel{\triangleleft}{=} t$ est une paire de filtrage si $s =^? t$ est une paire d'unification avec t clos. Le symbole $\stackrel{\triangleleft}{=}$ est donc orienté. Nous dirons généralement qu'un terme clos, ou dont la tête est une constante ou une variable liée, est rigide; il est flexible sinon. Une paire de filtrage a donc un terme rigide à droite et éventuellement flexible à gauche. Un ensemble S de paires d'unification est un problème d'unification, et un ensemble S de paires de filtrage est un problème de filtrage. □

Définition 18 Une substitution σ est un unificateur de $s =^? t$ si $\sigma(s) = \sigma(t)$. Une substitution σ est un filtre de $s \stackrel{?}{=} t$ si $\sigma(s) = t$. σ est un unificateur pour un problème d'unification S si σ unifie toutes les paires de S. De la même façon, σ est un filtre d'un problème de filtrage S si elle filtre toutes les paires de S. □

Définition 19 $\mathcal{U}(S)$ est l'ensemble de tous les unificateurs du problème S. Un ensemble complet d'unificateurs de S, noté $\mathcal{ECU}(S)$ est un sous ensemble de $\mathcal{U}(S)$ tel que pour toute $\sigma \in \mathcal{U}(S)$, il existe $\theta \in \mathcal{ECU}(S)$ telle que $\theta \leq \sigma$. De même, $\mathcal{F}(S)$ est l'ensemble de tous les filtres du problème S. Un ensemble complet de filtres de S, noté $\mathcal{ECF}(S)$ est un sous ensemble de $\mathcal{F}(S)$ tel que pour toute $\sigma \in \mathcal{F}(S)$, il existe $\theta \in \mathcal{ECF}(S)$ telle que $\theta \leq \sigma$. □

Notons qu'il n'existe pas de notion de minimalité pour les filtres, puisque les filtres sont des substitutions closes. Il faut entendre par substitution close, qu'à toute variable de son domaine, elle associe soit un terme clos soit cette variable elle -même. En effet, toute variable X n'apparaissant pas dans l'ensemble des variables libres du problème considéré peut recevoir n'importe quel terme. En conséquence, nous considérons que la substitution lui associe elle-même. De plus, il arrive qu'une variable apparaissant dans les variables libres du problème puisse recevoir n'importe quel terme. Par exemple pour le filtrage de $F(X) \stackrel{?}{=} a$, la solution $\sigma = \{F \mapsto \lambda x.a\}$ peut associer n'importe quel terme à X. Nous considérons donc ici que σ associe X à X.

Nous pouvons à présent aborder l'algorithme de filtrage. L'algorithme que nous présentons ici est celui de Huet et Lang [HL78], si ce n'est que nous en avons modifié la présentation pour le donner sous forme de règles. Pour des raisons pratiques, nous représentons un problème de filtrage par une liste de paires. L'opération @ représente la concaténation de listes. Nous allons présenter cet algorithme comme la transformation d'un couple $< S, \sigma >$, où S est un problème de filtrage et σ une substitution. Une donnée de cet algorithme est de la forme $< S_0, \{\} >$. Il y a succès lorsque l'algorithme termine avec un couple de la forme $< \{\}, \theta_f >$. Dans ce cas, θ_f est un filtre pour le problème initial, S_0. Il y a échec lorsqu'aucune séquence de transformation ne termine avec un couple de ce type.

En premier lieu, donnons la règle de décomposition rencontrée dans la plupart des algorithmes de filtrage et d'unification :

$$\frac{< \{\lambda\overline{x}.f(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_n})\}@S, \sigma >}{< \{\lambda\overline{x}.s_1 \stackrel{?}{=} \lambda\overline{x}.t_1, \ldots, \lambda\overline{x}.s_n \stackrel{?}{=} \lambda\overline{x}.t_n\}@S, \sigma >} \quad \text{(D)}$$

lorsque f est un symbole de constante ou une variable liée.

Deux autres règles couvrent le cas appelé "flexible-rigide", puisque le terme de gauche possède une variable libre en tête. Il s'agit de la projection et de l'imitation :

$$\frac{< \{\lambda\overline{x}.F(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.t\}@S, \sigma >}{< \sigma'\{\lambda\overline{x}.s_i \stackrel{?}{=} \lambda\overline{x}.t\}@\sigma'(S), \sigma'\sigma >} \quad \text{(P)}$$

avec $\sigma' = \{F \mapsto \lambda\overline{y_n}.y_i\}$ pour un i tel que $1 \leq i \leq n$ et $\tau(s_i) = \tau(t)$. Notons que la projection à l'ordre deux est sensiblement plus simple qu'à l'ordre supérieur en général. En effet, dans le cas général, la variable y_i peut être d'ordre supérieur à un, et donc posséder des arguments, ce qui complique largement les choses. La règle d'imitation s'exprime ainsi :

$$\frac{< \{\lambda\overline{x}.F(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m})\}@S, \sigma >}{< \{\sigma'(\lambda\overline{x}.F(\overline{s_n})) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m})\}@\sigma'(S), \sigma'\sigma >} \quad (I)$$

avec $\sigma' = \{F \mapsto \lambda\overline{y_n}.f(H_1(\overline{y_n}), \cdots, H_m(\overline{y_n}))\}$ où les $H_1, \cdots, H_m$ sont de nouvelles variables libres du type adéquat. Ce sont donc des variables introduites par le processus de filtrage. D'autre part, selon le type de t_i, $H_i(\overline{y_n})$ peut être expansé en $\lambda\overline{w_{k_i}}.H_i(\overline{w_{k_i}}, (\overline{y_n}))$. Mais nous tenterons de ne pas surcharger avec cette notation. Cette règle s'appelle imitation, car elle consiste à imiter la tête du terme rigide et à introduire des variables pour filtrer les arguments. Regardons ce que donne l'application de cette règle sur un exemple simple.

Exemple 20 Prenons la paire de filtrage $\lambda x.F(x,a) \stackrel{\triangleleft}{=} \lambda y.f(a,x,a)$. L'application de l'imitation donne :

$$\frac{< \{\lambda x.F(x,a) \stackrel{\triangleleft}{=} \lambda x.f(a,x,a)\}, \sigma >}{< \sigma'(\lambda x.F(x,a)) \stackrel{\triangleleft}{=} \lambda x.f(a,x,a)\}, \sigma'\sigma >} \quad (I)$$

avec $\sigma' = \{F \mapsto \lambda z_1 z_2.f(H_1(z_1,z_2), H_2(z_1,z_2), H_3(z_1,z_2))\}$. Donc,

$$\sigma'(\lambda x.F(x,a)) = \lambda x.((\lambda z_1 z_2.f(H_1(z_1,z_2), H_2(z_1,z_2), H_3(Z_1,z_2))\ (x,a))$$

ce qui donne, après β-réduction: $\lambda x.f(H_1(x,a), H_2(x,a), H_3(x,a))$. Nous pouvons voir comment cette règle tient compte des arités. La seule règle à pouvoir être appliquée ensuite, sera la décomposition. Ce qui mènera aux trois paires de filtrage :

$$\{\lambda x.H_1(x,a) \stackrel{\triangleleft}{=} \lambda x.a,\ \lambda x.H_2(x,a) \stackrel{\triangleleft}{=} \lambda x.x,\ \lambda x.H_3(x,a) \stackrel{\triangleleft}{=} \lambda x.a\}$$

D'autre part, en appliquant la projection sur le premier argument à la place de l'imitation, nous aurions :

$$\frac{< \{\lambda x.F(x,a) \stackrel{\triangleleft}{=} \lambda x.f(a,x,a)\}, \sigma >}{< \{\lambda x.x \stackrel{\triangleleft}{=} \lambda x.f(a,x,a)\}, \sigma >} \quad (P)$$

■

Les cas d'échec surviennent lorsque pour une paire $\lambda\overline{x}.s \stackrel{\triangleleft}{=} \lambda\overline{x}.t$, la tête de s et la tête de t sont deux constantes ou variables liées différentes, et lorsque les types de s et t sont différents. Sinon, les règles sont appliquées de façon indéterministe et respectivement aux conditions d'application. Cet algorithme est correct et complet. De plus, il termine toujours, ce qui le rend très intéressant du point de vue pratique. Pour le montrer, il suffit de prendre la mesure de complexité (ξ_2, ξ_1) où ξ_1 est le multi-ensemble des tailles des termes membres gauches et ξ_2 le multi-ensemble des tailles des termes membres droits. Cette mesure décroît strictement dans tous les cas, excepté pour l'imitation. Mais après l'imitation, la seule règle applicable est la décomposition, et la mesure décroît strictement en considérant l'application successive de ces deux règles.

Ceci constitue donc l'algorithme classique de filtrage syntaxique d'ordre deux. C'est un outil puissant, mais insatisfaisant pour le but que nous nous sommes fixé. Considérons par exemple un schéma de formule logique (i.e. une formule représentée par un terme d'ordre deux) dont la tête est un connecteur : $A \wedge B$, A et B étant des formules arbitrairement complexes. Si nous avons d'autre part une formule d'ordre un $b \wedge a$ telle que b filtre avec B et a filtre avec A. Nous n'arriverons pas à filtrer $b \wedge a$ avec le schéma $A \wedge B$ si nous ne prenons pas en considération la commutativité du connecteur $\wedge$. C'est pourquoi il est nécessaire d'intégrer ce que

nous appellerons une théorie au processus de filtrage. En fait, nous allons utiliser la théorie Associative-Commutative (AC). Cette théorie reflète assez bien les propriétés des connecteurs, et a été largement étudiée. De plus, ce choix a des justifications techniques que nous verrons par la suite. Dans la suite, nous allons donner un algorithme naïf construit sur le modèle de ce qui a été fait par Nipkow et Qian pour l'unification d'ordre supérieur dans une théorie régulière. Ensuite, nous donnerons un algorithme plus efficace, mais spécifique à la théorie AC.

II.3 Combinaison de l'ordre deux et d'une théorie

A nouveau, nous avons besoin de quelques définitions complémentaires.

Définition 21 Une théorie algébrique E est un ensemble fini de paires non ordonnées de termes $g \simeq d$ (appelées équations) telles que $\tau(g) = \tau(d)$. La E équivalence notée $=_E$ est la plus petite relation d'équivalence sur les termes telle que (i) $\sigma(g) =_E \sigma(d)$ pour toute substitution σ et toute équation $g \simeq d \in E$, (ii) si $u =_E v$ et $s =_E t$, alors $(u\ s) =_e (v\ t)$ et $\lambda x.s =_E \lambda x.t$. □

Définition 22 Une théorie est dite régulière, si $s =_E t$ implique $\mathcal{FV}(s) = \mathcal{FV}(t)$. □

Par exemple, la théorie AC,

$$AC = \begin{array}{c} \{f(x, f(y,z)) \simeq f(f(x,y),z); \\ f(x,y) \simeq f(y,x)\} \end{array}$$

est régulière. D'autre part, une constante de fonction est dite *libre*, si elle n'apparaît pas dans E. Nous utiliserons l'équivalence $=_{\beta\eta E}$ pour représenter l'équivalence modulo E et les β et η réductions. Un résultat de Breazu-Tannen [BT88] montre que pour tous termes u et v, $u =_{\beta\eta E} v \Longrightarrow u_{\beta\eta\uparrow} =_E v_{\beta\eta\uparrow}$. Ceci nous permet, puisque nous n'utilisons que les termes en forme longue, de ne considérer que l'équivalence $u =_E v$. Les définitions que nous avons données pour les substitutions dans la théorie vide s'expriment maintenant de la façon suivante.

Définition 23 Pour toutes substitutions σ et θ, $\sigma =_{\beta\eta E} \theta[\mathcal{W}]$ signifie que $\sigma(X) =_{\beta\eta E} \theta(X)$ pour tout $X \in \mathcal{W}$. De plus, $\sigma \leq_{\beta\eta E} \theta[\mathcal{W}]$ s'il existe une substitution ρ telle que $\rho\sigma =_{\beta\eta E} \theta[\mathcal{W}]$. $[\mathcal{W}]$ peut être omis lorsque $\mathcal{W} = \mathcal{V}$. D'autre part, pour toute substitution σ, et $\mathcal{W} \in \mathcal{D}om(\sigma)$, il existe toujours une substitution σ' telle que $\mathcal{D}om(\sigma') \cap \mathcal{I}(\sigma') = \emptyset$, $\mathcal{D}om(\sigma) = \mathcal{D}om(\sigma')$, $\sigma \leq_{\beta\eta E} \sigma'[\mathcal{W}]$ et $\sigma' \leq_{\beta\eta E} \sigma[\mathcal{W}]$. Puisqu'il n'y a pas, généralement d'ambiguïté sur $\mathcal{W}$, nous nous restreindrons à σ'. L'intérêt étant, comme pour la théorie vide, que $\mathcal{D}om(\sigma') \cap \mathcal{I}(\sigma') = \emptyset$ implique $\sigma'\sigma' =_{\beta\eta E} \sigma'$. □

De même, pour les filtres et unificateurs modulo une théorie E, les définitions deviennent :

Définition 24 Une substitution σ est un E-unificateur de $s =^? t$ si $\sigma(s) =_E \sigma(t)$. Une substitution σ est un E-filtre de $s \stackrel{?}{=} t$ si $\sigma(s) =_E t$. σ est un E-unificateur pour un problème d'unification S si σ unifie toutes les paires de S. De la même façon, σ est un filtre d'un problème de filtrage S si elle filtre toutes les paires de S. □

Définition 25 $\mathcal{U}_E(S)$ est l'ensemble de tous les E-unificateurs du problème S. Un ensemble complet de E-unificateurs de S, noté $\mathcal{ECU}_E(S)$ est un sous ensemble de $\mathcal{U}_E(S)$ tel que pour toute $\sigma \in \mathcal{U}_E(S)$, il existe $\theta \in \mathcal{ECU}_E(S)$ telle que $\theta \leq_{\beta\eta E} \sigma$. De même, $\mathcal{F}(S)$ est l'ensemble de

tous les E-filtres du problème S. Un ensemble complet de filtres de S, noté $\mathcal{ECF}_E(S)$ est un sous ensemble de $\mathcal{F}_E(S)$ tel que pour toute $\sigma \in \mathcal{F}_E(S)$, il existe $\theta \in \mathcal{ECF}(S)$ telle que $\theta \leq_{\beta\eta E} \sigma$. Pour simplifier, nous appellerons souvent filtres et unificateurs, les E-filtres et E-unificateurs. □

Pour une substitution $\sigma = \{\overline{X_n \mapsto t_n}\}$, nous utiliserons la notation $[\sigma]$ pour le problème d'unification $\{\overline{X_n \stackrel{?}{=} t_n}\}$.

L'intuition de l'algorithme naïf est relativement simple. Le cas de figure qui diffère du cas sans théorie, survient lorsque, pour une paire de filtrage, la tête des deux termes est un symbole de la théorie. Il s'agit alors de couper les termes au niveaux de leurs sous termes étrangers, c'est-à-dire au niveau de leurs symboles libres les plus hauts, d'abstraire les sous-termes par des variables ou des constantes, d'appeler un algorithme de filtrage dans la théorie concernée à l'ordre un, puis de recomposer les solutions. Cette méthode est symbolisée par la figure II.1.

Pour définir formellement cette méthode, nous devons introduire la notion de sous-termes étrangers maximaux.

Définition 26 L'ensemble des sous-termes étrangers maximaux d'un terme t ($\mathcal{ESEM}_E(t)$) est défini par :

$$\mathcal{ESEM}_E(t) = \begin{cases} \bigcup_{1 \leq i \leq n} \mathcal{ESEM}_E(s_i) & \text{si } t = f(\overline{s_n}), n \geq 0, f \in \mathcal{C}(E) \\ \{t\} & \text{sinon} \end{cases}$$

□

Exemple 27 Prenons le terme $t = f(X \vee Y) \vee (X \vee g(a))$ où l'opérateur $\vee$ noté de façon infixée est dans la théorie. On a alors $\mathcal{ESEM}_E(t) = \{f(X \vee Y);\ X;\ g(a)\}$ ■

Notons que les sous-termes étrangers ne sont pas composés que de symboles libres. Simplement, leur tête est un symbole libre.

Pour comprendre plus facilement la règle (E) qui est en fait le lien entre le filtrage d'ordre deux et un algorithme de E-filtrage d'ordre un, il convient de la voir comme le résumé de trois étapes.

- Le première de ces étapes consiste en une abstraction (notée ψ). ψ est une fonction qui remplace tous les sous-termes étrangers maximaux par une nouvelle variable si ce sous-terme contient des variables libres, et par une nouvelle constante sinon. Le résultat de cette étape est une paire de filtrage pure dans la théorie considérée, et constituée de termes du premier ordre.

- La deuxième étape applique à cette paire un algorithme de E-filtrage du premier ordre. Or, les solutions de ce E-filtrage ne peuvent être que de la forme $\{\overline{X_n \mapsto s_n}\}$ où les X_i sont les variables introduites lors de la première étape, puisqu'elles sont les seules variables libres du problème.

- La dernière étape consiste, pour une solution résultant du processus à l'ordre un, à reformer les paires de filtrage, avec, comme membres gauches, les sous-termes qui ont été abstraits par une variable, et comme membres droit+s, les termes clos qui leur ont été attribués par le E-filtrage d'ordre un. Dans ces derniers, les constantes introduites par l'abstraction sont remplacées par les sous termes clos abstraits (figure II.1), ce qui correspond à l'application de l'inverse de la fonction ψ.

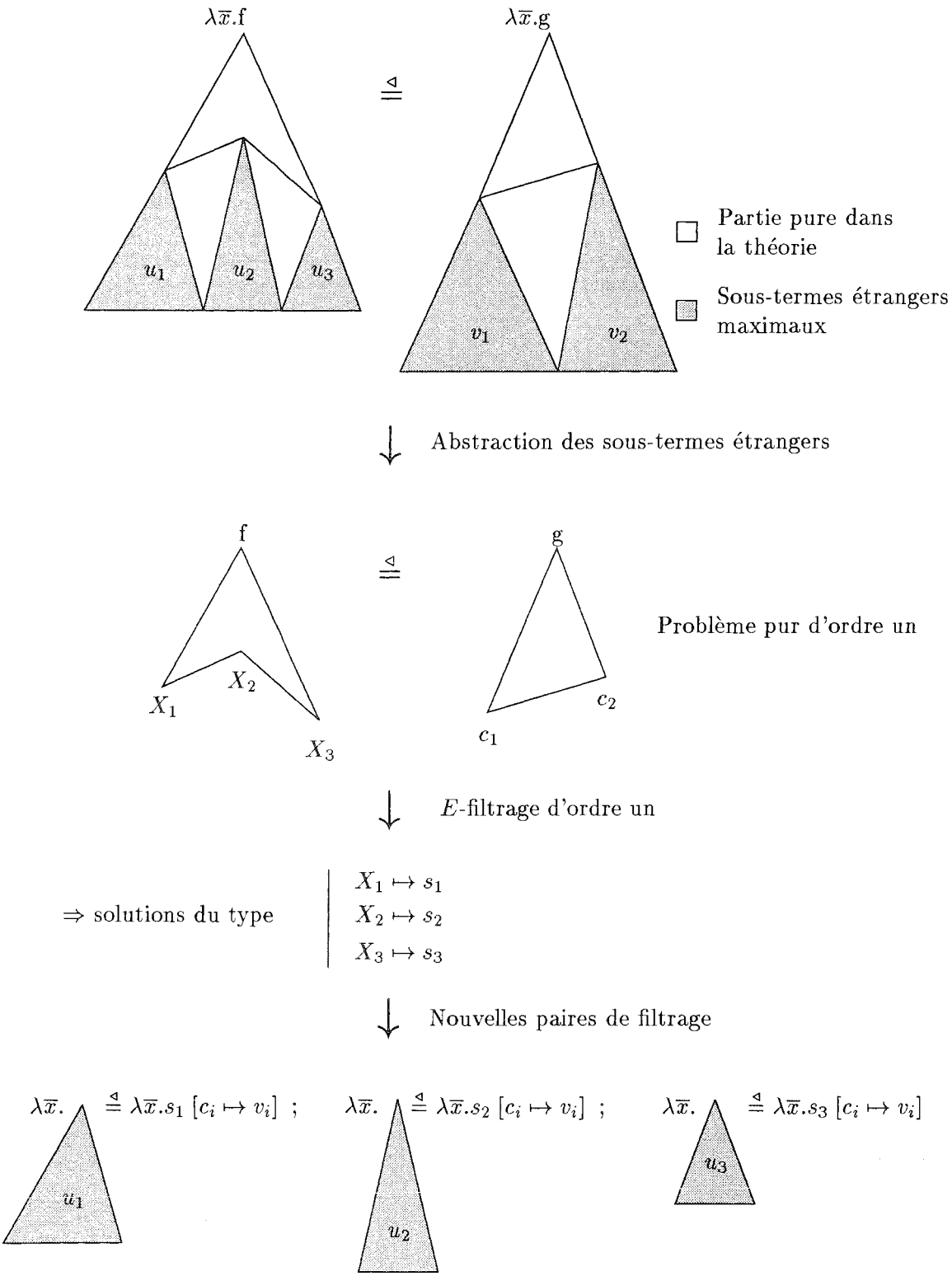


Figure II.1: Principe de la combinaison naïve

- Un point à observer particulièrement est le traitement des variables liées. En effet, celles-ci disparaissent lors de l'abstraction. Pour l'appel du filtrage équationnel du premier ordre, le lieur a également disparu. Lorsque nous reconstruisons les nouvelles paires de filtrage, nous retrouvons le lieur, et les variables liées qui ont été abstraites se retrouvent capturées par celui-ci. L'exemple 29 illustre ce mécanisme. Ensuite, il convient de remarquer que la théorie utilisée est d'ordre un, ce qui signifie que les constantes apparaissant dans cette théorie sont d'ordre deux au plus. Par conséquent, il ne peut y avoir de lieurs qu'au dessus de la tête des termes à abstraire, ou à l'intérieur des sous-termes étrangers. Ce qui veut dire que les termes donnés à l'algorithme de filtrage équationnel du premier ordre ne contient aucun lieur.

Cette méthode a été introduite dans un cadre plus général par Nipkow et Qian [NQ91]. Il s'agissait alors de définir un algorithme modulaire de E-unification d'ordre supérieur, pour des théories régulières. Leur algorithme a été amélioré par la suite par Qian et Wang [QW92] pour couvrir les théories quelconques. Mais dans le cadre du filtrage, la régularité de la théorie considérée est nécessaire. Nous allons voir pourquoi. La notion de modularité est importante ici, puisque l'algorithme de E-unification ou de E-filtrage est en fait un paramètre. La seule propriété requise est que celui-ci possède de bonnes propriétés, c'est-à-dire la correction et selon les cas, la complétude, la terminaison, ou les deux. Cette propriété de modularité donne à cet algorithme une grande puissance, puisqu'il suffit de changer l'algorithme d'ordre un pour travailler dans une autre théorie. De plus, elle permet de réutiliser les efforts, parfois très conséquents pour certaines théories, qui ont été accomplis pour donner des algorithmes utilisables à l'ordre un. Malheureusement, nous verrons par la suite que la généralité et l'efficacité s'accommodent mal, comme c'est souvent le cas.

Revenons à la règle (E). Elle se formalise ainsi :

$$\frac{< \{\lambda\overline{x}.s \stackrel{\triangleleft}{=} \lambda\overline{x}.t\}@S, \sigma >}{< \{\lambda\overline{x}.u_1 \stackrel{\triangleleft}{=} \lambda\overline{x}.\psi^{-1}(v_1), \cdots, \lambda\overline{x}.u_m \stackrel{\triangleleft}{=} \lambda\overline{x}.\psi^{-1}(v_m)\}@S, \sigma >} \quad \text{(E)}$$

si $\mathcal{H}(s) \in E$ avec

- $\{u_1, \ldots, u_n\} = \mathcal{ESEM}_E(s) \cup \mathcal{ESEM}_E(t)$ avec $u_1, \ldots, u_m$ contenant des variables libres et $u_{m+1} \ldots, u_n$ n'en contiennent pas,
- $X_1, \ldots, X_m$ sont de nouvelles variables libres telles que $X_i = X_j$ si et seulement si $u_i = u_j$.
- $c_{m+1}, \ldots, c_n$ sont de nouvelles constantes libres telles que $c_i = c_j$ si et seulement si $u_i = u_j$.
- $\psi = \{u_i \mapsto X_i \mid 1 \leq i \leq m\} \cup \{u_i \mapsto c_i \mid m+1 \leq i \leq n\}$ est une application,
- $\{\overline{X_m \mapsto v_m}\}$ est un E-filtre de $\psi(s) \stackrel{\triangleleft}{=} \psi(t)$ calculé par l'algorithme du premier ordre donné.

Exemple 28 Prenons le problème de filtrage

$$< \{(F(X,Y) \vee X) \vee g(a) \stackrel{\triangleleft}{=} g(a) \vee (b \vee (a \Leftrightarrow b))\}@S, \sigma >$$

où $\vee$ est toujours le seul symbole de la théorie. Par abstraction, nous avons donc $\psi(s) = (X_1 \vee X_2) \vee c_1$, et $\psi(t) = c_1 \vee (c_2 \vee c_3)$, par

$$\psi = \begin{cases} F(X,Y) \mapsto X_1 \\ X \mapsto X_2 \\ g(a) \mapsto c_1 \\ b \mapsto c_2 \\ a \Leftrightarrow b \mapsto c_3 \end{cases}$$

Supposons que la théorie considérée est la théorie AC. Nous avons alors à résoudre à l'ordre un le problème $\psi(s) \stackrel{?}{=} \psi(t)$. Ce travail est assuré par un algorithme de filtrage AC du premier ordre. Celui-ci va retourner, parmi les solutions, la solution suivante : $\{X_1 \mapsto c_3;\ X_2 \mapsto c_2\}$. Un des nouveaux problèmes générés par la règle (E) est alors :

$$< \{F(X,Y) \stackrel{?}{=} a \Leftrightarrow b;\ X \stackrel{?}{=} b\}@S, \sigma >$$

■

Donnons maintenant un exemple illustrant le traitement des variables liées.

Exemple 29 Soit le problème de filtrage $\{\lambda x.x + F(x) \stackrel{?}{=} \lambda z.f(z) + z\}$ où $+$ est un symbole AC. Pour le terme de gauche, les sous-termes étrangers sont x et $F(x)$. x sera donc abstrait par c_1, une nouvelle constante, puisqu'il n'y a pas de variable libre dans ce sous terme, et $F(x)$ sera abstrait par X_1. Pour le terme de droit, $f(z)$ est abstrait par c_2, puisque $\lambda x.x$ est différent de $\lambda z.f(z)$. Par contre, $\lambda x.x =_\alpha \lambda z.z$, donc z sera abstrait par c_1. Le problème livré à l'algorithme de filtrage AC d'ordre un est donc :

$$c_1 + X_1 \stackrel{?}{=} c_2 + c_1$$

L'unique solution est : $\{X_1 \mapsto c_2\}$. La nouvelle paire de filtrage est donc $\lambda x.X_1[X_1 \mapsto F(x)] \stackrel{?}{=} \lambda z.c_2[c_2 \mapsto f(z)0$, où $[X_1 \mapsto F(x), c_2 \mapsto f(z)]$ n'est pas une substitution, mais une opération de remplacement, ce qui permet au lieur de re-capturer les variables. Ainsi, on obtient la paire de filtrage :

$$\{\lambda x.F(x) \stackrel{?}{=} \lambda z.f(z)\}$$

■

Puisque la théorie E est supposée régulière, aucune variable libre ne peut être introduite par la théorie. C'est la raison pour laquelle nous imposons la régularité de la théorie. Sans cela, nous pourrions obtenir des problèmes d'unification à la place de problèmes de filtrage, or, l'unification d'ordre deux est indécidable [Gol81]. Dans la suite, $\frac{A}{B}(\mathcal{R})$ pourra être écrit $A \Longrightarrow_{(\mathcal{R})} B$.

II.3.1 Propriétés de l'algorithme naïf pour la théorie AC

Nous allons montrer les propriétés de cet algorithme pour la théorie AC. Les symboles de la théorie AC forment l'ensemble $ACop$ des opérateurs AC. L'algorithme est donc formé des règles (D), (P), (I) et (E). Notons que l'application de la règle (D) est restreinte à des symboles de tête qui ne sont pas AC. La règle (I) est l'imitation simple, puisqu'il n'existe pas de cas, dans la théorie AC, où l'on pourrait trouver $f(\overline{s}) =_{AC} g(\overline{t})$ avec f et g, deux opérateurs AC différents. D'autre part, la théorie AC n'est pas effondrante. Elles sont appliquées de façon indéterministe, suivant leurs conditions d'application. Nous allons montrer la correction, la terminaison et la complétude de cet algorithme.

A Correction et complétude

Le premier lemme concerne la correction et la complétude de la règle (D).

Lemme 30 *(Nous utilisons les notations de la règle (D)) Une substitution σ est un filtre AC de $\lambda\overline{x}.f(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_n})$, avec $f \notin ACop$ si et seulement si c'est un filtre AC de $\{\lambda\overline{x}.s_1 \stackrel{\triangleleft}{=} \lambda\overline{x}.t_1, \ldots, \lambda\overline{x}.s_n \stackrel{\triangleleft}{=} \lambda\overline{x}.t_n\}$*

Preuve Directe, par la définition de (D), en considérant qu'elle n'est appliquée que lorsque les symboles de tête sont libres. □

Montrons à présent la correction des règles (P) et (I).

Lemme 31 *(Nous utilisons les notations de la règle (P)) Si θ unifie $\{\lambda\overline{x}.s_i \stackrel{\triangleleft}{=} \lambda\overline{x}.t\}@\sigma'(S)@[\sigma'\sigma]$ avec $\sigma' = \{F \mapsto \lambda\overline{y_n}.y_i\}$, alors θ unifie $\{\lambda\overline{x}.F(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.t\}@S@[\sigma]$.*

Preuve Puisque θ unifie $[\sigma']$ et $\lambda\overline{x}.s_i \stackrel{\triangleleft}{=} \lambda\overline{x}.t$, θ unifie $\lambda\overline{x}.F(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.t$. Puisque θ unifie $[\sigma']$ et $\sigma'(S)$, θ unifie S. Enfin, puisque θ unifie $[\sigma']$ et $[\sigma'\sigma]$, θ unifie $[\sigma]$. □

Lemme 32 *(Nous utilisons les notations de la règle (I)). Si θ unifie $\{\sigma'(\lambda\overline{x}.F(\overline{s_n})) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m})\}$ $@\sigma'(S)@$ $[\sigma'\sigma]$ avec $\sigma' = \{F \mapsto \lambda\overline{y_n}.f(H_1(\overline{y_n}), \cdots, H_m(\overline{y_n}))\}$, alors θ unifie $\{\lambda\overline{x}.F(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m})\}$ $@S@$ $[\sigma]$.*

Preuve Soit $\theta \in \mathcal{U}_{AC}(\{\sigma'(\lambda\overline{x}.F(\overline{s_n})) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m})\}@\sigma'(S)@[\sigma'\sigma])$. Alors $\theta\sigma' \in \mathcal{U}_{AC}(\{\lambda\overline{x}.F(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m})\}@S@[\sigma])$. Les $H_1, \cdots, H_m$ sont de nouvelles variables.

Puisque $(\mathcal{FV}(\lambda\overline{x}.F(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m}))) \cap \mathcal{D}om(\sigma) = \{\}$, $F \notin \mathcal{D}om(\sigma)$. Par conséquent, $\theta \in \mathcal{U}_{AC}([\sigma'\sigma])$ implique que $\theta(F) =_{\beta\eta AC} \theta(\sigma'(F))$. D'où $\theta \in \mathcal{U}_{AC}([\sigma])$. Par $\theta\sigma' \in \mathcal{U}_{AC}(\{\lambda\overline{x}.F(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m})\}@S)$, nous avons $\theta \in \mathcal{U}_{AC}(\{\lambda\overline{x}.F(\overline{s_n}) \stackrel{\triangleleft}{=} \lambda\overline{x}.f(\overline{t_m})\}@S@[\sigma])$. □

Il reste à montrer la correction de la règle (E) appliquée à la théorie AC.

Lemme 33 *(Nous utilisons les notations de la règle (E)). Si θ est un filtre AC de $\{\lambda\overline{x}.u_n \stackrel{\triangleleft}{=} \lambda\overline{x}.\psi^{-1}v_n\}$ tel qu'il existe $\sigma_{AC} = \{\overline{X_n \mapsto v_n}\}$ qui est un filtre AC de $\psi(s) \stackrel{\triangleleft}{=} \psi(t)$, alors θ est un filtre AC de $\{\lambda\overline{x}.s \stackrel{\triangleleft}{=} \lambda\overline{x}.t\}$.*

Preuve Du fait que σ_{AC} est un filtre AC de $\psi(s) \stackrel{\triangleleft}{=} \psi(t)$, nous savons que $\sigma_{AC}(\psi(s)) =_{AC} \psi(t)$. Or, en utilisant la notation $w[u \mapsto v]$ pour signifier le remplacement du terme u par v dans w, nous avons globalement $\sigma_{AC}(\psi(s)) = s[\overline{u_n \mapsto v_n}] =_{AC} \psi(t)$. Et, puisque $\sigma_{AC}(\psi(s)) =_{AC} \psi(t)$, $\psi^{-1}(\psi(s)) =_{AC} \psi^{-1}(\psi(t)) = t$. Or, $\psi^{-1}(\lambda\overline{x}.s[\overline{u_n \mapsto v_n}]) = \lambda\overline{x}.s[\overline{u_n \mapsto \psi^{-1}(v_n)}]$ puisque les u_n sont les sous-termes étrangers maximaux de s. D'autre part, nous avons l'hypothèse selon laquelle θ est telle que $\theta u_i =_{AC} \psi^{-1}(v_i)$ pour $i = 1, \ldots, n$. Donc, $\theta(\lambda\overline{x}.s) =_{AC} s[\overline{u_n \mapsto \psi^{-1}v_n}] =_{AC} \lambda\overline{x}.t$. □

Par extension, nous pouvons formuler le théorème de correction de l'algorithme naïf.

Théorème 34 *Si $< S_0, \{\} > \Rightarrow^* < \{\}, \sigma_f >$ est un séquence de transformations par les règles (D), (P), (I) et (E) (avec E=AC), σ_f est un filtre AC de S_0.*

Intéressons-nous maintenant à la complétude, à commencer par le cas flexible-rigide.

Lemme 35 *Soit θ, un filtre AC de $\{\lambda\overline{x}.F(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_m})\}$, alors il existe une règle R ((P) ou (I)) telle que*

$$< \lambda\overline{x}.F(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_m})\}, \{\} > \Longrightarrow_R < S', \sigma' >$$

telle qu'il existe θ', un filtre AC de S' avec $\theta'\sigma' =_{AC} \theta[\mathcal{FV}(\lambda\overline{x}.F(\overline{s_n}))]$.

Preuve Si $\theta(F) = \lambda\overline{y_n}.y_i$. Alors R =(P), $S' = \{\lambda\overline{x}.s_i \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_m})\}$ et $\sigma' = \{F \mapsto \lambda\overline{y_n}.y_i\}$. D'où, $\theta' = \theta(F) \cup \theta''$ où θ'' est un filtre AC de S' tel que $\theta''\theta' =_{AC} \theta[\mathcal{FV}(\lambda\overline{x}.F(\overline{s_n}))]$.
Si $\theta(F) = \lambda\overline{y_n}.f(\overline{t'_m})$. Alors, $\theta(S) = \lambda\overline{x}.f(\theta(\overline{t'_m}))$. Prenons R =(I). Soit $s'_i = \lambda\overline{y_n}, \overline{w_{k_i}}.H_i(\overline{y_n}, olw_{k_i})$ pour $i = 1, \ldots, m$, un terme du type de t_i, avec H_i une nouvelle variable libre. Par (I), nous avons $\sigma' = \{F \mapsto \lambda\overline{y_n}.f(\overline{s'_m})\}$. Puisque $\lambda\overline{x}.f(\theta(\overline{t'_m})) = \theta(\lambda\overline{x}.F(\overline{s_n}))$, il existe θ' telle que $\theta'(s'_i) =_{AC} t'_i$ pour $i = 1, \ldots, m$. Donc, il existe θ' telle que θ' est un filtre AC de S' avec $\theta'\sigma' =_{AC} \theta[\mathcal{FV}(\lambda\overline{x}.F(\overline{s_n}))]$. □

Pour la complétude de la règle (E), nous avons :

Lemme 36 *Soit θ un filtre AC de $\{\lambda\overline{x}.f(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_n})\}$ avec $f \in AC op$. Alors, par application de la règle (E), il existe une dérivation telle qu'il existe θ', un filtre AC de $\{\lambda\overline{x}.u_n \stackrel{?}{=} \lambda\overline{x}.\psi^{-1}v_n\}$ tel que $\theta' =_{AC} \theta[\mathcal{FV}(\overline{s_n})]$.*

Preuve $\mathcal{D}om(\theta) = \mathcal{FV}(\overline{u_k})$ où $\{\overline{u_k}\} = \mathcal{ESEM}_{AC}(\lambda\overline{x}.f(\overline{s_n}))$. Soit $\sigma'_{AC} = \{\overline{X_k \mapsto w_k}\}$ avec $w_j = \theta(u_j)$ pour $j = 1, \ldots, k$. θ est telle que $\lambda\overline{x}.\theta(u_j) = \lambda\overline{x}.w_j$. Or, par complétude de l'algorithme de filtrage AC d'ordre un, $\exists\, \sigma_{AC}$ solution de $\psi(s) \stackrel{?}{=} \psi(t)$ telle que $\sigma_{AC} = \{\overline{X_k \mapsto v_k}\}$ avec $w_i =_{AC} \psi^{-1}(v_i)$ pour $i = 1, \ldots, k$. Par conséquent, θ est elle même solution de $\{\lambda\overline{x}.u_n \stackrel{?}{=} \lambda\overline{x}.\psi^{-1}v_n\}$. □

Par extension, nous pouvons formuler le théorème de complétude de l'algorithme.

Théorème 37 *Soit θ, un filtre AC de S, il existe une séquence de transformations par les règles (D), (P), (I) et (E) telle que*

$$< S, \{\} > \Longrightarrow^* < \{\}, \sigma_f >$$

telle que $\sigma_f =_{AC} \theta[\mathcal{FV}(S)]$.

Maintenant que nous nous sommes assurés que toutes les séquences produites par cet algorithme sont correctes et qu'il est capable de calculer un ensemble complet de solutions, nous allons montrer que toute séquence de transformation par ces règles est finie.

B Terminaison

Pour le cas AC, la terminaison ne pose pas de problème majeur. Le fait que la théorie AC est finitaire revient à dire que l'algorithme de filtrage d'ordre un rendra un nombre fini de solution, et en un temps fini. C'est bien sûr, une condition nécessaire pour la terminaison de l'algorithme global. Nous pouvons utiliser la même mesure de complexité que dans le cas syntaxique pour montrer la terminaison [HL78]. Soient ξ_1 le multi-ensemble de la taille des termes membres gauches d'un problème S et ξ_2 le multi-ensemble de la taille des termes membres droits. Pour nous, la taille d'un termes est simplement le nombre des symboles qui le composent. La mesure de complexité est définie par

$$C(S) = < \xi_2(S), \xi_1(S) >$$

L'ordre utilisé est l'extension de l'ordre sur les entiers aux multi-ensembles. Nous allons voir que cette mesure décroît strictement par application des règles énoncées.

Lemme 38 *Soit S un problème de filtrage. Toute séquence d'application des règles (D), (P), (I) et (E) fait décroître strictement la complexité $C(S)$.*

Preuve

- (D) : ôte la tête des termes. par conséquent, $\xi_1(S)$ et $\xi_2(S)$ décroissent strictement.
- (E) : le théorie AC est permutative (si $s =_{AC} t$, t peut être obtenu de s en permutant les arguments des symboles AC), ce qui implique que $s =_{AC} t \Rightarrow |s|=|t|$. Donc, pour σ, solution du problème $\psi(s) \stackrel{?}{=} \psi(t)$ délivrée par l'algorithme du premier ordre, on a toujours $|\sigma(X)|<|\psi(t)|$, puisque $\sigma(psi(s)) =_{AC} \psi(t)$. On a donc $|\sigma(\psi(s))|=|\psi(t)|$, et $|\psi^{-1}(\sigma(\psi(s)))|=|\psi^{-1}\psi(t)|$. D'où $|\psi^{-1}(\sigma(X))|<|\psi^{-1}\psi(t)|$. Or, $\psi^{-1}\psi(t) = t$, on a donc bien, $|\psi^{-1}(v_i)|<|t|$. Donc, ξ_2 décroît strictement.
- (P) : ξ_2 n'est pas modifié, alors que ξ_1 décroît strictement puisque le membre gauche est remplacé par l'un de ses sous-termes strictes.
- (I) : après application de cette règle, ξ_2 n'est pas modifié, mais ξ_1 peut avoir augmenté. Mais la règle suivante ne peut être que (D) ou (E), qui font toutes deux décroître strictement ξ_2. Globalement, ξ_2 décroît donc strictement et $C(S)$ avec lui.

□

Ceci nous permet de conclure par le théorème de terminaison.

Théorème 39 *Pour tout problème de filtrage S, toute séquence d'application des règles (D), (P), (I) et (E) est finie.*

II.3.2 Discussion sur la théorie utilisée

Nous allons discuter ici les raisons qui nous ont poussés à nous intéresser à la théorie AC, qui est un cas particulier de théorie régulière. Tout d'abord, précisons que pour le cas général de combinaison d'une théorie et du filtrage d'ordre deux, la règle d'imitation doit être légèrement modifiée. En effet, pour le filtrage syntaxique, comme pour le cas AC, lorsque deux termes sont équivalents $s = t$ tels que leur tête est une constante (ou $s =_{AC} t$ tels que leur tête est un opérateur AC), nous avons toujours $\mathcal{H}(s) = \mathcal{H}(t)$. Ceci n'est pas vrai dans le cas général. La théorie considérée peut contenir des axiomes du type $f(x) \simeq g(x)$. Auquel cas, toute imitation de la fonction f doit être doublé d'une imitation du symbole g. Sinon, le processus ne saurait être complet. La règle d'imitation devient alors :

$$\frac{< \{\lambda\overline{x}.F(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_m})\}@S, \sigma >}{< \{\sigma'(\lambda\overline{x}.F(\overline{s_n})) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_m})\}@\sigma'(S), \sigma'\sigma >} \quad (I)$$

avec $\sigma' = \{F \mapsto \lambda\overline{y_n}.g(H_1(\overline{y_n}), \cdots, H_k(\overline{y_n}))\}$ où k est l'arité de la fonction g. Cette imitation doit être appliquée pour tout fonction $g \in CS_E(f)$ où $\mathcal{CS}_E(F)$ est défini de la façon suivante.

Définition 40 Pour une théorie E, la relation de E-compatibilité $\sim_E$ est définie par la plus petite relation d'équivalence sur $\mathcal{C}$ telle que $\mathcal{H}(l) \sim_E \mathcal{H}(r)$ pour toute équation $l \simeq r \in E$ où $\mathcal{H}(l)$ et $\mathcal{H}(r)$ sont tous deux des symboles de fonction. L'ensemble des symboles E-compatibles d'une fonction f est alors $CS_E(f) = \{g \in \mathcal{C} \mid f \sim_E g\}$. Cet ensemble est toujours fini. □

Toujours pour les théories régulières, il existe un autre cas de figure où il est nécessaire de multiplier les imitations. Ce cas de figure apparaît pour les théories effondrantes, c'est-à-dire les théories dans lesquelles un terme non variable peut être E-équivalent à une variable. Prenons un exemple. Soit la théorie $E = \{f(g(X)) \simeq X\}$. En prenant le problème de filtrage $\{F(Y) \stackrel{?}{=} c\}$, la solution donnée par $\{F \mapsto \lambda x.f(x);\ Y \mapsto g(c)\}$ ne peut être obtenue par le processus décrit. Pour être capables de calculer de telles solutions, il faut tenir compte du fait que l'axiome de la théorie est effondrant – i.e. l'un de ses membres est réduit à une variable –. La solution consiste à chaque fois qu'une imitation est calculée, à la doubler d'une imitation[1] de f qui est la tête de cet axiome effondrant. Ainsi, l'algorithme redevient complet. Mais nous ne sommes pas au bout de nos peines. Notre but reste d'obtenir un algorithme complet qui termine dans tous les cas. Or, il existe des théories finitaires d'ordre un telles que leur combinaison avec le filtrage d'ordre deux entraîne une infinité de solutions différentes dans certains cas. En effet, il suffit de considérer la théorie $E = \{f(1) \simeq 1\}$ et le problème de filtrage $\{F(1) \stackrel{?}{=} 1\}$. les solutions $\{F \mapsto \lambda x.x\}$ et $\{F \mapsto \lambda x.f(x)\}$ ne forment pas un ensemble complet de solutions, puisque $\{F \mapsto \lambda x.f^n(x)\}$ est également une solution de ce problème et n'est pas comparable modulo E à $\{F \mapsto \lambda x.f(x)\}$ lorsque $n > 1$. En conséquence, ce problème est infinitaire, et on ne peut prétendre à un algorithme complet qui termine pour en calculer les solutions.

En résumé, les théories telles que l'on soit obligés de considérer tous les symboles de l'ensemble $CS_E(f)$ à chaque imitation de f, et les théories effondrantes, lorsqu'elles permettent un algorithme complet, engendrent un accroissement très sensible de l'espace de recherche des solutions. Ne perdons pas de vue qu'il s'agit pour nous de construire un outil, donc d'obtenir une méthode utilisable. D'un autre coté, nous désirons rendre compte des propriétés algébriques des connecteurs logiques. La théorie AC semble alors être un bon compromis. Cette théorie possède bon nombre d'avantages. Elle est suffisamment puissante pour accroître considérablement les possibilités du filtrage d'ordre deux, et possède suffisamment de bonnes propriétés pour être traitée de façon raisonnable. Rappelons que le filtrage modulo AC est déjà un problème NP-complet à l'ordre un [BKN87]. C'est pour toutes ces raisons que nous nous sommes "restreints" à la théorie AC, bien que dans notre esprit, il s'agisse plus d'un choix raisonnable que d'une restriction. De plus un avantage supplémentaire de cette théorie est qu'elle a déjà été largement étudiée.

A partir de l'algorithme déjà donné, appliqué à cette théorie, nous allons voir les problèmes d'efficacité auxquels nous avons à faire face. Ce qui nous a menés à proposer un nouvel algorithme de filtrage AC d'ordre deux qui évite beaucoup de travail inutile, en essayant de voir le filtrage d'ordre deux d'une autre façon, et en y incorporant, ce que l'on pourrait appeler l'"esprit AC" – en particulier en utilisant des termes aplatis. Pour réaliser ce travail, nous avons eu la chance de travailler en collaboration avec Zhenyu Qian.

II.3.3 Problèmes d'efficacité de l'algorithme classique

L'algorithme issu de l'unification modulo une théorie régulière énoncée par Nipkow et Qian [NQ91] est très général. Ce qui implique malheureusement qu'il peut se montrer inefficace pour le cadre particulier dans lequel nous envisageons de l'utiliser. Comme il est indiqué dans [Cur93,

1 Le terme d'imitation ne convient plus vraiment, puisqu'il s'agit d'explorer un autre symbole que la tête du terme rigide. Toutefois, nous conservons ce terme, puisqu'il fait référence à la même opération.

MW94], un des principaux problèmes de cet algorithme est qu'il génère beaucoup de solutions inutiles. Nous l'avons implanté, et nous avons pu le constater, comme sur l'exemple qui suit.

Exemple 41 Considérons le problème de filtrage $F(X) \stackrel{?}{=} a + b + c$ où le symbole $+$ est AC. Ce problème possède 8 solutions différentes qui sont :

$F \mapsto \lambda x.a + b + c$
$F \mapsto \lambda x.x + b + c \; ; \; X \mapsto a$
$F \mapsto \lambda x.x + a + c \; ; \; X \mapsto b$
$F \mapsto \lambda x.x + a + b \; ; \; X \mapsto c$
$F \mapsto \lambda x.x + a \; ; \; X \mapsto b + c$
$F \mapsto \lambda x.x + b \; ; \; X \mapsto a + c$
$F \mapsto \lambda x.x + c \; ; \; X \mapsto a + b$
$F \mapsto \lambda x.x \; ; \; X \mapsto a + b + c$

L'algorithme naïf en calcul 55! ■

Essayons de comprendre les raisons de ce décalage. La première étape consistant en une imitation du symbole $+$ produit la substitution

$$\{F \mapsto \lambda x.H_1(x) + H_2(x)\}$$

Ainsi, nous obtenons le nouveau problème $\{H_1(X)+H_2(X) \stackrel{?}{=} a+b+c\}$. Par abstraction, le terme de gauche est transformé en un terme AC pure pour obtenir la paire : $X_1+X_2 \stackrel{?}{=} a+b+c$. Comme décrit plus haut, ceci déclenche l'appel d'un algorithme de filtrage AC d'ordre un. Celui-ci, qui est supposé complet, va produire au moins six solutions. Considérons en trois parmi elles :

1. $\{X_1 \mapsto a, X_2 \mapsto b + c\}$, qui donne le problème $\{H_1(X) \stackrel{?}{=} a, H_2 \stackrel{?}{=} b + c\}$,

2. $\{X_1 \mapsto b, X_2 \mapsto c + a\}$, qui donne le problème $\{H_1(X) \stackrel{?}{=} b, H_2 \stackrel{?}{=} c + a\}$,

3. $\{X_1 \mapsto b + c, X_2 \mapsto a\}$, qui donne $\{H_1(X) \stackrel{?}{=} b + c, H_2 \stackrel{?}{=} a\}$.

Il est clair que dans les cas 1 et 3, le problème généré est le même. Ceci est simplement dû au fait que les solutions d'ordre un sont symétriques. Les solutions seront donc dupliquées. Une meilleure prise en compte de la théorie AC devrait permettre de remédier à cela. Cet exemple illustre un problème émanant de la théorie AC, mais nous pouvons également imputer le même type de redondances à la combinaison du filtrage d'ordre deux avec une théorie quelconque. En effet, dans la suite du processus, nous aurons, comme solution du problème 1 $\{H_1 \mapsto \lambda x.a, H_2 \mapsto \lambda x.b+c\}$, ce qui donne, globalement, comme solution : $\{F \mapsto \lambda x.a+b+c\}$. Cette solution va être dupliquée autant de fois qu'il y a de solutions générées par l'algorithme d'ordre un. Enfin, pour en finir avec cet exemple, tous les problèmes du type $H_1(X) \stackrel{?}{=} b + c$ vont à leur tour engendrer un appel de l'algorithme d'ordre un, donc générer de nouvelles redondances, etc... Il est donc facile, en définitive, d'imaginer comment il est possible de passer de 8 solutions, théoriquement suffisantes, à 55 solutions calculées en pratique. Nous avons donc décidé d'adopter un autre point de vue sur le filtrage d'ordre deux. Ceci pour nous permettre de mieux combiner l'ordre deux et le processus AC. En particulier, nous allons voir que cela nous permet de définir un algorithme utilisable aussi bien pour la théorie vide que pour le cas AC, et qui évite la plupart des redondances.

II.4 Algorithme efficace pour le filtrage AC d'ordre deux

Forts des constatations qui précèdent, nous proposons un nouvel algorithme de filtrage d'ordre deux et modulo la théorie AC. Cet algorithme s'applique également à la théorie vide, sans avoir à être modifié. L'approche que nous adoptons ici nous semble plus proche de l'intuition. Par contre, elle est moins générale, puisqu'elle s'appuie sur des concepts relatifs à la théorie AC, comme les termes aplatis par exemple. En quelques mots, les atouts de cet algorithme sont les suivants :

- La plupart des redondances sont évitées.
- Grâce à la réutilisation de problèmes déjà résolus, beaucoup de solutions ne sont pas calculées par filtrage, mais construites à partir d'autres solutions.
- Aucune nouvelle variable libre d'ordre deux n'est introduite au cours du processus.
- L'algorithme est utilisable tel quel pour la théorie vide.

L'idée générale est la suivante. Les règles d'ordre un de l'algorithme classique, c'est-à-dire (D) et (E), sont conservées. En effet, le problème d'efficacité se situe à l'ordre deux. Nous allons donc introduire un mécanisme spécifique pour calculer les solutions θ d'une paire de filtrage $\{\lambda\overline{x}.F(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_m})\}$. Or, pour une solution θ, si l'on veut montrer que $\theta(\lambda\overline{x}.F(\overline{s_n})) =_{AC} \lambda\overline{x}.f(\overline{t_m})$, certains arguments parmi les $\theta(s_1), \ldots, \theta(s_n)$ doivent être AC-équivalents à des sous-termes de $\lambda\overline{x}.f(\overline{t_m})$. Le principe de ce mécanisme consiste donc à tenter de filtrer chacun des arguments de F avec tous les sous-termes de $\lambda\overline{x}.f(\overline{t_m})$, puis, à partir des solutions obtenues, à construire les solutions pour F.

D'autre part, il s'avère très utile de regarder de près les occurrences des variables liées. En effet, pour deux termes équivalents $\lambda\overline{x_n}.s = \lambda\overline{x_n}.t$, toutes les variables liées x_i apparaissant dans t apparaissent également dans s. Il en est de même pour l'équivalence modulo AC, puisque cette théorie est dite "permutative", c'est-à-dire qu'elle ne fait que permuter les symboles, mais le multi-ensemble des symboles est le même pour les deux membres de l'égalité. Nous pouvons donc affirmer, dans le cas général, que pour

$$\lambda\overline{x_n}.s =_{AC} \lambda\overline{x_n}.t$$

l'ensemble des x_i qui apparaissent dans s et le même que l'ensemble des x_i apparaissant dans t. Nous allons tirer parti de cette constatation après l'introduction de quelques définitions dont nous allons avoir besoin. Rappelons d'abord que la théorie AC est définie par :

$$\begin{aligned}(X+Y)+Z &\simeq X+(Y+Z)\\ X+Y &\simeq Y+X\end{aligned}$$

Nous noterons $ACop$ l'ensemble des opérateurs AC.

Définition 42 Pour un terme $s = \lambda\overline{x_n}.t$, $UBV(s)$, est l'ensemble des variables x_i apparaissant dans t (ou variables liées utiles). □

Définition 43 Deux substitutions σ_1 et σ_2 sont *compatibles* si et seulement si elles vérifient la condition suivante :

$$\forall x \in (\mathcal{D}om(\sigma_1) \cap \mathcal{D}om(\sigma_2)),\ \ \sigma_1(x) =_{AC} \sigma_2(x)$$

□

Définition 44 Une *Position* est une suite d'entiers. L'ensemble des positions d'un λ-terme t est noté $\mathcal{P}os(t)$. Soient p et q des positions et Λ la suite vide dénotant la racine. On définit $t_{|p}$, le sous-terme de t à la position p par :

$$\begin{aligned} t_{|\Lambda} &= t \\ (\lambda x.t)_{|1.p} &= t_{|p} \\ a(t_1,\ldots,t_n)_{|k.p} &= (t_k)_{|p} \end{aligned}$$

□

Définition 45 Une position p est *au dessus de* q (noté $p \leq q$) si p est un préfixe de q. Lorsque $p \not\leq q$ et $q \not\leq p$, les positions p et q sont dites *indépendantes*. De la même façon, $t_{|p_1}$ et $t_{|p_2}$ sont deux sous-termes *indépendants* de t si p_1 et p_2 sont deux positions indépendantes de t. □

Définition 46 Un ensemble de positions indépendantes d'un terme t est appelé *coupure* de t. L'ensemble de toutes les coupures de t est noté $\mathcal{C}oup(t)$. D'autre part, pour un terme t et une coupure $c = \{p_1,\ldots,p_n\} \in \mathcal{C}oup(t)$, et des λ-termes $s_1,\ldots,s_n$ tels que $\tau(t_{|p_i}) = \tau(s_i)$ pour $i = 1,\ldots,n$, $t[s_1,\cdots,s_n]_{p_1,\cdots,p_n}$ est le terme t où chaque $t_{|p_i}$ a été remplacé par s_i. De plus, les symboles $\Box_1,\cdots,\Box_k$ représentent des "trous" dans le terme t et $t[\Box_1,\cdots,\Box_k]_{p_1,\cdots,p_k}$ est appelé contexte. □

Nous allons utiliser une notion classique dans le domaine AC, qui est celle des termes aplatis. La forme aplatie d'un terme est une représentation. Il faut pour cela considérer les opérateurs AC comme ayant une arité variable (≥ 2). Ainsi, en prenant $\wedge$ comme opérateur AC, la forme aplatie de $a \wedge b \wedge c$ est $\wedge(a,b,c)$ les sous-termes a, b et c étant eux-mêmes aplatis. Nous pouvons donner la définition formelle suivante, avec $flat$, la fonction d'aplatissement.

$$\begin{aligned} flat(t) &= t && \text{si } t \text{ est un symbole du premier ordre} \\ flat(\lambda x.t) &= \lambda x.flat(t) && \\ flat(f(t_1,\ldots,t_n)) &= f(flat(t_1),\ldots,(flat(t_n))) && \text{lorsque } f \notin ACop \\ flat(t) &= f(flat(t_1),\ldots,(flat(t_n))) && \text{lorsque } t =_{AC} f(t_1, f(t_2,\cdots f(t_{n-1},t_n)\cdots)) \\ & && \text{et } f \in ACop \text{ et } \mathcal{H}(t_i) \neq f \end{aligned}$$

De plus, pour un terme aplati $s = f(\overline{t_m})$ où $f \in ACop$, si $n = 1$, alors $s = t_1$. Notons qu'un terme aplati représente plusieurs λ-termes AC-équivalents. Nous allons à présent introduire une notion de position mieux adaptée au cas AC que la définition classique.

Définition 47 Une *position aplatie* est une suite $p \cdot I$, où p est une position aplatie, et I un ensemble non-vide d'entiers. Lorsque $I = \{k\}$, nous pourrons écrire k plutôt que I. La notion de position aplatie subsume celle de position. Nous utiliserons p et q pour dénoter des positions aplaties. Nous définissons un *sous-terme aplati* d'un terme aplati à une position aplatie de la façon suivante, avec Λ la position à la racine :

$$\begin{aligned} t_{|\Lambda} &= t \\ (\lambda x.t)_{|1.p} &= t_{|p}, \\ a(t_1,\cdots,t_n)_{|k.p} &= (t_k)_{|p} && a \notin ACop, 1 \leq k \leq n \\ f(t_1,\cdots,t_n)_{|\{k_1,\cdots,k_m\}.p} &= f(t_{k_1},\cdots,t_{k_m})_{|p} && f \in ACop, \{k_1,\cdots,k_m\} \subseteq \{1,\cdots,n\}, \end{aligned}$$

□

Par exemple, $h(+(a,b,c))_{|1.\{1,3\}} = a + c$, et $h(+(a,b,c))_{|1.2} = b$.

Définition 48 Une position aplatie p_1 de la forme $q \cdot I$ est *au dessus* d'une position aplatie p_2 (noté $p_1 \leq p_2$) si p_2 est de la forme $q \cdot J$ avec $J \subseteq I$ ou de la forme $q \cdot k \cdot p_3$ avec $k \in I$ pour une position aplatie p_3. Deux positions aplaties sont dites *indépendantes* si aucune n'est au-dessus de l'autre ou si elles sont de la forme $q \cdot I$ et $q \cdot J$ avec $I \cap J = \{\}$. Deux sous-termes à des positions aplaties indépendantes sont dit *indépendants.* □

Définition 49 Un ensemble de positions indépendantes d'un terme t est appelé coupure de t. L'ensemble de toutes les coupures de t est noté $Coup(t)$. Les coupures réduites à un singleton sont dites *élémentaires.* □

Notons qu'une coupure ne concerne pas nécessairement toutes les branches d'un terme. Elle peut par exemple n'en couper qu'une seule.

Nous avons indiqué plus haut que par l'étude des variables liées réellement utilisées dans les termes, nous avons des indications quant à la forme des solutions. Supposons par exemple que pour la paire de filtrage $\{\lambda\overline{x_n}.F(\overline{s_m}) \stackrel{?}{=} \lambda\overline{x_n}.f(\overline{t_k})\}$, $UBV(t)$ (où t est le terme $\lambda\overline{x_n}.f(\overline{t_k})$) est réduit au singleton $\{x_i\}$. Puisque, pour deux termes équivalents modulo AC, les ensembles UBV doivent être les mêmes pour les deux termes, toutes les solutions σ de ce filtrage doivent être telles que :

$$UBV(\sigma(\lambda\overline{x_n}.F(\overline{s_m}))) = UBV(t)$$

Or, quelle que soit σ, pour F, elle est de la forme $\sigma(F) =_{AC} \lambda\overline{y_m}.t[y_{j_1}, \ldots, y_{j_l}]_c$ où c est une coupure de t. Par conséquent, par application de σ, et β-réduction, on obtient :

$$\sigma(\lambda\overline{x_n}.F(\overline{s_m})) \rightarrow_\beta \lambda\overline{x_n}.t[\sigma(s_{j_1}), \ldots, \sigma(s_{j_l})]_c$$

Les variables contenues dans $UBV(t)$ doivent apparaître dans $\sigma(s_{j_1}), \ldots, \sigma(s_{j_l})$. Dans notre cas, cela signifie que l'un des $\sigma(s_{j_l})$ contient x_i; ce qui implique, si l'on applique récursivement la condition sur les variables liées, qu'il existe s_{j_i} contenant x_i. En conséquence, toute solution de ce problème est telle que $y_{j_i} \in \sigma(F)$ avec s_{j_i} contenant x_i. De plus, la position p_{j_i} qui est associée à y_{j_i} dans $\sigma(F)$ doit être telle que si p_i est la position de x_i dans t, $p_{j_i} \leq p_i$. C'est le sens du lemme 56. L'algorithme que nous allons présenter tient d'abord compte de cette propriété, puisqu'elle permet d'éliminer un grand nombre de recherches de solutions inutiles. Illustrons ceci sur quelques exemples très simples.

Exemple 50

- La paire de filtrage $\lambda x.F(t) \stackrel{?}{=} \lambda x.f(x)$ n'a pas de solution si t ne contient pas x.
- Pour la paire $\lambda x.F(G(x),t) \stackrel{?}{=} \lambda x.f(a,x)$, toutes les solutions pour F seront du type $F \mapsto \lambda\overline{y_2}.t[y_1]_p$ avec $p = \Lambda$ ou $p = \{2\}$. Ce qui donne respectivement les deux solutions pour F :
  - $\{F \mapsto \lambda\overline{y_2}.y_1\}$ avec les solutions de $\{\lambda x.G(x) \stackrel{?}{=} \lambda x.f(a,x)\}$,
  - $\{F \mapsto \lambda\overline{y_2}.f(a,y_1)\}$ avec les solutions – s'il y en a – de $\{\lambda x.G(x) \stackrel{?}{=} \lambda x.x;\ \lambda x.t \stackrel{?}{=} \lambda x.a\}$.

- Pour la paire $\{\lambda x.F(t,x) \stackrel{?}{=} \lambda x.f(x,x)\}$, (où x n'apparaît pas dans t) il y a cette fois-ci deux variables liées obligatoires dans $\sigma(F)$ qui sont toutes les deux y_2, puisque c'est dans le second argument de F que la variable liée x apparaît. D'où l'unique solution pour F : $\{F \mapsto \lambda\overline{y_2}.f(y_2,y_2)\}$.

■

L'algorithme proposé réutilise les règles (E) appliquée à AC et (D). Ces règles sont celles qui traitent le premier ordre. Mais un algorithme spécifique est introduit pour le cas flexible-rigide. Nous allons formuler l'algorithme de la façon suivante : d'une part l'algorithme général, séparant les cas rigide-rigide et flexible-rigide, puis la fonction F-R qui traite le cas flexible-rigide. Ce n'est pas répété dans l'algorithme, mais toutes les positions considérés sont des positions aplaties.

Algorithme de filtrage

- Cas rigide-rigide : application, selon le cas de la règle (E) avec la théorie AC, ou de la règle (D).
- Cas cas flexible-rigide :

$$\frac{< \{\lambda\overline{x}.F(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_m})\}@S, \sigma >}{< \sigma'(S), \sigma'\sigma >} \qquad \text{(flex-rig)}$$

pour toutes les solutions σ' calculées par la fonction F-R appliquée à

$$< \{\lambda\overline{x}.F(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.f(\overline{t_m})\}, \{\} >$$

Fonction F-R Pour $< \{\lambda\overline{x_n}.F(\overline{s_m}) \stackrel{?}{=} \lambda\overline{x_n}.f(\overline{t_k})\}, \{\} >$,

(Nous utilisons s pour le terme $F(\overline{s_m})$ et t pour le terme $f(\overline{t_k})$ toutes les substitutions notées σ' sont des résultats de cette fonction)

0- Si $UBV(\lambda\overline{x_n}.t) \not\subseteq UBV(\lambda\overline{x_n}.s)$, alors, il n'y a pas de solution. Sinon, si $UBV(\lambda\overline{x_n}.t) = \emptyset$, alors $\sigma' = \{F \mapsto \lambda\overline{y_m}.t\}$ est solution et l'étape 1 n'est pas utile. Sinon, faire l'étape 1.

1- Pour toutes les occurrences à une position p_i dans t des x_i, éléments de $UBV(t)$, résoudre tous les problèmes $< \{\lambda\overline{x_n}.s_j \stackrel{?}{=} \lambda\overline{x_n}.t_{|p}\}, \{\} >$ (les N solutions d'un tel problème sont notées $\theta_j^1, \ldots, \theta_j^N$).
où

- s_j est un argument de F où x_i apparaît,
- et p est une position de t telle que $p \leq p_i$.

S'il n'y a pas au moins une solution pour chaque x_i, il n'y a pas de solution. Sinon, faire l'étape 2.

2- Résoudre tous les problèmes $< \{\lambda\overline{x_n}.s_r \stackrel{?}{=} \lambda\overline{x_n}.t_{|q}\}, \{\} >$ tels que q est une position indépendante de tous les p_i précédemment considérés et pour $r = 1, \ldots, m$ (les M solutions d'un tel problème sont notées $\theta_r^1, \ldots, \theta_r^M$).

3- (Construction des solutions pour F)

Construire toutes les substitutions σ' telles que :

$$\sigma' = \cup_{\alpha=1}^{P} \theta_\alpha^K \cup \{F \mapsto \lambda y_m.t[y_{i_1}, \ldots, y_{i_P}]_c\}$$

avec

- $\forall x_i \in UBV(t)$, il existe i_j tel que $y_{i_j} \in UBV(\sigma'(F))$ et x_i apparaît dans s_{i_j}.
- θ_α^K est une solution de $\{\lambda\overline{x_n}.s_\alpha \stackrel{\triangleleft}{=} \lambda\overline{x_n}.t_{|q_{i_j}}\}$ pour $c = \{q_{i_1}, \ldots, q_{i_P}\}$, une coupure de t,
- tous les θ_α^K sont compatibles entre eux et avec $\sigma'[\{F\}]$.

Déroulons cet algorithme sur quelques exemples.

Exemple 51 Premier exemple :

$$\{\lambda\overline{x_2}.F(x_1, x_2, a) \stackrel{\triangleleft}{=} \lambda\overline{x_2}.f(x_1, g(x_1), a)\}$$

où f n'est pas un symbole AC.

0- $UBV(\lambda\overline{x_2}.t) = \{x_1\} \neq \emptyset$ et $UBV(\lambda\overline{x_2}.t) \subseteq UBV(\lambda\overline{x_2}.s) = \{x_1, x_2\}$.

1- La première occurrence de x_1 dans t intervient à la position $p_1 = 1$, la seconde à $p_2 = 2.1$.

- p_1 : les problèmes à résoudre sont :

Sous-terme gauche		Sous-terme droit	Solutions
$\lambda\overline{x_2}.x_1$	$\stackrel{\triangleleft}{=}$	$\lambda\overline{x_2}.t_{\|\Lambda} = \lambda\overline{x_2}.f(x_1, g(x_1), a)$	$\emptyset$
$\lambda\overline{x_2}.x_1$	$\stackrel{\triangleleft}{=}$	$\lambda\overline{x_2}.t_{\|1} = \lambda\overline{x_2}.x_1$	$\theta_1^1 = \{\}$

- p_2 : les problèmes à résoudre sont :

Sous-terme gauche		Sous-terme droit	Solutions
$\lambda\overline{x_2}.x_1$	$\stackrel{\triangleleft}{=}$	$\lambda\overline{x_2}.t_{\|\Lambda} = \lambda\overline{x_2}.f(x_1, g(x_1), a)$	$\emptyset$
$\lambda\overline{x_2}.x_1$	$\stackrel{\triangleleft}{=}$	$\lambda\overline{x_2}.t_{\|2} = \lambda\overline{x_2}.g(x_1)$	$\emptyset$
$\lambda\overline{x_2}.x_1$	$\stackrel{\triangleleft}{=}$	$\lambda\overline{x_2}.t_{\|2.1} = \lambda\overline{x_2}.x_1$	$\theta_2^1 = \{\}$

2- Le seul problème à résoudre est ici $\lambda\overline{x_2}.a \stackrel{\triangleleft}{=} \lambda\overline{x_2}.a$, puisque toutes les autres positions dans t sont au dessus d'une variable liée. On a donc $\theta_3^1 = \{\}$.

3- La reconstruction consiste donc en l'union d'une solution pour F faisant apparaître y_1 à une position au dessus de p_1 et à une position au dessus de p_2, d'une solution θ_1^i, d'une solution θ_2^j, et éventuellement d'une solution θ_3^k. L'union est une solution si les substitutions qui la composent sont compatibles et les positions des variables liées sont indépendantes. Nous obtenons les deux solutions :

$$\sigma'_1 = \theta_1^1 \cup \theta_2^1 \cup \{F \mapsto \lambda\overline{y_3}.t[y_1, y_1]_{p_1,p_2}\}$$

avec $\lambda\overline{y_3}.t[y_1, y_1]_{p_1,p_2} = \lambda\overline{y_3}.f(y_1, g(y_1), a)$, et

$$\sigma'_2 = \theta_1^1 \cup \theta_2^1 \cup \theta_3^1 \cup \{F \mapsto \lambda\overline{y_3}.t[y_1, y_1, y_3]_{p_1,p_2,p_3}\}$$

avec $p_3 = 3$ et $\lambda\overline{y_3}.t[y_1, y_1, y_3]_{p_1,p_2,p_3} = \lambda\overline{y_3}.f(y_1, g(y_1), y_3)$

Deuxième exemple :

$$\{\lambda x.F(a, F(x, b)) \stackrel{?}{=} \lambda x.f(f(b, x), a)\}$$

où f n'est pas un symbole AC.

0- $UBV(\lambda\overline{x}.t) = \{x\} \neq \emptyset$ et $UBV(\lambda\overline{x}.t) \subseteq UBV(\lambda\overline{x}.s) = \{x\}$.

1- L'occurrence de x dans t intervient à la position $p_1 = 1.2$.

- p_1 : les problèmes à résoudre sont :

Sous-terme gauche		Sous-terme droit	Solutions
$\lambda\overline{x}.F(x,b)$	$\stackrel{?}{=}$	$\lambda\overline{x}.t_{\|\Lambda} = \lambda\overline{x}.f(f(b,x),a))$	θ_1^N
$\lambda\overline{x}.F(x,b)$	$\stackrel{?}{=}$	$\lambda\overline{x}.t_{\|1} = \lambda\overline{x}.f(b,x)$	$\theta_2^1 = \{F \mapsto \lambda\overline{y_2}.f(y2, y1)\}$
$\lambda\overline{x}.F(x,b)$	$\stackrel{?}{=}$	$\lambda\overline{x}.t_{\|1.2} = \lambda\overline{x}.x$	$\theta_3^1 = \{F \mapsto \lambda\overline{y_2}.y1\}$

2- La position non encore explorée dans t est la position 2. Ce qui donne le problème $\lambda x.a \stackrel{?}{=} \lambda x.a$ et $\theta_4^1 = \{\}$.

3- Construction des solutions de F : Nous pouvons voir tout de suite qu'aucune des solutions θ_1^N n'est compatible avec $\{F \mapsto \lambda\overline{y_2}.y_1\}$, et il en est de même pour θ_3^1. Ensuite, nous pouvons essayer : $\sigma' = \theta_2^1 \cup \{F \mapsto \lambda\overline{y_2}.t[y_2]_1\}$, mais ces deux substitutions ne sont pas compatibles puisqu'elles affectent deux termes différents à F. Il ne reste que la solution :

$$\sigma' = \theta_2^1 \cup \theta_4^1 \cup \{F \mapsto \lambda\overline{y_2}.t[y_2, y_1]_{1,2}\}$$

avec $\lambda\overline{y_2}.t[y_2, y_1]_{1,2} = \lambda\overline{y_2}.f(y_2, y_1)$.

Enfin, dernier exemple, pour montrer le comportement de la fonction F-R avec des symboles AC :

$$F(F(a,b),c) \stackrel{?}{=} f(b,c,a)$$

où f est un symbole AC.

0- $UBV(\lambda\overline{x}.t) = UBV(\lambda\overline{x}.s) = \emptyset$. On a donc $\sigma'_1 = \{F \mapsto \lambda\overline{y_2}.f(b,c,a)\}$, et nous sautons l'étape 1.

2- Les problèmes sont les suivants :

$F(a,b)$	$\stackrel{?}{=}$	$t_{\|\Lambda} = f(b,c,a)$	$\theta_1^1 = \{F \mapsto \lambda\overline{y_2}.f(b,c,a)\}$
$F(a,b)$	$\stackrel{?}{=}$	$t_{\|1} = b$	$\theta_2^1 = \{F \mapsto \lambda\overline{y_2}.b\}$, $\theta_2^2 = \{F \mapsto \lambda\overline{y_2}.y_2\}$
$F(a,b)$	$\stackrel{?}{=}$	$t_{\|2} = c$	$\theta_3^1 = \{F \mapsto \lambda\overline{y_2}.c\}$
$F(a,b)$	$\stackrel{?}{=}$	$t_{\|3} = a$	$\theta_4^1 = \{F \mapsto \lambda\overline{y_2}.a\}$, $\theta_4^2 = \{F \mapsto \lambda\overline{y_2}.y_1\}$
$F(a,b)$	$\stackrel{?}{=}$	$t_{\|\{1,2\}} = f(b,c)$	$\theta_5^1 = \{F \mapsto \lambda\overline{y_2}.f(b,c)\}$, $\theta_5^2 = \{F \mapsto \lambda\overline{y_2}.f(y_2,c)\}$
$F(a,b)$	$\stackrel{?}{=}$	$t_{\|\{1,3\}} = f(b,a)$	$\theta_6^1 = \{F \mapsto \lambda\overline{y_2}.f(b,a)\}$, $\theta_6^2 = \{F \mapsto \lambda\overline{y_2}.f(y_1,a)\}$
			$\theta_6^3 = \{F \mapsto \lambda\overline{y_2}.f(y_2,b)\}$, $\theta_6^4 = \{F \mapsto \lambda\overline{y_2}.f(y1,y2)\}$
$F(a,b)$	$\stackrel{?}{=}$	$t_{\|\{2,3\}} = f(c,a)$	$\theta_7^1 = \{F \mapsto \lambda\overline{y_2}.f(c,a)\}$, $\theta_7^2 = \{F \mapsto \lambda\overline{y_2}.f(y_1,c)\}$
$F(a,b)$	$\stackrel{?}{=}$	$t_{\|\{1,2,3\}} = f(b,c,a)$	$\theta_8^1\{F \mapsto \lambda\overline{y_2}.f(b,c,a)\}$
c	$\stackrel{?}{=}$	$t_{\|\Lambda} = f(b,c,a)$	$\emptyset$
c	$\stackrel{?}{=}$	$t_{\|1} = b$	$\emptyset$
c	$\stackrel{?}{=}$	$t_{\|2} = c$	$\theta_9^1 = \{\}$
c	$\stackrel{?}{=}$	$t_{\|3} = a$	$\emptyset$
$\vdots$	$\stackrel{?}{=}$	$\vdots$	$\emptyset$

3- Il n'y a pas de solution de la première partie (avec $F(a,b)$ en partie gauche) qui soit compatible avec $\{F \mapsto \lambda\overline{y_2}.f(y_1,c)\}$. La seule recomposition possible est

$$\sigma'_2 = \theta_6^4 \cup \theta_9^1 \cup \{F \mapsto \lambda\overline{y_2}.t[y1,y2]_{1,2}\} = \{F \mapsto \lambda\overline{y_2}.f(y_1,y_2)\}$$

■

II.4.1 Propriétés du nouvel algorithme

Les propriétés à montrer sont relatives au cas flexible-rigide. Les autres cas sont inchangées. Montrons la correction des solutions calculées par l'algorithme. Pour cela, il suffit de montrer la correction de la construction des solutions de la troisième étape de la fonction F-R, et celle de la solution particulière qui peut être engendrée à l'étape 0.

Lemme 52 *Toute substitution σ', résultat de la fonction F-R est un filtre AC de*

$$Pb = \{\lambda\overline{x_n}.F(\overline{s_m}) \stackrel{?}{=} \lambda\overline{x_n}.f(\overline{t_k})\}.$$

Preuve Nous utilisons les notations de l'algorithme. Deux types de solutions peuvent être générées.

- A l'étape 0, si $UBV(s) = UBV(t) = \emptyset$. Alors, $\sigma' = \{F \mapsto \lambda\overline{y_m}.t\}$. Or, si $UBV(t) = \emptyset$,

  $$\sigma'(\lambda\overline{x_n}.F(\overline{s_m})) = \lambda\overline{x_n}.((\lambda\overline{y_m}.t)\ (\overline{s_n})) \rightarrow_\eta \lambda\overline{x_n}.t$$

  Donc, σ' est un filtre AC de Pb.

- Les autres solutions générées sont de la forme :

  $$\sigma' = \cup_{\alpha=1}^{P}\theta_\alpha^K \cup \{F \mapsto \lambda y_m.t[y_{i_1},\ldots,y_{i_P}]_c\}$$

  avec les conditions précisées. Tout d'abord, puisque toutes les substitutions θ_α^K sont compatibles entre elles et avec $\{F \mapsto \lambda y_m.t[y_{i_1},\ldots,y_{i_P}]_c\}$, σ' est bien une substitution. Nous avons, par application d'une telle substitution

  $$\sigma'(\lambda\overline{x_n}.F(\overline{s_m})) \rightarrow_\beta \lambda\overline{x_n}.t[\sigma'(s_{i_1}),\ldots,\sigma'(s_{i_P})]_c =_{AC} \lambda\overline{x_n}.t[\theta_1^{k_1}(s_{i_1}),\ldots,\theta_P^{k_P}(s_{i_P})]_c$$

  où les θ_α^K sont des filtres AC des problèmes $\lambda\overline{x_n}.s_\alpha \stackrel{?}{=} \lambda\overline{x_n}.t_{|p_{i_j}}$ pour $\alpha = i_1,\ldots,i_P$ où p_{i_j} est la position associée à y_{i_j} dans c. Par ailleurs, c est une coupure de t, donc toutes les positions qu'elle contient sont indépendantes. On a donc bien $\sigma'(\lambda\overline{x_n}.F(\overline{s_m})) =_{AC} \lambda\overline{x_n}.t$.

□

La correction de la règle (flex-rig) en découle, et nous pouvons formuler le théorème de correction.

Théorème 53 *(Correction de l'algorithme de filtrage) Pour un problème de filtrage $< S,\{\} >$, toute solution obtenue par application de l'algorithme de filtrage est un filtre AC de ce problème.*

Preuve Par les lemmes 30, 33 et 52. □

Maintenant que nous savons que les substitutions que nous construisons sont bien des filtres AC du problème, assurons-nous de la terminaison du processus. En premier lieu, montrons que les problèmes engendrés par la fonction F-R sont tous d'une complexité strictement plus petite que celle du problème initial. Pour cela, nous utiliserons la même mesure de complexité que pour l'algorithme naïf.

Lemme 54 *Tous les problèmes de filtrage générés par la fonction F-R ont une complexité strictement inférieure à celle du problème d'entrée de la fonction.*

Preuve L'étape 0 n'engendre pas de problème de filtrage. Lors de l'étape 1, nous devons résoudre des problèmes du type

$$\{\lambda\overline{x_n}.s_j \stackrel{?}{=} \lambda\overline{x_n}.t_{|p}\}$$

où s_j est un argument de F, et donc un sous-terme strict du terme flexible, et $t_{|p}$ ne peut pas être plus grand que t. Par conséquent, ξ_1 diminue strictement, et ξ_2 n'augmente pas. D'où $CS(S) =< \xi_2(S), \xi_1(S) >$ décroît strictement.

Pour les problèmes engendrés par l'étape 2, le cas est exactement le même. Quant à l'étape 3, elle n'engendre pas de problème. □

Formulons à présent le théorème de terminaison.

Théorème 55 *Toute séquence d'application des règles (E) pour la théorie AC, (D) et de (flex-rig) termine.*

Preuve Pour les règles (E) et (D), le résultat est donné par le lemme 38. Pour la fonction F-R, il suffit de nous assurer que le nombre de problèmes générés est toujours fini. Pour les étapes 1 et 2, nous pouvons dire que F possède un nombre fini d'arguments, et que t possède un nombre fini de positions aplaties. Le nombre de combinaisons des deux est donc fini. Pour ce qui est de l'étape 3, elle consiste à combiner, là encore, un nombre fini d'éléments de solutions. □

Intéressons-nous maintenant à la complétude. De la même façon que pour la correction, nous avons à montrer la complétude du cas flexible-rigide. Pour un problème de la forme

$$< \{\lambda\overline{x_n}.F(\overline{s_m}) \stackrel{?}{=} \lambda\overline{x_n}.f(\overline{t_k})\}, \{\} >$$

toutes les solutions pour F, sont de la forme

$$F \mapsto \lambda\overline{y_m}.t'[y_{i_1}, \ldots, y_{i_P}]_{c'}$$

avec $t' =_{AC} t$ – parce que la théorie AC est un théorie permutative – et telle qu'il existe c, coupure de t telle que $\lambda\overline{y_m}.t'[y_{i_1}, \ldots, y_{i_P}]_{c'} =_{AC} \lambda\overline{y_m}.t[y_{i_1}, \ldots, y_{i_P}]_c$. Une solution, pour être complet, serait de générer toutes les substitutions construites sur ce modèle. Les solutions qui sont engendrées par la fonction F-R forment un sous-ensemble de ces substitutions, et la complétude consiste à montrer que les substitutions que nous évitons de construire ne peuvent pas mener à une solution du problème; ou encore à montrer que les conditions imposées sont des conditions nécessaires pour obtenir des filtres AC du problème. En premier lieu, nous allons donner la propriété relative aux variables liées réellement utilisées dans les termes.

Lemme 56 *Soit σ, un filtre AC de $Pb = \{\lambda x_n.F(\overline{s_m}) \stackrel{?}{=} \lambda\overline{x_n}.f(\overline{t_k})\}$. Alors,*

$$\sigma(F) = \{F \mapsto \lambda\overline{y_m}.t'[y_{i_1}, \ldots, y_{i_P}]_{c'}\}$$

avec

- $t' =_{AC} t = f(\overline{t_k})$,
- $\forall x_i \in UBV(t)$, $\exists y_{i_j} \in UBV(\sigma(F))$ *avec* x_i *apparaissant dans* s_{i_j},
- *pour* p_i, *la position de* x_i *dans* t, *la position* p_{i_j} *associée à* y_{i_j} *dans* c', *coupure de* t' *est telle que* $p_{i_j} \leq p_i$ *(p_{i_j} est au dessus de p_i).*

Preuve Avant tout, nous pouvons ramener le cas AC au cas syntaxique. En effet, la théorie AC est dite permutative. Ce qui signifie que deux termes clos aplatis sont AC équivalents si et seulement si il est possible d'aller de l'un à l'autre par une série de permutations des arguments des symboles AC. Par conséquent, pour tout terme t' et toute coupure c' de t', si $t' =_{AC} t''$, t'' peut être obtenu par permutations sur t' et il existe c'', une coupure de ce terme t'' telle que

$$\lambda\overline{y_m}.t'[y_{i_1}, \ldots, y_{i_P}]_{c'} =_{AC} \lambda\overline{y_m}.t''[y_{i_1}, \ldots, y_{i_P}]_{c''}$$

Ce qui nous permet de ne considérer que $t' = t$, c'est-à-dire la matrice du terme rigide, et c une coupure de ce terme. Considérons donc une solution de la forme :

$$\sigma(F) = \{F \mapsto \lambda\overline{y_m}.t[y_{i_1}, \ldots, y_{i_P}]_c\}$$

L'application de cette substitution au terme flexible, puis la β-réduction mènent à :

$$\lambda\overline{x_n}.t[\sigma(s_{i_1}), \ldots, \sigma(s_{i_P})]_c =_{AC} \lambda\overline{x_n}.t$$

La condition $UBV(t) = UBV(s)$ doit toujours être vérifiée. Or, l'application d'une substitution à un λ-terme t ne peut pas ajouter d'élément à l'ensemble $UBV(t)$ (elle peut simplement ajouter des éléments au multi-ensemble des variables liées utilisées, par duplication). En effet, par la définition de l'application d'une substitution,

$$\{Y \mapsto x\}(\lambda x.Y) \rightarrow \lambda x.Z$$

où Z est une nouvelle variable libre. Ainsi, si x_i apparaît dans $\sigma(s_{i_j})$, x_i apparaît dans s_{i_j}.

De plus, $\sigma(\lambda\overline{x_n}.s_{i_j}) =_{AC} \lambda\overline{x_n}.t_{|p_{i_j}}$ où p_{i_j} est la position associée à y_{i_j} dans c. Donc, x_i doit apparaître dans t à une position p_i telle que $p_{i_j} \leq p_i$. □

Lemme 57 *Quel que soit σ, filtre AC du problème $Pb =< \{\lambda\overline{x_n}.F(\overline{s_m}) \stackrel{?}{=} \lambda\overline{x_n}.f(\overline{t_k})\}, \{\} >$, il existe une solution σ' résultat de la fonction F-R avec l'argument Pb, telle que $\sigma' =_{AC} \sigma[\mathcal{FV}(Pb)]$.*

Preuve Nous montrons que les solutions pour les sous-termes contenant des variables de $UBV(t)$ sont nécessairement des solutions des problèmes engendrés par l'étape 1 de la fonction F-R, et que les autres sont nécessairement des solutions des problèmes engendrés par l'étape 2. De plus, les conditions imposées par l'étape 3 pour la construction des solutions pour F sont des conditions nécessaires pour la correction.

De façon générale, pour $\sigma(F) = \lambda\overline{y_m}.t'[y_{i_1}, \ldots, y_{i_P}]_{c'} =_{AC} \lambda\overline{y_m}.t[y_{i_1}, \ldots, y_{i_P}]_c$, nous pouvons écrire

$$\sigma(\lambda\overline{x_n}.F(\overline{s_m})) =_{AC} \lambda\overline{x_n}.t[\sigma(s_{i_1}), \ldots, \sigma(s_{i_P})]_c$$

Soient $\sigma_1 = \sigma[\mathcal{FV}(s_{i_1})], \ldots, \sigma_P = \sigma[\mathcal{FV}(s_{i_P})]$. On a alors,

$$\sigma =_{AC} \sigma_1 \cup \ldots \cup \sigma_P \cup \{F \mapsto \lambda\overline{y_m}.t[y_{i_1}, \ldots, y_{i_P}]_c$$

Le lemme 56 nous permet de dire que pour toute solution, pour chacun des x_i, élément de $UBV(t)$, il existe y_{i_j} tel que $y_{i_j} \in UBV(\sigma(F))$, avec

- x_i apparaît dans s_{i_j} et
- p_{i_j}, position associée à y_{i_j}, vérifie $p_{i_j} \leq p_i$ où p_i est la position de x_i dans t.

Si nous appelons $\{\overline{\sigma_J}\}$ l'ensemble des substitutions σ_{i_j}, où un x_i apparaît dans $\sigma_{i_j}(y_{i_j})$, alors chacun des σ_{i_j} est un filtre AC de $\lambda\overline{x_n}.s_{i_j} \stackrel{\triangleleft}{=} \lambda\overline{x_n}.t_{|p_{i_j}}$ où p_{i_j} est une position au dessus de celle de x_i.

Toutes les substitutions de ce type sont calculées par l'étape 1 de la fonction F-R. En effet, les restriction énoncées sont celles de l'étape 1.

Par ailleurs, toutes les positions autres que ces p_{i_j} sont indépendantes, puisque c est une coupure de t. Donc, pour les substitutions σ_j telles que x_i n'apparaît pas dans $\sigma_j(s_j)$, sont des solutions de $\lambda\overline{x_n}.s_j \stackrel{\triangleleft}{=} \lambda\overline{x_n}.t_{|q}$ où q est une position de t indépendante de tous les p_{i_j} et s_j un argument de F. Ce sont les substitutions calculées par l'étape 2 de la fonction F-R.

Enfin, l'étape 3 consiste à faire toutes les combinaisons possibles, vérifiant trois conditions dont nous avons vu qu'elles sont nécessaires

- pour que σ' soit un filtre AC du problème pour les deux premières,
- pour que σ' soit une substitution pour la troisième.

□

Théorème 58 *Soit θ, un filtre AC du problème de filtrage $< S, \{\} >$, il existe une solution obtenue par une séquence d'application des règles (D) et (E) dans le cas rigide-rigide, et de la règle (flex-rig) dans le cas flexible-rigide dont le résultat $< \{\}, \theta_f >$ est tel que $\theta_f =_{AC} \theta[\mathcal{FV}(S)]$.*

Preuve Par les lemmes 30, 36 et 57. □

Nous avons donc conservé les propriétés de l'algorithme naïf, tout en l'améliorant sur trois points essentiels :

- suppression des variables d'ordre deux introduites durant le déroulement de l'algorithme,
- calcul d'un ensemble restreint de solutions par filtrage grâce à la composition de solutions élémentaires,
- élimination de la plupart des redondances.

Cet algorithme constitue dans la suite un outil fondamental pour la reconnaissance des similitudes entre les formules, puis pour le calcul de la différence. A présent, nous possédons un outil puissant et efficace, et nous allons l'appliquer à la preuve par analogie.

III

Analogie pour des formules similaires

Les analogies que nous avons étudiées jusqu'à présent supposaient que la formule de référence est enchâssée dans la formule objet, la similitude entre les formules étant d'ordre propositionnel. C'est ce que nous avons appelé des analogies simples. Pour ce cas, nous avons décrit une méthode complète pour résoudre automatiquement l'analogie. Dans le présent chapitre, nous allons étendre largement le champ des analogies automatisables. Notre définition intuitive de la similitude est qu'une formule B est similaire à A si une preuve de B peut être construite automatiquement à partir d'une preuve de A. Concrètement, on sait par exemple, qu'une preuve de typage d'un λ-terme est similaire – au sens intuitif – à une déduction naturelle. Notre notion de similitude devait donc permettre de passer de l'un à l'autre. En fait, pour nous, une similitude est une relation R telle que :

- R est décidable,
- si $R(A, B)$ et A est valide, alors B est valide,
- si $R(A, B)$ et on a une preuve de A, alors on peut construire une preuve de B par un algorithme.

Dans ce chapitre, nous montrons que la notion de similitude peut englober :

- la correspondance entre deux formules. Par exemple, les formules $A = \forall x\ p(f(x), a)$ et $B = \forall x\ q(x)$ peuvent correspondre, parce que, par abstraction de A, nous avons le terme $\forall(\lambda x.P(F(x), Z))$ où P, F et Z sont des variables libres; et le filtrage de ce terme avec $\forall(\lambda x.q(x))$ qui représente B, donne la solution $\{P \mapsto \lambda z_1 z_2.q(z_1)\ ;\ F \mapsto \lambda z.z\}$. Ainsi, en appliquant la correspondance $\{p \mapsto \lambda z_1 z_2.q(z_1)\ ;\ f \mapsto \lambda z.z\}$ – qui est la composition de l'abstraction et de la solution du filtrage –, de A, nous obtenons B. Nous verrons comment appliquer cette même correspondance aux preuves.
- Les propriétés AC des connecteurs $\vee$ et $\wedge$. Pour le calcul de la correspondance ci-dessus, il suffit de considérer les propriétés AC des connecteurs lors du filtrage d'ordre deux.
- Certaines propriétés des quantificateurs. Les formules $\forall x\ (\exists y A(x, y) \wedge B(x)), \forall x\ \exists y\ (A(x, y) \wedge B(x))$ et $\forall x\ \exists y\ A(x, y) \wedge \forall z\ B(z)$ sont équivalentes. Pour tenir compte de ces propriétés, nous utilisons une forme normale qui consiste à descendre les quantificateurs aussi profondément que possible dans les formules. Cette forme normale ayant l'intérêt d'être unique

pour une formule donnée. La mise en forme anti-prénexe a déjà été utilisée de façon assez proche dans [Bib74], et beaucoup plus récemment dans [BdlTCC88, Egl94] sous le nom d'"antiprenexing" ou de "miniscoping". Les preuves peuvent également être transformées d'une forme à l'autre. Ainsi, si deux formules ont la même forme normale, une preuve de l'une peut être transformée en une preuve de l'autre.

Nous montrons qu'il est également possible de combiner ces différents critères de similitude. Pour terminer, nous traitons un exemple. Il s'agit de confirmer l'idée intuitive selon laquelle il y a similitude entre typage et déduction naturelle. Il existe, bien sûr, des résultat théoriques à ce sujet, mais notre but est simplement de montrer que l'on peut formaliser la notion de similitude et automatiser l'analogie qui rentre dans ce cadre. Cette similitude n'est pas traitée de façon *ad-hoc*, mais avec un processus général, pouvant aussi bien être appliqué dans le cadre de l'isomorphisme de Curry-Howard que pour la transformation d'une preuve sur les listes en une preuve sur les ensembles. La condition à respecter est la définition formelle que nous donnons de la similitude. Cette définition n'a rien d'absolu, elle essaie simplement d'utiliser quelques propriétés qui font que deux formules logiques sont équivalentes ou peuvent être considérées comme telles.

III.1 Correspondance

Le cas de figure qui nous préoccupe ici est une formule objet B qui peut être obtenue de la formule de référence A par correspondance entre les termes. Prenons par exemple les formules $A = \forall x\ r(f(x), b)$, et $B = \forall x\ (p(x) \wedge q(a))$. La correspondance qui peut être établie entre les termes représentant ces formules est la suivante :

$$\begin{aligned} \lambda x_1 x_2 . r(x_1, x_2) &\leftrightarrow \lambda x_1 x_2 . x_1 \wedge x_2 \\ \lambda x_1 . f(x_1) &\leftrightarrow \lambda x_1 . p(x_1) \\ b &\leftrightarrow q(a) \end{aligned}$$

Nous allons, pour reconnaître ce type de correspondance, utiliser le filtrage d'ordre deux. Cette notion de correspondance englobe celle de renommage des symboles utilisée par exemple dans [Pla81]. Ainsi, pour deux formules $A = \forall x\ p(x, a) \Rightarrow \exists x\ p(a, x)$ et $B = \forall x\ q(x, b) \Rightarrow \exists x\ q(b, x)$, un simple renommage de p en q et de a en b permet de passer de A à B. Dans ce cas, il n'est pas utile de mettre en œuvre un processus d'ordre deux. Mais les correspondances que nous allons pouvoir déterminer ici sont beaucoup plus puissantes. En effet, pour le premier exemple, un processus d'ordre un ne peut pas faire correspondre les deux termes. La première étape, de reconnaissance de ces correspondances va se faire en deux temps. D'abord, nous allons abstraire la formule de référence, puis filtrer en utilisant l'algorithme donné au chapitre II. La seconde étape consiste à transformer la preuve de la formule de référence en une preuve de la formule objet. Nous montrerons que la méthode est complète. C'est-à-dire que s'il existe une correspondance, elle permet de construire automatiquement une preuve de la formule objet. Ce type de similitude a été étudié par Plaisted dans [Pla81] dans le cadre de la résolution. Mais elle n'englobait que le renommage qui est un affaiblissement de la correspondance.

A Abstraction d'une formule

Il s'agit de transformer une formule d'ordre un, en une formule dont toute instance close est une formule similaire à celle de départ.

Définition 1 Soit A une formule du premier ordre, et $\{\overline{c_n}\}$ l'ensemble des symboles de constantes libres apparaissant dans A telles que $O(c_i) \leq 2$ pour $i = 1, \ldots, n$. $Abst = \{\overline{c_n \mapsto X_n}\}$ où $\overline{X_n}$ est un ensemble de nouvelles variables libres du type approprié et telles que $X_i = X_j$ si et seulement si $c_i = c_j$. □

Exemple 2 Par exemple, la formule $p(0) \wedge \forall x\ (p(x) \Rightarrow p(s(x))) \Rightarrow\ \forall x\ p(x)$ sera abstraite en un schéma :

$$P(X) \wedge \forall x\ (P(x) \Rightarrow P(F(x))) \Rightarrow\ \forall x\ P(x)$$

où P, F et X sont des variables libres. ■

Rappelons que les constantes qui ne sont pas considérées comme libres sont les connecteurs $\vee$ et $\wedge$, mais aussi la négation $\neg$ et les quantificateurs $\exists$ et $\forall$. Dans [KW94], Kolbe et Walther abstraient leurs preuves de façon semblable pour une procédure par analogie sur les preuves par induction.

B Filtrage d'une formule abstraite et d'une formule d'ordre un

Définition 3 Soit *Abst*, l'abstraction de la formule A, et σ une solution du problème de filtrage $Abst(A) \stackrel{?}{=} B$, alors

$$\rho = \sigma \circ Abst \text{ est une correspondance de } B \text{ vers } A$$

□

Cette définition implique que la notion de correspondance n'est pas symétrique, puisque $\rho A = B \not\Rightarrow \rho B = A$. Globalement, une correspondance est donc de la forme $\rho = \{c_n \mapsto c'_n\}$ où les c_i sont les constantes d'ordre un et deux de A, et les c'_i des termes clos. Nous pouvons donner notre premier critère de similitude.

Définition 4 Une formule B est *similaire par correspondance* à une formule A si et seulement si B peut être obtenue de A par application d'une correspondance. □

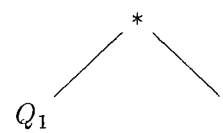
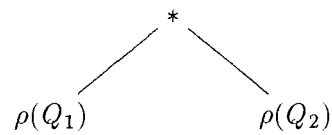
A présent que nous avons défini ce premier critère de similitude entre les formules, nous allons définir la transformation qui, appliquée à une preuve de la formule de référence, fournit une preuve de la formule objet, lorsqu'il existe une correspondance entre ces formules.

Définition 5 L'application d'une correspondance ρ à un arbre d'expansion Q, est définie par $Q' = \rho(Q)$, où Q' représente Q où ρ est appliquée aux formules de chaque nœud de Q, et à chaque étiquette. □

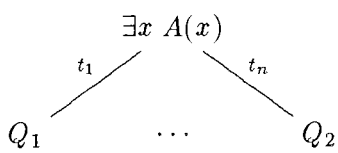
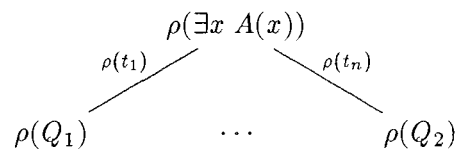
Montrons que l'application de la correspondance à tout l'arbre d'expansion, représentant la formule de référence, fournit un arbre d'expansion de la formule objet. Nous montrerons ensuite que la connexion couvre toujours l'arbre en question.

Lemme 6 *Soit ρ une correspondance entre les formules B et A. Si Q est un arbre d'expansion représentant A, alors $\rho(Q)$ est un arbre d'expansion représentant B.*

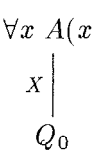
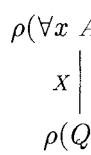
Preuve Par induction structurelle sur Q.
Cas de base : $Q = A$ (arbre réduit à une feuille). Alors, $\rho(Q)$=B et $\rho(Q)^s = B$.
Cas inductifs (Q et Q' ne peuvent être en correspondance que s'ils ont même racine puisque les connecteurs et les quantificateurs ne sont pas des constantes libres) :

- Pour $* = \vee$ ou $\wedge$: $Q =$  et $Q' =$ 

Vérifié par application directe de l'hypothèse d'induction à $\rho(Q_1)$ et $\rho(Q_2)$

- $Q =$  et $Q =$ 

Or, par hypothèse d'induction, $\rho(Q_i)^s = B_i = \rho(A(t_i))$ et $\rho(Q_i)$, arbre d'expansion pour B_i. On a donc pour $i = 1, \ldots, n$ $\rho(Q_i)^s = \rho(A(t_i)) = \rho(A(\rho(t_i)))$, et $Q'^s = \rho(\exists x\ A(x))$ et $\rho(Q)$, arbre d'expansion pour $\rho(Q^s)$.

- $Q =$  et $Q' =$ 

avec $X \notin \mathcal{D}om(\rho)$, puisque, par définition, X ne peut pas apparaître dans A. Par induction, $\rho(Q_0)$ est un arbre d'expansion pour $\rho(Q_0)^s$. On a donc $\rho(Q_0)^s = \rho(A(X))$. D'où Q' arbre d'expansion pour $\rho(\forall x\ A(x)) = B$.

Pour la précédence, $s <_Q t \equiv \rho(s) <_{\rho(Q)} \rho(t)$, puisque si x est sélectionnée sous t dans Q, elle le sera également sous $\rho(t)$ dans $\rho(Q)$, et si x est libre dans $\rho(s)$, elle le sera également dans $\rho(s)$. De plus, aucune variable sélectionnée n'est introduite. □

Vérifions maintenant que la connexion est préservée par correspondance.

Lemme 7 *Soit ρ, une correspondance de B vers A. Si Q est un arbre d'expansion représentant A et si $\mathcal{M}$ couvre Q, alors $\rho(\mathcal{M})$ couvre $\rho(Q)$.*

Preuve Si la correspondance conserve l'égalité des termes, nous aurons $A = \neg B \Rightarrow \rho(A) = \neg\rho(B)$ puisque les connecteurs ne sont pas modifiés par application d'une correspondance. Par conséquent, pour toute paire $(A, B) \in \mathcal{M}$, nous avons $\rho(A) = \neg\rho(B)$, et puisque $\mathcal{M}$ couvre Q, $\rho(\mathcal{M})$ couvre $\rho(Q)$. □

Le théorème suivant découle de ces deux lemmes.

Théorème 8 *Soit ρ, une correspondance de la formule B vers A,*

$$(Q, \mathcal{M}) \vdash A \Rightarrow (\rho(Q), \rho(\mathcal{M})) \vdash B$$

Le problème de correspondance est donc relativement simple. Mais nous avons un algorithme de filtrage qui supporte les propriétés AC des connecteurs logiques. Nous allons lier la similitude par correspondance à la similitude modulo ces propriétés en utilisant l'algorithme de filtrage AC au lieu du filtrage syntaxique.

III.2 Propriétés AC des connecteurs

Le but de notre algorithme de filtrage décrit au chapitre II, est de prendre en compte certaines propriétés des connecteurs. Si nous sommes capables de produire, par analogie, une preuve de B à partir d'une preuve de A, il serait très restrictif de ne pas considérer les formules équivalentes à B modulo l'associativité-commutativité du $\vee$ et du $\wedge$. Dans le cas contraire, nous ne serions même pas capables de construire une preuve de $a \wedge b$ à partir d'une preuve de $b \wedge a$. Pour ce faire, il suffit d'utiliser l'algorithme de filtrage d'ordre deux qui tient compte de ces propriétés et de s'assurer que l'on peut trouver la preuve de B automatiquement. En réalité, il n'y a rien à faire de plus que de modifier la définition des connexions en permettant de connecter deux nœuds de l'arbre d'expansion Q_1 et Q_2 si $Q_1^s =_{AC} \neg Q_2^s$. Ce que nous définissons de façon formelle.

Définition 9 Une relation symétrique $\mathcal{M}_{AC}$ est un *ensemble d'AC-connexions* si c'est un ensemble de couples non-ordonnés (l^s, k^s) tels que $l^s =_{AC} \neg k^s$ pour chacun de ces couples. Dans la suite, nous parlerons simplement d'une *connexion*, pour un ensemble d'AC-connexions. □

Nous définissons maintenant la similitude modulo les propriétés AC des connecteurs en la liant directement à celle par correspondance.

Définition 10 Une formule B est AC-similaire par correspondance à une formule A si et seulement si il existe une correspondance ρ telle que $\rho A =_{AC} B$. Dans la suite, nous dirons que B est obtenue de A par AC-correspondance. □

Vérifions que l'AC-correspondance permet de construire directement la preuve de la formule objet.

Lemme 11 *Soit $\rho = \sigma(Abst)$ une AC-correspondance entre la formule B et la formule A.*

$$Si\ (Q, \mathcal{M}) \vdash A,\ alors\ (\rho(Q), \rho(\mathcal{M})) \vdash B.$$

Preuve $\rho(Q)$ est un arbre d'expansion pour $\rho(A) = B$ à condition de remplacer, dans la définition, les égalités syntaxiques $Q^s = A(t_i)$ et $Q^s = A(X)$ respectivement par $Q^s =_{AC} A(t_i)$ et $Q^s =_{AC} A(X)$. Les conditions sur la précédence $<_{\rho(Q)}$ et sur les variables sélectionnées sont vérifiées comme dans le lemme 6. Et $\rho(\mathcal{M})$ couvre $\rho(Q)$ en considérant la nouvelle définition des connexions, puisque la relation entre les formules associées aux nœuds de Q et de $\rho(Q)$ est $\rho(Q_0)^s =_{AC} Q_0^s$ pour tous les nœuds Q_0 de Q. □

Les propriétés AC des connecteurs ne posent donc pas de problème majeurs. La difficulté est en fait déplacée dans le filtrage. Nous allons nous intéresser à présent à un autre type de propriété, qui concerne les quantificateurs.

III.3 Propriétés des quantificateurs

Tout comme pour les propriétés AC, qui préservent l'équivalence des formules, il existe des propriétés des quantificateurs, qui rendent les formules différentes a priori, alors qu'il y a équivalence logique entre elles. Regardons les propriétés de quantificateurs qui devraient être considérées. Dans cette section, nous ne considérons que la forme normale négative des formules, mais l'extension de tout ce qui est dit aux formules générales, il suffit, par exemple, de transformer les formules en leur forme normale négative, puisqu'elle est unique modulo le renommage des variables liées. D'autre part, indépendamment de l'analogie, les manipulations de quantificateurs dans les formules que nous allons donner peuvent être utilisées pour éliminer les quantificateurs inutiles dans un théorème.

III.3.1 Propriétés à prendre en compte

Lorsque deux mêmes quantificateurs apparaissent côte à côte dans une formule, ils peuvent commuter.

$$Qx\ Qy\ A(x,y) \iff Qy\ Qx\ A(x,y)$$

Pour $Q = \exists$ ou $Q = \forall$. Par exemple

$$\forall x\ \forall y\ A(x,y) \iff \forall y\ \forall x\ A(x,y)$$

Le quantificateur $\forall$ est distributif par rapport au connecteur $\wedge$. Ceci s'exprime par la formule,

$$\forall x\ (A(x) \wedge B(x)) \iff \forall x\ A(x) \wedge \forall x\ B(x).$$

Symétriquement, le quantificateur $\exists$ est distributif par rapport au connecteur $\vee$. Ce qui s'exprime par la formule,

$$\exists x\ (A(x) \vee B(x)) \iff \exists x\ A(x) \vee \exists x\ B(x).$$

Enfin, la dernière propriété que nous évoquerons concerne le nom des variables. Les quantificateurs sont des lieurs dans les formules, au même titre que l'abstraction λ dans les λ-termes. Ainsi, la variable qu'il lie est-elle muette. C'est à dire que son nom n'a pas d'importance. Encore faut-il respecter certaines règles de visibilité. On peut remarquer que la représentation classique des quantificateurs dans les langages qui représentent les formules par des λ-termes (ELF par exemple), font justement le rapport entre le λ du λ-calcul et les quantificateurs dans les formules. Ainsi, on représente la formule

$$\forall x\ A(x) \text{ par le terme } \forall(\lambda x.A(x)).$$

Il suffit donc de suivre les règles du renommage du λ-calcul pour les termes représentant les formules. Dans ce qui suit, nous allons évoquer les propriétés énoncées plus haut, mais le renommage des variables, lui, va être implicite, puisque nous travaillons avec des λ-termes modulo l'α-conversion. Pour prendre en compte ces propriétés des quantificateurs, observons ce qui se passe lors de la mise en forme prénexe des formules, et nous allons voir que l'utilisation d'une forme anti-prénexe, semblable à ce qui est utilisé dans [Egl94] nous permettra de considérer les propriétés présentées.

III.3.2 La forme prénexe

Classiquement, lorsque l'on veut simplifier la forme des formules logiques que l'on manipule, on utilise une forme prénexe, qui consiste à sortir les quantificateurs pour les positionner en début de formule. Ainsi, la formule $\forall x\ (\forall y\ A(y,x) \wedge \exists z\ B(x,z))$ pourra s'écrire sous la forme prénexe $\forall x\ \forall y\ \exists z\ (A(y,x) \wedge B(x,z))$ si $y \notin \mathcal{FV}(B(x,z))$ et $z \notin \mathcal{FV}(A(y,x))$. Ces deux formules sont équivalentes du point de vue logique. Par contre, une formule peut avoir plusieurs formes prénexes. Par exemple, la formule $\forall x\ \exists z\ \forall y\ (A(y,x) \wedge B(x,z))$ est également une forme prénexe de la première. Or, deux quantificateurs différents ne commutent pas, en général. Il est donc difficile de voir qu'il s'agit de formules équivalentes. Rappelons brièvement les règles de mise sous forme prénexes (pour des formules en FNN) :

$$\begin{array}{lcll} (Qx\ A) \vee B & \rightarrow & Qx\ (A \vee B) & \text{quand } x \notin \mathcal{FV}(B) \\ (Qx\ A) \wedge B & \rightarrow & Qx\ (A \wedge B) & \text{quand } x \notin \mathcal{FV}(B) \\ & & & \text{pour } Q = \exists, \forall \end{array}$$

Selon l'ordre d'application de ces règles, on obtiendra une forme prénexe, ou bien une autre, comme l'illustre l'exemple qui précède. Or, nous avons besoin d'une forme normale unique afin de pouvoir affirmer que deux formules ayant la même forme normale sont équivalentes (mais malheureusement, deux formules équivalentes n'ont pas forcément la même forme normale). Nous allons pour cela utiliser une forme normale qui s'oppose à la forme prénexe dans ce sens qu'elle positionne les quantificateurs aussi profondément que possible dans la formule. Ce type de transformation a déjà été utilisé, et même implanté par Boy de la Tour et al. [BdlTCC88]. Mais le but était de réduire le champ d'application des quantificateurs avant la skolémisation des formules, afin de faciliter la résolution. Dans notre cas, c'est l'obtention d'une forme normale unique – pour une formule donnée – qui motive cette transformation.

III.3.3 Forme normale anti-prénexe

Pour éviter les problèmes rencontrés avec la forme prénexe, nous allons descendre les quantificateurs dans le terme au lieu de les remonter. Grâce à des tests d'occurrence des variables dans les sous-termes, les quantificateurs vont être distribués quand cela est possible. Définissons le système de réécriture de mise en forme anti-prénexe. Selon les différents cas de figure, Q peut être remplacé par l'un ou l'autre des quantificateurs, et $*$ par $\vee$ ou $\wedge$. Nous utilisons Q et Q' lorsque les quantificateurs sont différents. Enfin, le signe $\tilde{}$ n'est qu'un marquage des quantificateurs.

La règle (1) est la règle de base. Si la variable liée n'apparaît pas dans A ou si A est un littéral. On remarque alors qu'un quantificateur qui ne peut plus être déplacé vers le bas de la formule est marqué. (2) et (3) reflètent les cas de non-distributivité. (4) et (5) sont les cas de distributivité. Dans les cas (6) et (7), la variable liée par le quantificateur courant n'apparaît que d'un côté. Le cas (8) simule la commutativité des quantificateurs identiques. On voit que le quantificateur le plus proche de la formule ne peut être descendu, mais peut-être celui qui est devant lui le peut-il, à condition qu'il s'agisse du même quantificateur. Le cas (9), enfin, dit que si le quantificateur Q' ne peut pas être descendu dans la formule, et qu'il est différent de Q, Q est bloqué également, puisque deux quantificateurs différents ne peuvent commuter, en général.

Définition 12 La mise en forme anti-prénexe d'une formule consiste à appliquer les règles ci-dessus aux quantificateurs depuis ceux, situés le plus bas dans la formule, jusqu'à ceux, situés le

(1)	$Qx\ A$	$\rightarrow$	$\tilde{Q}x\ A$ A	si $x \in \mathcal{FV}(A)$ et A est un littéral si $x \notin \mathcal{FV}(A)$
(2)	$\forall x\ (A \vee B)$	$\rightarrow$	$\tilde{\forall} x\ (A \vee B)$	pour $x \in \mathcal{FV}(A) \cap \mathcal{FV}(B)$
(3)	$\exists x\ (A \wedge B)$	$\rightarrow$	$\tilde{\exists} x\ (A \wedge B)$	pour $x \in \mathcal{FV}(A) \cap \mathcal{FV}(B)$
(4)	$\forall x\ (A \wedge B)$	$\rightarrow$	$\forall x\ (A) \wedge \forall x\ (B)$	pour $x \in \mathcal{FV}(A) \cap \mathcal{FV}(B)$
(5)	$\exists x\ (A \vee B)$	$\rightarrow$	$\exists x\ (A) \vee \exists x\ (B)$	pour $x \in \mathcal{FV}(A) \cap \mathcal{FV}(B)$
(6)	$Qx\ (A * B)$	$\rightarrow$	$Qx\ (A) * B$	pour $x \in \mathcal{FV}(A)$ et $x \notin \mathcal{FV}(B)$
(7)	$Qx\ (A * B)$	$\rightarrow$	$A * Qx\ (B)$	pour $x \notin \mathcal{FV}(A)$ et $x \in \mathcal{FV}(B)$
(8)	$Qx_1\ \tilde{Q}x_2\ A$	$\rightarrow$	$\tilde{Q}x_2\ Qx_1\ A$	
(9)	$Qx_1\ \tilde{Q}'x_2\ A$	$\rightarrow$	$\tilde{Q}x_1\ \tilde{Q}'x_2 A$	pour $Q \neq Q'$

Figure III.1: Règles de mise en forme anti-prénexe

plus haut. La formule est en forme normale lorsqu'elle ne comporte plus de quantificateurs non marqués. □

Lemme 13 *Toute formule possède une forme normale anti-prénexe qui lui est équivalente.*

Preuve L'équivalence entre les membres gauches et les membres droits des règles est triviale en considérant les conditions d'application. C'est donc vrai par induction structurelle sur la formule. □

Lemme 14 *Le processus de mise en forme anti-prénexe termine.*

Preuve Soit h la profondeur d'un symbole dans la formule. On définit sa hauteur par $(t - h)$ où t est la hauteur de la formule. Prenons ξ comme mesure de complexité où ξ est le multi-ensemble des hauteurs des quantificateurs non encore marqués. Chacune des règles fait décroître strictement ξ par l'extension classique de l'ordre aux multi-ensembles. □

Lemme 15 *Le système de mise en forme anti-prénexe est localement confluent.*

Preuve Pour que deux règles soient simultanément applicables, il est nécessaire que deux quantificateurs se trouvent à la même profondeur dans la formule. Par conséquent, ces deux quantificateurs se trouvent à des positions indépendantes. Appliquer une règle à l'un des quantificateurs revient à mettre le sous-terme dont il est la tête en forme normale. Il est donc indifférent de mettre l'un ou l'autre de ces sous-termes en forme normale en premiers, puisqu'ils apparaissent à des positions indépendantes. Nous obtiendrons la même forme normale. □

De ces trois lemmes, découle le résultat suivant :

Théorème 16 *Toute formule possède une forme normale anti-prénexe unique.*

Par la suite nous montrerons (lemme 24) que si deux formules possèdent une forme prénexe commune, elles ont même forme normale anti-prénexe.

On peut vérifier sur l'exemple donné plus haut que les deux formules $\forall x\ \forall y\ \exists z\ (A(y,x) \wedge B(x,z))$ et $\forall x\ \exists z\ \forall y\ (A(y,x) \wedge B(x,z))$ ont bien la même forme anti-prénexe $\forall x\ (\forall y\ A(y,x) \wedge \exists z\ B(x,z))$. Nous disposons donc d'une forme normale unique modulo la commutativité des quantificateurs identiques et le renommage des variables liées. Attachons nous maintenant à la transformation des preuves pour ce critère de similitude.

III.3.4 Transformation des preuves pour des formules ayant la même forme normale anti-prénexe

L'objectif visé est toujours d'être capable de construire automatiquement la preuve d'une formule B à partir de celle d'une formule A lorsque A est B ont même forme normale. Nous avons décrit une transformation des formules qui permet d'obtenir une forme unique modulo la commutativité des quantificateurs. Nous ne nous préoccupons pas du renommage, c'est l'affaire de l'α-conversion des λ-termes. Nous allons, dans un premier temps, voir comment transformer la preuve d'une formule en la preuve de sa forme normale anti-prénexe.

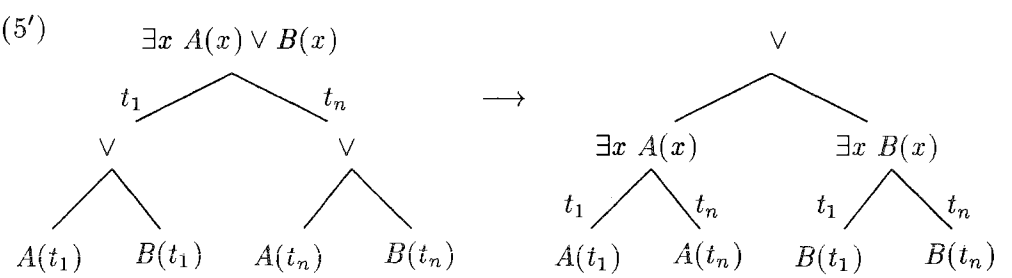
A De la forme prénexe vers la forme anti-prénexe

Puisque ces deux formules sont équivalentes, il est légitime de penser que la preuve de référence peut être réutilisée au maximum, en particulier, en réutilisant les mêmes sélections et expansions. Le problème est que le nombre de quantificateurs a pu augmenter par application de la propriété de distributivité. L'idée utilisée est simplement d'appliquer les mêmes règles dans l'arbre que dans la formule dans la plupart des cas. C'est ce que nous allons faire pour tous les cas sauf pour (4), (5) et (8) lorsque $Q = \exists$. Pour tous les autres cas, les règles s'appliquent sur les arbres comme sur les formules. Il suffit d'appliquer la règle sur la formule superficielle du nœud concerné. Nous ne les réécrivons donc pas. Intéressons-nous aux trois cas qui nécessitent un traitement particulier. Pour le cas (5), il s'agit simplement de dupliquer les expansions de la preuve lorsqu'un quantificateur existentiel est lui-même dupliqué. Par exemple, la règle (4) pour la formule donne :

$$(5) \qquad \exists x\ (A \vee B) \ \rightarrow\ \exists x(A) \vee \exists x\ (B)$$

Pour la preuve, nous appliquerons la règle :

(5′)

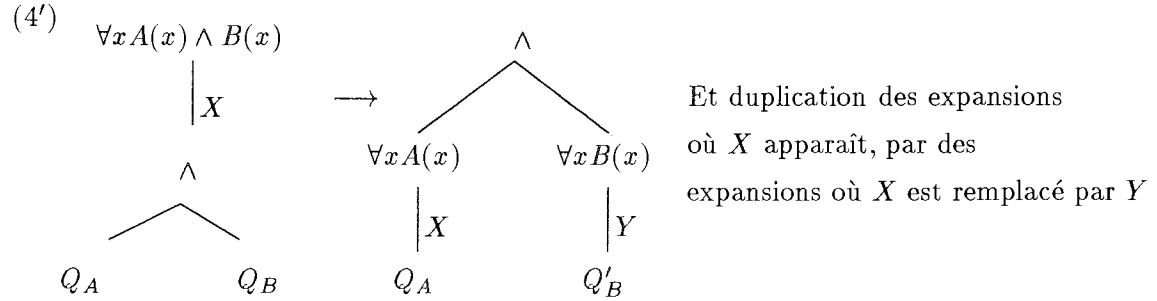


De cette façon, en appliquant les règles de mise en forme anti-prénexe à l'arbre, parallèlement à la formule, et en dupliquant les expansions en même temps que les quantificateurs, nous obtiendrons un arbre d'expansion pour la forme anti-prénexe de la formule. Le problème des quantificateurs universels est légèrement plus délicat. En effet, les sélections ne peuvent être

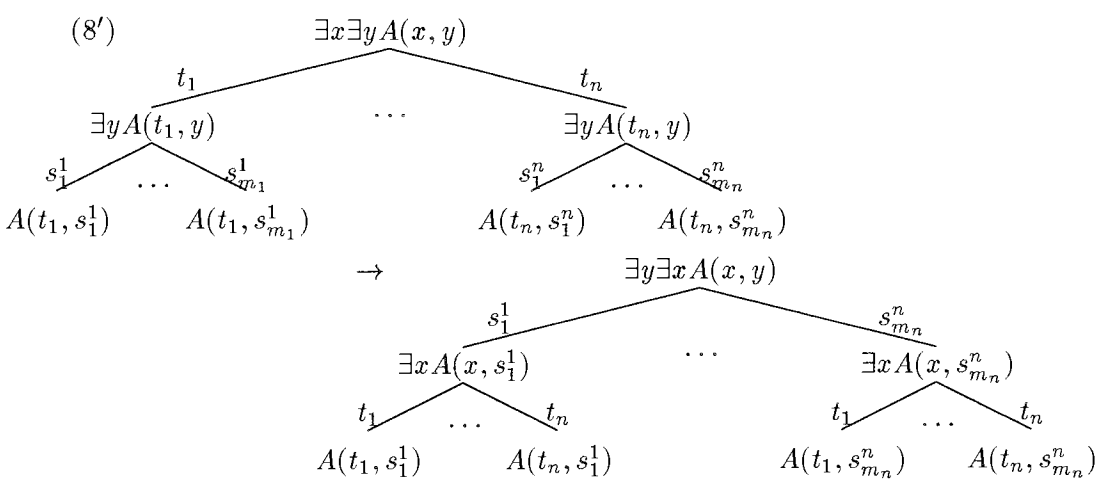
dupliquées puisqu'une variable libre ne peut être sélectionnée qu'une seule fois dans un arbre d'expansion. Cette condition est une garantie de correction, car sans elle, nous trouverions sans difficulté une preuve par expansion de la formule $\exists x\ A(x) \Rightarrow \forall x\ A(x)$ en sélectionnant la même variable libre pour les deux occurrences positives du quantificateur universel. Au lieu de dupliquer les sélections, nous allons garder la sélection X existante, dans une branche, sélectionner une nouvelle variable libre Y dans l'autre, et dupliquer toutes les expansions de l'arbre qui contiennent X par les mêmes expansions où X est remplacée par Y. La règle sur les formules s'exprime par :

$$(4) \qquad \forall x\ (A \wedge B) \ \rightarrow\ \forall x(A) \wedge \forall x\ (B)$$

nous avons, pour les arbres :



Voyons maintenant la règle (8). Du point de vue des nœuds de sélection, opérer les permutations correspondantes ne modifie en rien les propriétés de la preuve. En effet, si en descendant le long d'une branche d'arbre d'expansion, nous trouvons successivement le nœud $\forall x$ où X est sélectionnée pour x, puis le nœud $\forall y$ où Y est sélectionné pour y, inverser ces deux nœuds ne changera rien à la preuve. C'est pourquoi nous ne mentionnons le cas (8) que pour $Q = \exists$. Mais ce cas reste simple, même si sa représentation laisse penser le contraire. La règle (8') s'exprime par :



Ceci revient à effectuer en premier les expansions de y, puis, pour chacune d'elles, ré-appliquer les expansions que nous avions pour x. Ce qui ne perturbe pas la preuve, puisque que la relation $<_Q$ n'est pas modifiée, et comme nous n'avons fait que rajouté des feuilles, $\wedge\mathrm{cc}(Q) \preceq \wedge\mathrm{cc}(Q')$, puisque les clauses complètes sont des disjonctions des expansions.

Nous ne donnons pas les autres règles puisqu'elles sont identiques à celles utilisées pour les formules. Encore faut-il montrer que nous obtenons bien un arbre d'expansion vérifiant les bonnes propriétés, qu'il existe une connexion qui couvre cet arbre, et que la contrainte de précédence est respectée.

Définition 17 Mise en forme anti-prénexe d'un arbre d'expansion. Soit une formule A telle que $(Q, \mathcal{M}) \vdash A$. A toute application d'une règle de mise en forme anti-prénexe de cette formule, correspond une règle de transformation de l'arbre Q. Ces règles sont les mêmes que pour les formules, excepté dans les cas (3), (4) et (8) où les règles correspondantes sont respectivement (3'), (4') et (8') (pour $Q = \exists$ seulement). □

L'arbre obtenu représente la formule en forme anti-prénexe, montrons que les conditions sont réunies pour en faire une preuve par expansion.

Lemme 18 *Si $(Q, \mathcal{M}) \vdash A$, alors il existe une connexion $\mathcal{M}'$ telle que $(Q', \mathcal{M}') \vdash A'$ où A' est dérivée de A par application d'une règle de mise en forme anti-prénexe et Q' obtenu de Q par la règle correspondante.*

Preuve Regardons chacune des propriétés à remplir pour avoir $(Q', \mathcal{M}') \vdash A'$ à commencer par l'existence de $\mathcal{M}'$.

- Montrons que pour chaque règle, $\wedge\!\!\!\!\wedge cc(Q) \Rightarrow \wedge\!\!\!\!\wedge cc(Q')$. Pour les règles (1), (2), (3), il y a égalité. Nous verrons la règle (4) en dernier. Pour la règle (5'), (nous confondons les nœuds et la formule superficielle qui leur est associée) :
  $\wedge\!\!\!\!\wedge cc(Q) = \exists x\ (A \vee B) \vee \wedge\!\!\!\!\wedge cc(A(t_1)) \vee \wedge\!\!\!\!\wedge cc(B(t_1)) \vee \ldots \vee \wedge\!\!\!\!\wedge cc(A(t_n)) \vee \wedge\!\!\!\!\wedge cc(B(t_n))$,
  et $\wedge\!\!\!\!\wedge cc(Q') = \exists x\ A \vee \exists x\ B \vee \wedge\!\!\!\!\wedge cc(A(t_1)) \vee \ldots \wedge cc(A(t_n)) \vee \wedge\!\!\!\!\wedge cc(B(t_1)) \vee \ldots \vee \wedge\!\!\!\!\wedge cc(B(t_n))$.
  Il y a donc équivalence. Pour les règles (6) et (7), il y a équivalence également, respectivement aux conditions d'application. Pour la règle (8), $\wedge\!\!\!\!\wedge cc(Q) \Leftrightarrow \wedge\!\!\!\!\wedge cc(Q')$ puisque $\forall x\ y\ A(x, y) \Leftrightarrow \forall y\ x\ A(x, y)$. Pour la règle (8'), $\wedge\!\!\!\!\wedge cc(Q') \preceq \wedge\!\!\!\!\wedge cc(Q)$ car nous avons introduit de nouvelles feuilles, comme $A(t_2, s_1^1)$ par exemple. Ce qui ne fait que rajouter des éléments à la disjonction. Pour la règle (9), il y a égalité. Pour la règle (4'), nous n'avons pas d'équivalence. Globalement, la conjonction des clauses complètes de l'arbre s'écrit

$$\wedge\!\!\!\!\wedge cc(Q) = C * ((\forall x\ A \wedge B) \vee (\wedge\!\!\!\!\wedge cc(Q_A) \wedge \wedge\!\!\!\!\wedge cc(Q_B)))$$

$$= (C * ((\forall x\ A \wedge B) \vee \wedge\!\!\!\!\wedge cc(Q_A))) \wedge (C * ((\forall x\ A \wedge B) \vee \wedge\!\!\!\!\wedge cc(Q_B)))$$

  où C est la conjonction des clauses complètes du reste de l'arbre et $*$ le connecteur correspondant au nœud au dessus du nœud de sélection. On a, pour Q' :

$$\wedge\!\!\!\!\wedge cc(Q') = (C \vee C[X \mapsto Y]) * (\forall x\ A \wedge B \vee (\forall x\ A \vee \wedge\!\!\!\!\wedge cc(Q_A)) \wedge (\forall x\ B \vee \wedge\!\!\!\!\wedge cc(Q_B)[X \mapsto Y]))$$

$$= ((C \vee C[X \mapsto Y]) * (\forall x\ A \wedge B \vee (\forall x\ A \vee \wedge\!\!\!\!\wedge cc(Q_A))))$$

$$\wedge((C \vee C[X \mapsto Y]) * (\forall x\ A \wedge B \vee (\forall x\ B \vee \wedge\!\!\!\!\wedge cc(Q_B)[X \mapsto Y])))$$

  Or, $C * ((\forall x\ A \wedge B) \vee \wedge\!\!\!\!\wedge cc(Q_A)) \preceq (C \vee C[X \mapsto Y]) * (\forall x\ A \wedge B \vee (\forall x\ A \vee \wedge\!\!\!\!\wedge cc(Q_A))$, et)

$$(C * ((\forall x\ A \wedge B) \vee \wedge\!\!\!\!\wedge cc(Q_B)))[X \mapsto Y] \preceq$$

$$(C \vee C[X \mapsto Y]) * (\forall x\ A \wedge B \vee (\forall x\ B \vee \curlywedge\mathrm{cc}(Q_B)[X \mapsto Y]))$$

On a donc bien l'implication $\curlywedge\mathrm{cc}(Q) \Rightarrow \curlywedge\mathrm{cc}(Q')$. Par conséquent, il existe une connexion qui couvre le nouvel arbre. En effet, puisque $\curlywedge\mathrm{cc}(Q)$ est une tautologie, $\curlywedge\mathrm{cc}(Q')$ en est également une. Par conséquent, dans chacune des clauses complètes, il existe un couple A $\neg A$.

- La propriété pour les variables sélectionnées n'apparaissant pas dans la formule superficielle n'a pas été modifiée.
- En ce qui concerne la relation de dépendance sur les termes expansés, seule la règle (4) peut poser problème. Mais là où nous avions $f(X) <_Q t$, nous aurons maintenant $f(X) <_{Q'} t$ et $f(Y) <_{Q'} t$. Mais comme Y est un nouveau paramètre, si $<_{Q'}$ est acyclique, $<_Q$ l'est aussi, ce qui est contraire à l'hypothèse selon laquelle $(Q, \mathcal{M})$ est une preuve par expansion. De plus, il ne peut exister de relation du type $f(X) <_{Q'} f(Y)$ ou $f(Y) <_{Q'} f(X)$, sans cela, nous aurions $f(X) <_Q f(X)$.

Le fait que $Q'^s = A'$ est trivial. Par conséquent, nous pouvons affirmer qu'il existe $\mathcal{M}'$ tel que $(Q', \mathcal{M}') \vdash A'$. □

Par extension, en transformant A en sa forme anti-prénexe A', nous pouvons transformer Q tel que $(Q, \mathcal{M}) \vdash A$ en Q' tel qu'il existe une connexion $\mathcal{M}'$ pour $(Q', \mathcal{M}') \vdash A'$. Nous sommes donc capables de transformer la preuve d'une formule en la preuve de sa forme normale anti-prénexe.

B Permutation des quantificateurs

Réglons maintenant le problème des permutations de quantificateurs. Comme nous l'avons déjà dit, la formule $\forall x\ A(x)$ se représente par le λ-terme $\forall(\lambda x.(A\ x))$. Lorsque deux quantificateurs identiques se succèdent, nous avons une formule du type : $\forall x\ \forall y\ A(x, y)$ représentée par $\forall(\lambda x.\forall(\lambda y.((A\ y)\ x)))$, et non par $\forall(\lambda x.\forall(\lambda y.(A\ y))\ x)$ où x n'est plus dans le champ du $\forall\ y$. Cette petite modification du parenthésage signifie simplement que la variable, abstraite par le lieur le plus extérieur (x), ne peut pas apparaître au dessus du lieur de l'autre variable (y). De même, si nous avons une succession de n quantificateurs identiques : $\forall x_1 \cdots \forall x_n\ A(x_1, \ldots, x_n)$, cette formule sera représentée par le terme

$$\forall(\lambda x_1.(\ldots\forall(\lambda x_n.(\cdots((A\ x_n)x_{n-1})\cdots x_1)))),$$

où aucune des variables x_i ne peut apparaître au dessus de l'abstraction λx_n. Nous nous permettons donc d'utiliser la représentation suivante pour cette formule :

$$\forall(\lambda x_1, \ldots, x_n.(\cdots((A\ x_n)x_{n-1})\cdots x_1)),$$

ou plutôt, conformément à nos notations usuelles,

$$\forall(\lambda x_1, \ldots, x_n.A(x_1, \ldots, x_n))$$

Par conséquent, supposons que nous avons deux formules en forme anti-prénexe. Pour savoir si elles sont identiques modulo la permutation des quantificateurs, il suffit de permuter les variables liées lorsque le terme (ou un sous-terme) est de la forme

$$Q(\lambda x_1, \ldots, x_n.A(x_1, \ldots, x_n)).$$

Ce qui signifie que pour prendre en compte la permutation des quantificateurs du point de vue de la logique – c'est à dire des formules – nous considérons que les deux termes

$$Q(\lambda x.(\lambda y.((f\ y)\ x))) \text{ et } Q(\lambda y.(\lambda x((f\ y)\ x)))$$

sont équivalents où Q est un quantificateur (à condition tout de même que x et y soient de même type). La reconnaissance de cette relation entre deux termes doit être assumée par le filtrage. Mais ce n'est pas le cas pour l'instant. Aussi, pour deux formules en forme anti-prénexe, faut-il essayer toutes les permutations pour savoir si ces formules sont en relation par ce critère. Quant à la preuve, il suffit de réutiliser la règle (8′) du paragraphe précédent.

A présent, nous avons une méthode permettant de transformer une formule en sa forme anti-prénexe, et parallèlement, une transformation de la preuve associée. Il nous faut maintenant procéder au processus inverse. En effet, si l'on suppose que notre formule de référence est en forme anti-prénexe et notre formule objet en forme prénexe, nous devons pouvoir déduire une preuve de cette dernière. Il faut donc construire une méthode inverse de la première, c'est à dire faire correspondre à chaque transformation de mise en forme prénexe, une transformation de la preuve qui conserve les bonnes propriétés. En plus de la mise en forme prénexe, il faudra envisager l'élimination de quantificateurs correspondant à l'opération inverse des règles (3) et (4) de la mise en forme anti-prénexe. Dans le cas général, une formule qui n'est pas dans sa forme anti-prénexe peut être obtenue de sa forme anti-prénexe, par application de règles de mise en forme prénexe. Par conséquent, si nous connaissons la séquence de règles qui mène à cette formule, nous serons capables d'en fournir automatiquement la preuve.

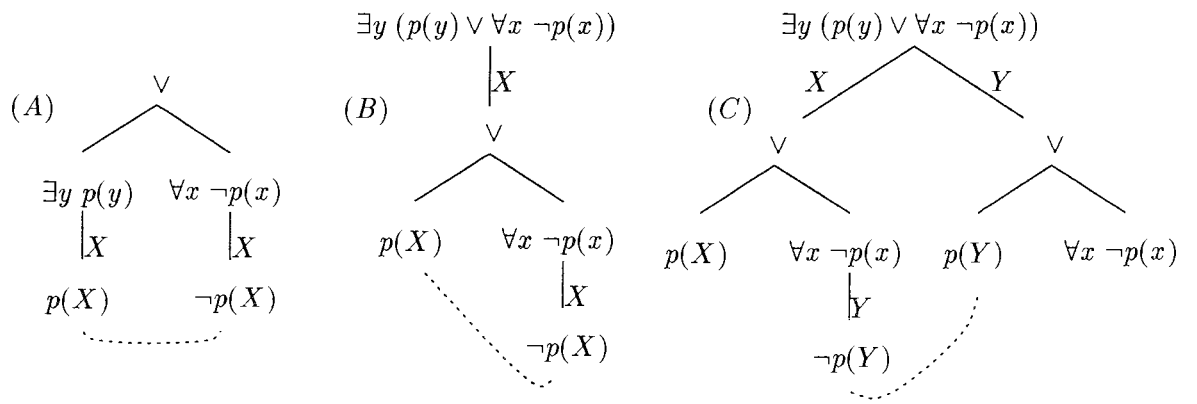
C De la forme anti-prénexe vers une forme prénexe

Nous considérons à présent que nous avons une preuve par expansion $(Q, \mathcal{M})$ d'une formule A, et B, la formule objet qui peut être obtenue de A par application d'une règle de mise en forme prénexe. Par conséquent, A et B ont la même forme anti-prénexe, que l'on nommera A_0. Ce qui précède nous permet de dire que nous avons construit une preuve $(Q_0, \mathcal{M}_0) \vdash A_0$. Il s'agit à présent de modifier cette preuve afin d'obtenir $(Q', \mathcal{M}') \vdash B$. Or, B peut être obtenu de A_0 par application des règles :

(1)	$(\exists x\ A(x)) \wedge B$	$\rightarrow$	$\exists x\ (A(x) \wedge B)$	$x \notin \mathcal{FV}(B)$
(2)	$(\exists x\ A(x)) \vee B$	$\rightarrow$	$\exists x\ (A(x) \vee B)$	$x \notin \mathcal{FV}(B)$
(3)	$(\forall x\ A(x)) \wedge B$	$\rightarrow$	$\forall x\ (A(x) \wedge B)$	$x \notin \mathcal{FV}(B)$
(4)	$(\forall x\ A(x)) \vee B$	$\rightarrow$	$\forall x\ (A(x) \vee B)$	$x \notin \mathcal{FV}(B)$
(5)	$\exists x\ \exists y(A(x) \vee B(y))$	$\rightarrow$	$\exists x\ (A(x) \vee B(x))$	$x \notin \mathcal{FV}(B(y))$ et $y \notin \mathcal{FV}(A(x))$
(6)	$\forall x\ \forall y(A(x) \wedge B(y))$	$\rightarrow$	$\forall x\ (A(x) \wedge B(x))$	$x \notin \mathcal{FV}(B(y))$ et $y \notin \mathcal{FV}(A(x))$

qui forment la mise en forme prénexe, plus l'éventuel effacement de quantificateurs qui ne figure pas, en général dans les règles classiques (en particulier les règles (5) et (6)). Nous allons donc proposer une transformation de preuves correspondant à chacune de ces règles de transformation de formules, et montrer qu'elles sont correctes. Les cas (3) à (6) se traitent relativement bien. Les règles de transformations sont représentées figure III.3. Par contre, un problème spécifique est introduit pour la remontée d'un quantificateur existentiel. Ce problème est relatif à la précédence sur les termes expansés. Pour l'expliquer, prenons un exemple. La preuve du théorème

$\exists y\ p(y) \vee \forall x\ \neg p(x)$ s'exprime facilement par la preuve étiquetée (A) ci-dessous. Par contre, la preuve étiquetée (B) pour la même formule où le quantificateur existentiel a été remonté n'est pas correcte, car elle introduit un cycle dans la précédence. En effet, on a alors $X <_Q X$. Par contre, (C) est une preuve correcte de cette formule. On a simplement pour la précédence, $X <_Q Y$.



Pour le cas de figure $\exists x\ (A(x)) * B \rightarrow \exists x\ (A(x) * B)$, la règle correspondante pour l'arbre sera en général la règle (1) ou (2) selon la nature de l'opérateur $*$. Mais nous devons introduire le cas particulier suivant décrit dans la figure III.2.

Définition 19 (figure III.2)
□

Cette transformation reflète la transformation de la preuve (A) en la preuve (C). Avant de poursuivre, nous allons montrer que cette transformation est correcte, puisqu'il s'agit de la principale difficulté de la mise en forme prénexe. Nous donnerons ensuite les règles qui régissent le cas général, montrerons leur correction, et nous donnerons un exemple.

Lemme 20 *Soit* $(Q, \mathcal{M}) \vdash F$, *et* $F \rightarrow F'$ *par la règle* $\exists x\ (A(x)) * B \rightarrow \exists x\ (A(x) * B)$. *La règle énoncée ci-dessus transforme* Q *en* Q' *tel qu'il existe* $\mathcal{M}'$ *vérifiant* $(Q', \mathcal{M}') \vdash F'$.

Preuve Les conditions sur les variables libres de F et les formules superficielles sont respectées. La difficulté réside dans l'existence de la connexion et la préservation de la condition sur la précédence $<_Q$. Justifions d'abord l'existence d'une connexion couvrant Q'. Pour alléger les notations, prenons $A = \wedge\mathrm{cc}(Q_1)$, $B = \wedge\mathrm{cc}(Q_B)$ et $C = \wedge\mathrm{cc}(Q_4)$. Nous allons montrer l'implication entre la conjonction des clauses complètes de Q et de Q', selon la valeur du connecteur $*$. Rappelons que X et Y sont des paramètres.

- $* = \vee$ alors,

$$\wedge\mathrm{cc}(Q) = (A \vee B) \vee C \text{ et}$$

$$\wedge\mathrm{cc}(Q') = A \vee B[X \mapsto Y] \vee A[X \mapsto Y] \vee C \vee C[X \mapsto Y]. \text{ Ou encore,}$$

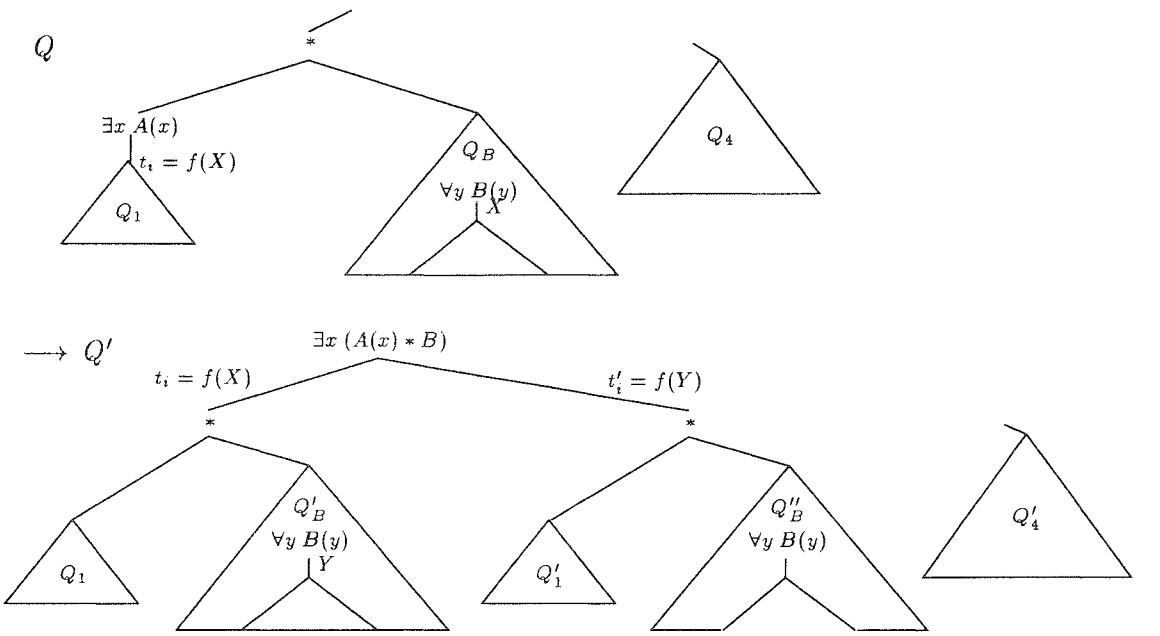
$$\wedge\mathrm{cc}(Q') = A \vee C \vee A[X \mapsto Y] \vee B[X \mapsto Y] \vee C[X \mapsto Y].$$

$$\text{D'où } \wedge\mathrm{cc}(Q) = (A \vee B \vee C) \Rightarrow (A[X \mapsto Y] \vee B[X \mapsto Y] \vee C[X \mapsto Y]) \Rightarrow \wedge\mathrm{cc}(Q')$$

Soient $\{\overline{t_n}\}$ les termes d'expansion sous le quantificateur existentiel à remonter. S'il existe $t_i \in \{\overline{t_n}\}$ tel que le paramètre X apparaît dans t_i et X est sélectionné dans Q_B,

- remonter le nœud d'expansion au sommet,
- substituer la sélection de X dans Q_B par une sélection de Y, une nouvelle variable libre,
- ajouter une expansion t'_i telle que $t'_i = t_i[X \mapsto Y]$ sous laquelle la sélection de X dans Q_B est effacée de façon à ce que le nœud de sélection devienne une feuille.
- Dans le reste de l'arbre (Q_4), dupliquer les expansions où X apparaît, avec des expansions où Y est substitué à X.

Cette opération est représentée par la figure :



S'il n'existe pas de tel t_i, les règles (1) ou (2) de la figure ?? (selon la nature de $*$) sont appliquées normalement.

Figure III.2: Remontée d'un quantificateur existentiel

- $* = \wedge$ alors,

$$\wedge\text{cc}(Q) = (A \vee B) \wedge C \text{ et}$$

$$\wedge\text{cc}(Q') = (A \vee B[X \mapsto Y] \vee A[X \mapsto Y]) \wedge C \vee C[X \mapsto Y]. \text{ Ou encore,}$$

$$\wedge\text{cc}(Q') = D \vee ((A[X \mapsto Y] \vee B[X \mapsto Y]) \wedge C[X \mapsto Y]).$$

$$\text{D'où } \wedge\text{cc}(Q) = ((A \vee B) \wedge C) \Rightarrow ((A[X \mapsto Y] \vee B[X \mapsto Y]) \wedge C[X \mapsto Y]) \Rightarrow \wedge\text{cc}(Q')$$

Il existe donc une connexion qui couvre Q' s'il existe une connexion qui couvre Q.

Pour la précédence, l'opération avait pour but de la conserver. Nous avons seulement introduit dans cette précédence, la relation $f(Y) <_{Q'} f(X)$ ce qui ne peut la rendre cyclique puisque

Y est un nouveau paramètre et que X n'est pas sélectionné sous $f(Y)$. □

La même opération est à renouveler pour tous les t_i où apparaît une variable sélectionnée dans Q_B. S'il existe un t_i dans lequel apparaissent plusieurs paramètres sélectionnés dans Q_B, le principe est le même. Nous allons donner les règles pour le cas général, avant de montrer leur correction, puis un exemple de remontée des quantificateurs existentiels. Pour le cas général, la figure III.3 donne les règles de transformation à appliquer à la preuve d'une formule parallèlement à sa transformation en forme prénexe.

Pour le cas (6), $Q[X \mapsto Z\ ;\ Y \mapsto Z]$ signifie que les paramètres X et Y sont remplacés par un nouveau paramètre Z dans tout l'arbre Q – dans les expansions puisque c'est le seul endroit où elle peuvent apparaître.

Définition 21 Mise en forme prénexe d'un arbre d'expansion. Soit une formule A telle que $(Q, \mathcal{M}) \vdash A$. Pour toute application d'une règle de mise en forme prénexe de cette formule, correspond une règle de transformation de l'arbre A. Ces règles sont celles de la figure III.3, plus le cas particulier considéré par la figure III.2. □

Voyons à présent si l'arbre obtenu, permet d'obtenir une preuve par expansion de la formule transformée.

Lemme 22 *Soit A une formule telle que $(Q, \mathcal{M}) \vdash A$. Pour toute règle de transformation de formule en forme prénexe $A \to A'$, soit $Q \to Q'$ l'application de la règle correspondante pour la transformation de preuves, il existe une connexion $\mathcal{M}'$ telle que $(Q', \mathcal{M}') \vdash A'$.*

Preuve $Q^s = A$ implique $Q'^s = A'$ est évident. Il suffit de comparer les formules superficielles aux racines des arbres et les formules. Elles sont identiques. S'il n'y a pas de variable sélectionnée dans Q, qui soit libre dans A, il en est de même dans Q', puisque la seule variable sélectionnée introduite (dans les règles générales) est c dans la règle (6) et qu'elle est nouvelle. L'autre cas d'introduction de sélection est couvert par le lemme 20. D'autre part, le seul cas où la précédence $<_Q$ est perturbée est également décrit dans le lemme 20. Il suffit donc de montrer qu'il existe une connexion qui couvre le nouvel arbre d'expansion.

- Dans les cas (1) et (2), $\models \wedge\mathrm{cc}(Q) \Rightarrow \wedge\mathrm{cc}(Q')$, puisque nous avons simplement distribué $\wedge\mathrm{cc}(Q_B)$. On a $\wedge\mathrm{cc}(Q) = (\exists x\ A(x) \vee \wedge\mathrm{cc}(Q_1) \vee \wedge\mathrm{cc}(Q_n)) * \wedge\mathrm{cc}(Q_B)$, et $\wedge\mathrm{cc}(Q') = (\exists x\ A(x) * B) \vee (\wedge\mathrm{cc}(Q_1) * \wedge\mathrm{cc}(Q_B)) \vee \ldots \vee (\wedge\mathrm{cc}(Q_n) * \wedge\mathrm{cc}(Q_B))$. L'implication est donc valide, pour $* = \vee, \wedge$. Il convient d'ajouter ici le lemme 20 pour le cas particulier qu'il décrit.

- Dans les cas (3) et (4), nous avons $\wedge\mathrm{cc}(Q) = (\forall x\ A(x) \vee \wedge\mathrm{cc}(Q_0)) * \wedge\mathrm{cc}(Q_B)$, et $\wedge\mathrm{cc}(Q') = (\forall x\ A(x) * B) \vee (\wedge\mathrm{cc}(Q_0) * \wedge\mathrm{cc}(Q_B))$. L'implication est également valide ici, pour $* = \vee, \wedge$.

- Dans le cas (5), $\wedge\mathrm{cc}(Q')$ est $\wedge\mathrm{cc}(Q)$ où a été rajoutée la disjonction des $A(s_i)$ et $B(t_j)$ pour $i = 1, .., n$ et $j = 1, .., m$, qui n'apparaissaient pas dans Q. Pour rester lisible, nous confondons le nœud et la formule superficielle qui lui est associée. Nous avons :

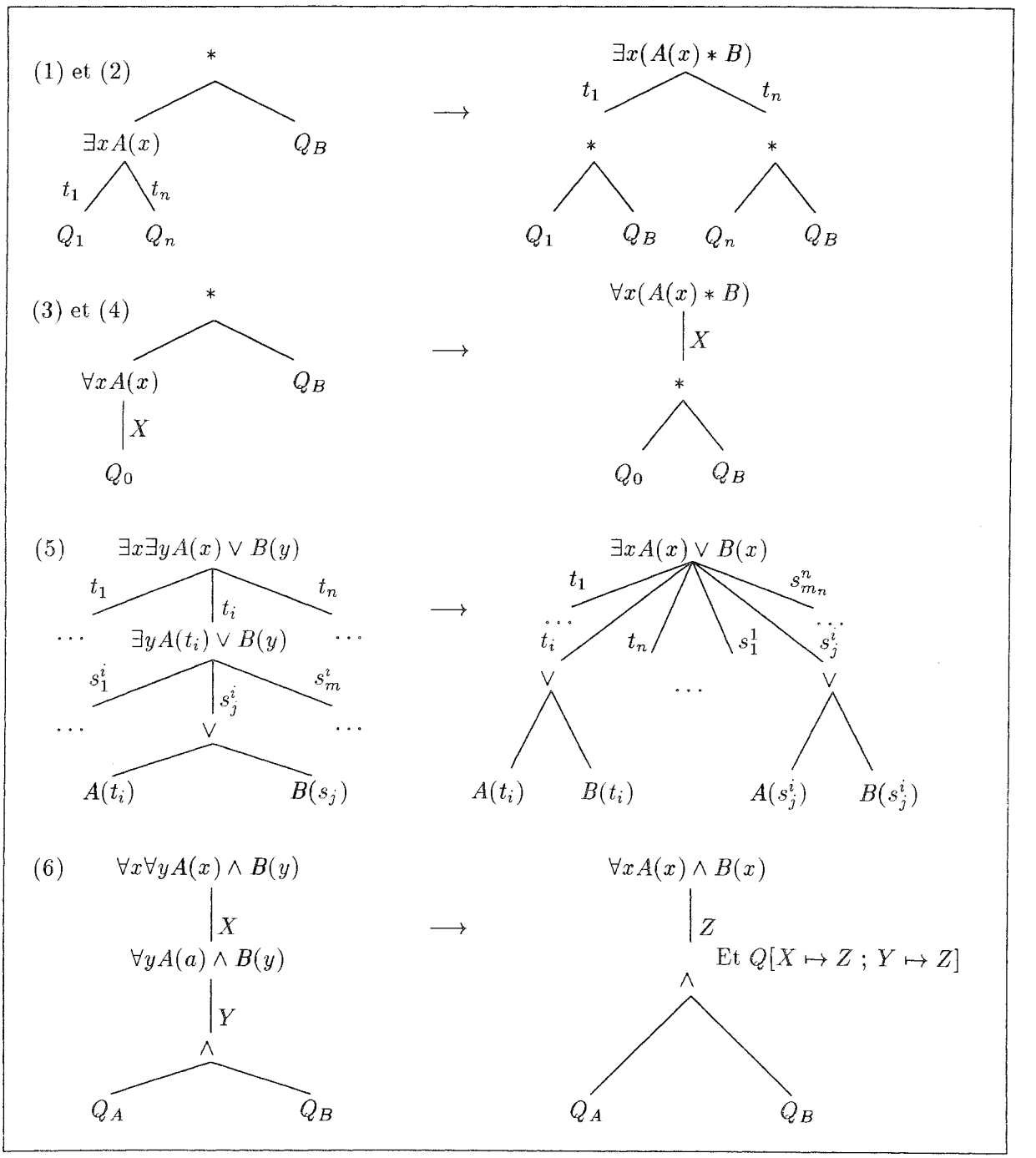


Figure III.3: Transformation anti-prénexe $\rightarrow$ prénexe

$$\begin{aligned}
\curlywedge\mathrm{cc}(Q) &= (\curlywedge\mathrm{cc}(A(t_1)) \vee (\curlywedge\mathrm{cc}(B(s_1^1)) \vee \ldots \vee \curlywedge\mathrm{cc}(B(s_{m_1}^1))))\\
&\quad \vee(\curlywedge\mathrm{cc}(A(t_2)) \vee (\curlywedge\mathrm{cc}(B(s_1^2) \vee \ldots \vee \curlywedge\mathrm{cc}(B(s_{m_2}^2))))\\
&\quad \vdots\\
&\quad \vee(\curlywedge\mathrm{cc}(A(t_n)) \vee (\curlywedge\mathrm{cc}(B(s_1^n) \vee \ldots \vee \curlywedge\mathrm{cc}(B(s_{m_n}^n))))
\end{aligned}$$

et

$$\begin{aligned}
\curlywedge\mathrm{cc}(Q') &= \curlywedge\mathrm{cc}(Q) \vee A(s_1^1) \vee \ldots \vee A(s_{m_n}^n) \vee A(t_1) \vee \ldots \vee A(t_n)\\
&\quad B(s_1^1) \vee \ldots \vee B(s_{m_n}^n) \vee B(t_1) \vee \ldots \vee B(t_n)
\end{aligned}$$

où les $A(s_j^i)$ et les $B(t_i)$ sont des feuilles qui n'apparaissent pas dans Q. L'implication est toujours valide puisque $\curlywedge\mathrm{cc}(Q) \preceq \curlywedge\mathrm{cc}(Q')$.

- Pour le cas (6), nous avons

$$\curlywedge\mathrm{cc}(Q) = \forall xy\ (A(x) \wedge B(y)) \vee (\curlywedge\mathrm{cc}(Q_A) \wedge \curlywedge\mathrm{cc}(Q_B)) \text{ et}$$

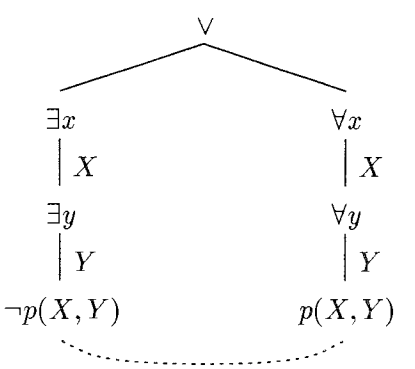
$$\curlywedge\mathrm{cc}(Q') = \forall x\ (A(x) \wedge B(x)) \vee \curlywedge\mathrm{cc}(Q_A)[X \mapsto Z\ ;\ Y \mapsto Z] \wedge \curlywedge\mathrm{cc}(Q_B)[X \mapsto Z\ ;\ Y \mapsto Z]$$

d'où $\curlywedge\mathrm{cc}(Q) \Rightarrow \curlywedge\mathrm{cc}(Q')$.

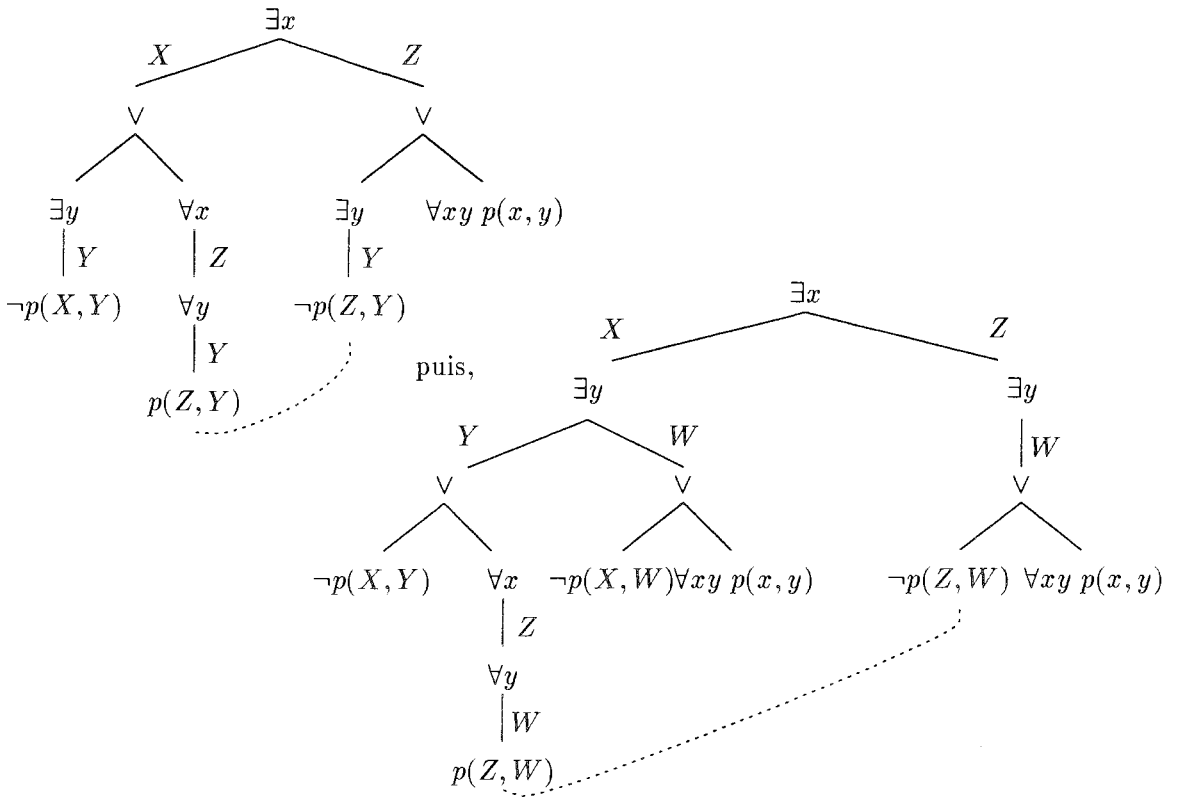
Il existe donc, pour chaque transformation de preuve, une connexion $\mathcal{M}'$ telle que $(Q', \mathcal{M}') \vdash A'$. □

Donnons un exemple de remontée de quantificateurs existentiels dans une preuve.

Exemple 23 Considérons la formule $A = \exists x\ \exists y\ \neg p(x,y) \vee \forall x\ \forall y\ p(x,y)$ et sa preuve, pour transformer cette dernière en preuve de $B = \exists x\ \exists y\ (\neg p(x,y) \vee \forall x\ \forall y\ p(x,y))$. Pour la preuve de A, nous avons simplement



Les paramètres X et Y sont des termes d'expansion à gauche, et des variables de sélection à droite. Aussi, va t-on avoir, successivement en remontant le premier, puis le second quantificateur existentiel :



La précédence obtenue est $Y <_{Q'} W$ et $W <_{Q'} Z$. Il n'y a donc pas de cycle, et la connexion $(p(Z,W), \neg p(Z,W))$ suffit à couvrir Q'. ■

Cet exemple montre également que pour une preuve par expansion, il peut être intéressant d'utiliser la forme anti-prénexe de la formule. Ainsi, il existe moins de contraintes entre les termes d'expansions et les variables sélectionnées, puisqu'il apparaissent plus souvent à des positions indépendantes.

III.4 Combinaison des critères de similitude

Nous avons étudié trois sortes de similitudes. La similitude par correspondance, la similitude modulo les propriétés AC des connecteurs logiques, et la similitude par rapport à certaines propriétés de quantificateurs. La combinaison des deux premières ne pose pas de problème, comme nous l'avons vu en introduisant la similitude modulo AC, puisque le traitement de l'associativité-commutativité est intégré dans le filtrage d'ordre deux. La combinaison de ces trois critères de similitude se ramène donc à la combinaison des deux derniers. Soit A et B, deux formules telles que B est similaire à A en considérant à la fois l'AC-correspondance, et les propriétés énoncées pour les quantificateurs. Pour considérer la combinaison de ces deux similitudes, il suffit de les considérer séparément. Si nous mettons A et B sous leur forme anti-prénexe dans un premier temps, et que nous considérons alors la correspondance AC, si cette correspondance AC existe, il permet de transformer la forme anti-prénexe de A en la forme anti-prénexe de B. Alors, partant d'une preuve de A, nous pouvons la transformer en une preuve de sa forme anti-prénexe, lui appliquer la correspondance, et enfin la transformer en une preuve de B.

Comme nous l'avons dit lorsque nous avons présenté notre approche de l'analogie (cf. figure I.1), c'est de la relation entre les formules, dont nous nous servons pour transformer les preuves. Ceci est très clair dans le cas qui nous occupe actuellement. La relation qui existe entre les formules A et B peut être symbolisée de la façon suivante :

$$\begin{array}{ccc} A & & B \\ \downarrow & & \downarrow \\ A_0\text{: forme anti-prénexe de } A & \xrightarrow{AC\text{-correspondance}} & B_0\text{: forme anti-prénexe de } B \end{array}$$

On voit alors que les transformations appliquées aux preuves suivent exactement le même schéma :

$$\begin{array}{ccc} P(A) & & P(B) \\ \downarrow & & \uparrow \\ P(A_0)\text{: forme anti-prénexe de } P(A) & \xrightarrow{AC\text{-correspondance}} & P(B_0)\text{: forme anti-prénexe de } P(B) \end{array}$$

Pour confirmer ceci, nous montrons que lorsque deux formules sont en relation par AC-correspondance et modulo les propriétés énoncées pour les quantificateurs, leurs formes anti-prénexes sont en AC-correspondance. Pour cela, nous montrons en premier lieu que lorsque deux formules ont une forme prénexe commune, elles ont même forme normale anti-prénexe.

Lemme 24 *Soient A B et C trois formules telles que C peut être dérivée de A et de B par application des règles de mise en forme prénexe. Alors, ces trois formules ont même forme anti-prénexe.*

Preuve Par récurrence sur la taille de C.

Cas de base : C est un littéral. Dans ce cas, $A = B = C$ et elles sont toutes les trois en forme anti-prénexe.

Récurrence : nous examinons le quantificateur le plus haut de C, en supposant que les sous-termes ont même forme anti-prénexe.

- $C = \exists x\ (C_1(x) \wedge C_2)$ avec $x \notin \mathcal{FV}(C_2)$. Dans ce cas, C est obtenue, de $A = \exists x\ (A_1(x) \wedge A_2)$ et $B = \exists x\ (B_1(x)) \wedge \exists x\ B_2$, avec par hypothèse de récurrence, $A_1(x)$, $B_1(x)$ et $C_1(x)$ ayant même forme normale anti-prénexe, ainsi que A_2, B_2 et C_2. Or, puisque $x \notin \mathcal{FV}(C_2)$, $x \notin \mathcal{FV}(B_2)$, et la forme anti-prénexe de $\exists x\ B_2$ est la forme anti-prénexe de B_2. La propriété est donc conservée dans ce cas.

- Les autres cas de figures pour C sont tout à fait semblables.

□

Montrons maintenant que lorsque deux formules sont en relation modulo l'AC-correspondance et les propriétés définies pour les quantificateurs, leur formes anti-prénexes sont en AC-correspondance (modulo la permutation des quantificateurs que l'on ne considère pas ici).

Lemme 25 *Soient A et B deux formules telles que :*

$$A \xleftrightarrow{AC\text{-correspondance + prop. des quantificateurs}} B$$

alors, pour A_0 et B_0 leurs formes anti-prénexes respectives,

$$A_0 \xleftarrow{AC\text{-}correspondance} B_0$$

Preuve

$$A \xleftarrow{AC\text{-correspondance + prop. des quantificateurs}} B$$

signifie que A et B peuvent s'écrire, à l'aide des règles de mise en forme prénexe ou anti-prénexe, sous la forme A' et B' telles que A' et B' se correspondent par AC-correspondance. Or dans ce cas, soit A est une forme prénexe de A', soit l'inverse. Mais par le lemme 24, nous savons qu'elles ont toutes les deux la même forme anti-prénexe A_0. De même pour B : B' est une forme prénexe de B ou B est une forme prénexe de B', mais dans les deux cas, elles ont même forme anti-prénexe B_0. Le problème revient donc à savoir si, lorsque

$$A' \xleftarrow{AC\text{-correspondance}} B'$$

ont conserve cette propriété par mise en forme anti-prénexe, pour obtenir

$$A_0 \xleftarrow{AC\text{-correspondance}} B_0$$

Lorsque A' et B' sont en AC-correspondance, il existe une correspondance ρ telle que $\rho(A') =_{AC} \rho(B')$. Il faut montrer que cette correspondance est conservée par application des règles de mise en forme anti-prénexe. La seule difficulté, pour cela est de considérer globalement les disjonctions et les conjonctions pour les règles (4), (5), (6) et (7).

Prenons par exemple $A' = \forall x\ (A_1 \wedge A_2 \wedge A_3)$ et $B' = \forall x\ (B_1 \wedge B_3 \wedge B_2)$ avec ρ tel que $\rho(A') =_{AC} B'$, et A_i en AC correspondance avec B_i pour $i = 1, 2, 3$. Si x apparaît dans A_1 et A_2 (resp. B_1 et B_2). La mise en forme normale anti-prénexe va tout d'abord donner pour A' : $A' \rightarrow_{(6)} \forall x\ (A_1 \wedge A_2) \wedge A_3$, et pour B' : $B' \rightarrow_{(4)} \forall x\ (B_1 \wedge B_3) \wedge \forall x\ B_2$ mais la AC-correspondance est rétablie à l'étape suivante par : $A' \rightarrow_{(6)(4)} \forall x\ (A_1) \wedge \forall x\ (A_2) \wedge A_3$, et $B' \rightarrow_{(4)(6)} \forall x\ (B_1) \wedge B_3 \wedge \forall x\ (B_2)$. Et le même raisonnement peut être tenu pour chacune de ces règles. □

Pour ce qui est de la façon dont la similitude entre les formules est reconnue, nous utilisons deux techniques assez différentes, dont l'une peut être qualifiée de faible, et l'autre de forte. En effet, les propriétés des quantificateurs sont prises en comptes pour des formules, que l'on peut appeler figées. C'est-à-dire qu'à ce stade, il n'est plus possible d'utiliser les propriétés AC des connecteurs ou un renommage, par exemple. Par contre, pour la relation AC, la reconnaissance est intégrée au filtrage. Cette technique est donc plus forte, car elle est capable de considérer les deux critères – AC et correspondance –simultanément et surtout, sans manipulation des formules. L'idéal serait de pouvoir en faire autant avec les propriétés des quantificateurs. Ainsi, la méthode serait plus générale et la reconnaissance se ferait en une seule opération. Mais nous sommes incapables de dire actuellement si l'intégration de ces critères au filtrage est réalisable. Si la décidabilité du filtrage modulo ces propriétés est un problème difficile, nous pouvons envisager d'introduire de nouveaux critères de similitudes de la façon faible. Par exemple, en introduisant un mécanisme de transformation des motifs $\neg\neg A$ en A. L'introduction d'un tel traitement n'est pas vraiment intéressant dans le cadre des preuves par expansion, puisque les formules sont en FNN. Mais il peut l'être dans d'autres formalismes. Ce type de traitement doit pouvoir se faire

comme nous traitons les quantificateurs, même si, encore une fois, l'idéal serait de l'intégrer directement au filtrage.

Au terme de cette combinaison des critères de similitudes, nous avons considérablement élargit le cadre de l'analogie automatique. Mais si nous sommes loin de l'équivalence syntaxique, les deux formules considérées sont équivalentes du point de vue de la logique, modulo la correspondance. Aussi doit-il être possible de considérer des formules réellement différentes – et non plus similaires – pour pratiquer l'analogie. C'est ce que nous ferrons, dans le chapitre IV, puis dans un cadre beaucoup plus général, au chapitre V. Mais en premier lieu, donnons un exemple d'application de l'analogie pour des formules similaires.

III.5 Exemple d'application

Puisqu'il existe une correspondance bien connue entre les preuves en déduction naturelle et le typage des λ-termes, nous nous proposons de montrer ici, que les preuves dans l'un des formalismes peuvent être automatiquement transformées en preuves dans l'autre formalisme. Nous allons donc donner un système de déduction naturelle pour la logique propositionnelle, ainsi qu'un système de typage simple pour les λ-termes, et nous allons coder ces deux systèmes par des formules de la logique du premier ordre. Ensuite, nous verrons qu'une preuve de déduction naturelle peut être automatiquement transformée en une preuve de typage. Les formules prouvées étant "correspondantes" au sens intuitif. Le but de cet exemple est de montrer que, puisque l'isomorphisme de Curry-Howard dit qu'il y a correspondance entre les preuves de déduction naturelle et les preuves de typage, il est possible de pratiquer l'analogie entre ces deux types de preuves. Mais il ne s'agit que d'une exploitation de l'isomorphisme de Curry-Howard, dans le sens où nous ne montrons pas que toute preuve de déduction naturelle se traduit en une preuve de typage, mais que nous pouvons appliquer notre notion de similitude entre les preuves relatives à ces deux systèmes.

III.5.1 Un système de déduction naturelle pour le calcul propositionnel

Comme nous utilisons la logique du premier ordre, nous pouvons seulement coder le calcul propositionnel. Nous allons donner un système de déduction naturelle pour ce système, construisant un terme de preuve. C'est à dire qu'au lieu d'avoir une requête du type $\vdash_N P$ où P est une proposition (et l'indice N signifie qu'il s'agit de déduction naturelle), nous pourrons formuler la requête $\exists x\ (\vdash_N x \bullet P)$ où $x \bullet P$ signifie que x est un terme représentant une preuve de la proposition P.

Ainsi, la règle $I_\supset \dfrac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \supset B}$ devient $I_\supset \dfrac{\Gamma, a \bullet A \vdash_N b \bullet B}{\Gamma \vdash_N \mathcal{I}_\supset(a,b) \bullet A \supset B}$

L'interprétation intuitive de la première version de la règle est :"si, sous l'hypothèse A (représentée $[A]$), nous pouvons déduire B, alors $A \supset B$ est valide". C'est l'introduction de l'implication avec déchargement de l'hypothèse $[A]$. La seconde version de cette règle, s'interprète de la façon suivante. $\Gamma \vdash_N a \bullet A$ signifie que sous les hypothèses contenues dans Γ, a est un terme représentant une preuve de A. La règle se lit donc :"si, sous les hypothèses Γ et l'hypothèse selon laquelle a représente une preuve de A, nous pouvons construire un terme b, représentant une

preuve de B, alors, sous les hypothèses Γ, $\mathcal{I}_{\supset}(a,b)$ est un terme de preuve de $A \supset B$". On peut voir que l'hypothèse $a \bullet A$ est également déchargée. Les autres règles que nous donnons pour notre système sont les suivantes :

$$E_{\supset}\frac{\Gamma \vdash_N a \bullet A \supset B \qquad \Gamma \vdash_N b \bullet A}{\Gamma \vdash_N \mathcal{E}_{\supset}(a,b) \bullet B} \qquad I_{\&}\frac{\Gamma \vdash_N a \bullet A \qquad \Gamma \vdash_N b \bullet B}{\Gamma \vdash_N \mathcal{I}_{\&}(a,b) \bullet A\&B}$$

$$E_{\&_1}\frac{\Gamma \vdash_N t \bullet A\&B}{\Gamma \vdash_N fst(t) \bullet A} \qquad E_{\&_2}\frac{\Gamma \vdash_N t \bullet A\&B}{\Gamma \vdash_N snd(t) \bullet B}$$

La règle $E_{\supset}$ est l'élimination de l'implication, $I_{\&}$, l'introduction de la conjonction (notée & afin qu'il n'y ait pas de symbole commun avec notre meta-langage), $E_{\&_1}$ l'élimination de la conjonction avec projection sur le premier argument, et $E_{\&_2}$ la projection sur le second argument de la conjonction. Il s'agit à présent de coder ces règles en logique d'ordre un. Nous allons utiliser les notations suivantes : $Ded(x|y)$ pour $x \vdash_N y$, $\Gamma@x$ pour l'adjonction de l'élément x au multi-ensemble Γ, Γ représente une variable de multi-ensemble d'hypothèses – ou conjonction d'hypothèses. Nous pouvons considérer l'opérateur @ parmi les opérateurs AC. Nous gardons les opérateurs "$\bullet$", "&" et "$\supset$" infixés pour rester lisibles. Les constructeurs de termes de preuves sont utilisés tels quels. Toutes les variables sont quantifiées universellement. La règle $I_{\supset}$ se traduit par :
$\forall \Gamma\ a\ b\ A\ B\ Ded(\Gamma@a \bullet A \mid b \bullet B) \Rightarrow Ded(\Gamma \mid \mathcal{I}_{\supset}(a,b) \bullet A \supset B)$
Où l'implication $\Rightarrow$ est celle de notre meta-langage. Le système complet est donc représenté par la formule $DedNat$ suivante :

$$\begin{array}{c} DedNat = \forall\ \Gamma\ a\ b\ A\ B \\ Ded(\Gamma@a \bullet A \mid b \bullet B) \Rightarrow Ded(\Gamma \mid \mathcal{I}_{\supset}(a,b) \bullet A \supset B) \\ \wedge \\ (Ded(\Gamma \mid a \bullet A \supset B) \wedge Ded(\Gamma \mid b \bullet A)) \Rightarrow Ded(\Gamma \mid \mathcal{E}_{\supset}(a,b) \bullet B) \\ \wedge \\ (Ded(\Gamma \mid a \bullet A) \wedge Ded(\Gamma \mid b \bullet B)) \Rightarrow Ded(\Gamma \mid \mathcal{I}_{\&}(a,b) \bullet A\&B) \\ \wedge \\ Ded(\Gamma \mid a \bullet A\&B) \Rightarrow Ded(\Gamma \mid fst(a) \bullet A) \\ \wedge \\ (Ded(\Gamma \mid a \bullet A\&B) \Rightarrow Ded(\Gamma \mid snd(a) \bullet B)) \end{array}$$

Les types sont les suivants, en considérant le multi-ensemble d'hypothèses comme une conjonction. Les types ι' et o' correspondent, pour le langage décrit, aux types ι et o du meta-langage. $\tau(Ded) = \tau(@) = o' \times o' \to o'$; $\tau(\Gamma) = o'$; $\tau(a) = \tau(b) = \tau(A) = \tau(B) = \iota'$; $\tau(\bullet) = \iota' \times \iota' \to o'$; $\tau(\mathcal{I}_{\supset}) = \tau(\mathcal{E}_{\supset}) = \tau(\mathcal{I}_{\&}) = \tau(\supset) = \tau(\&) = \iota' \times \iota' \to \iota'$; $\tau(fst) = \tau(snd) = \iota' \to \iota'$;

Ce typage est dû au fait qu'il s'agit d'un codage en logique des prédicats. Ded est par exemple considéré comme une proposition prenant comme arguments, deux propositions. Il est alors très simple d'utiliser ce système. Soient φ et ψ deux constantes, pour avoir une preuve dans ce système de la formule $\varphi\ \&\ \psi \supset \psi\ \&\ \varphi$, nous écrivons la formule

$$\exists x\ (DedNat \Rightarrow x \bullet (\varphi\ \&\ \psi \supset \psi\ \&\ \varphi))$$

La recherche de la preuve, confiée à un prouveur, peut donner à x la valeur

$$\mathcal{I}_{\supset}(h, \mathcal{I}_{\&}(snd(h), fst(h)))$$

où h est l'hypothèse $h \bullet \varphi \ \& \ \psi$ qui a été déchargée. Ce qui correspond à la dérivation suivante en déduction naturelle :

$$\dfrac{\dfrac{\dfrac{h \bullet \varphi\&\psi \vdash_N h \bullet \varphi\&\psi}{h \bullet \varphi\&\psi \vdash_N snd(h) \bullet \psi} E^2_{\&} \quad \dfrac{h \bullet \varphi\&\psi \vdash_N h \bullet \varphi\&\psi}{h \bullet \varphi\&\psi \vdash_N fst(h) \bullet \varphi} E^1_{\&}}{h \bullet \varphi\&\psi \vdash_N \mathcal{I}_{\&}(snd(h), fst(h)) \bullet \psi\&\varphi} I_{\&}}{\vdash_N \mathcal{I}_{\supset}(h, \mathcal{I}_{\&}(snd(h), fst(h))) \bullet \varphi\&\psi \supset \psi\&\varphi} I_{\supset}$$

III.5.2 Typage simple des λ-termes

Nous allons maintenant donner un système de typage simple des λ-termes. Là encore, nos propres formules sont représentées par des λ-termes, nous distinguons les symboles utilisés dans le λ-calcul décrit et ceux utilisés par notre langage. Le système d'inférence de types est le suivant :

$$\textit{Abstraction}\ \dfrac{\Gamma x : \tau_1 \vdash_T t : \tau_2}{\Gamma \vdash_T Abst(x,t) : \tau_1 \to \tau_2} \qquad \textit{Application}\ \dfrac{\Gamma \vdash_T t : \tau_1 \to \tau_2 \qquad \Gamma \vdash_T u : \tau_1}{\Gamma \vdash_T Appl(t,u) : \tau_2}$$

$$P_1\ \dfrac{\Gamma \vdash_T t : \tau_1 \times \tau_2}{\Gamma \vdash_T \pi_1(t) : \tau_1} \qquad P_2\ \dfrac{\Gamma \vdash_T t : \tau_1 \times \tau_2}{\Gamma \vdash_T \pi_2(t) : \tau_2}$$

$$\textit{Paire}\ \dfrac{\Gamma \vdash_T t : \tau_1 \qquad \Gamma \vdash_T u : \tau_2}{\Gamma \vdash_T [t,u] : \tau_1 \times \tau_2}$$

Paire représente la formation d'un couple, P_1 et P_2 sont respectivement le première et la seconde projection, et *Application* et *Abstraction* sont les opérations habituelles du λ-calcul. Le codage dans la logique du premier ordre va donner la formule *Typage* :

$$\begin{gathered}
Typage = \ \forall\, \Gamma\ t\ u\ \tau_1\ \tau_2 \\
(Typ(\Gamma @ x : \tau_1 \mid t : \tau_2)) \Rightarrow Typ(\Gamma \mid Abst(x,t) : \tau_1 \to \tau_2) \\
\wedge \\
(Typ(\Gamma \mid t : \tau_1) \wedge Typ(\Gamma \mid u : \tau_2)) \Rightarrow Typ(\Gamma \mid [t,u] : \tau_1 \times \tau_2) \\
\wedge \\
(Typ(\Gamma \mid t : \tau_1 \to \tau_2) \wedge Typ(\Gamma \mid u : \tau_1)) \Rightarrow Typ(\Gamma \mid Appl(t,u) : \tau_2) \\
\wedge \\
Typ(\Gamma \mid t : \tau_1 \times \tau_2) \Rightarrow Typ(\Gamma \mid \pi_1(t) : \tau_1) \\
\wedge \\
Typ(\Gamma \mid t : \tau_1 \times \tau_2) \Rightarrow Typ(\Gamma \mid \pi_2(t) : \tau_2)
\end{gathered}$$

Avec les types : $\tau(Typ) = \tau(@) = o' \times o' \to o'$; $\tau(\Gamma) = o'$; $\tau(a) = \tau(b) = \tau(\tau_1) = \tau(\tau_2) = \iota'$; $\tau(:) = \iota' \times \iota' \to o'$; $\tau(Abst) = \tau(Appl) = \tau(Paire) = \tau(\to) = \tau(\times) = \iota' \times \iota' \to \iota'$; $\tau(\pi_1) = \tau(\pi_2) = \iota' \to \iota'$;

L'utilisation de ce système est voisine de ce que nous avions précédemment. Ainsi, pour savoir si le type $\varphi' \times \psi' \rightarrow \psi' \times \varphi'$ est valide, nous pouvons utiliser la formule

$$(Typage \Rightarrow \exists t\ (t : \varphi' \times \psi' \rightarrow \psi' \times \varphi'))$$

Où une réponse valide d'un prouveur peut être d'attribuer à t la valeur

$$Abst(h, Paire(\pi_2(h), \pi_1(h)))$$

Voyons maintenant de plus près l'aspect analogie entre les deux systèmes.

III.5.3 Analogie système logique/système fonctionnel

D'après ce que nous avons vu, ces deux systèmes sont similaires. Ce qui signifie que la preuve de toute requête de déduction

$$A = \exists x\ (DedNat \Rightarrow x \bullet P)$$

où P est une formule propositionnelle, peut être automatiquement transformée en preuve de

$$B = (Typage \Rightarrow \exists x\ (x \bullet T))$$

dans le système de typage, où T est le type "correspondant" à la formule P. Voyons ce que donne l'analogie de preuve pour les formules données en exemple. Prenons la formule relative à la déduction naturelle comme référence – mais nous pourrions tout aussi bien faire le contraire. Le terme représentant la formule complète en forme anti-prénexe pour $\exists x\ (DedNat \Rightarrow x \bullet (\varphi\ \&\ \psi \supset \psi\ \&\ \varphi))$ est :

$$\begin{gathered}
A = \\
\forall(\lambda\Gamma\ a\ b\ A\ B\ .(Ded(\Gamma@a \bullet A \mid b \bullet B) \Rightarrow Ded(\Gamma \mid \mathcal{I}_{\supset}(a,b) \bullet A \supset B)) \\
\wedge \\
\forall(\lambda\Gamma\ a\ b\ A\ B\ .((Ded(\Gamma \mid a \bullet A \supset B) \wedge Ded(\Gamma \mid b \bullet A)) \Rightarrow Ded(\Gamma \mid \mathcal{E}_{\supset}(a,b) \bullet B)) \\
\wedge \\
\forall(\lambda\Gamma\ a\ b\ A\ B\ .((Ded(\Gamma \mid a \bullet A) \wedge Ded(\Gamma \mid b \bullet B)) \Rightarrow Ded(\Gamma \mid \mathcal{I}_{\&}(a,b) \bullet A\&B)) \\
\wedge \\
\forall(\lambda\Gamma\ a\ A\ B\ .(Ded(\Gamma \mid a \bullet A\&B) \Rightarrow Ded(\Gamma \mid fst(a) \bullet A)) \\
\wedge \\
\forall(\lambda\Gamma\ a\ A\ B\ .(Ded(\Gamma \mid a \bullet A\&B) \Rightarrow Ded(\Gamma \mid snd(a) \bullet B)) \\
\Rightarrow \\
\exists(\lambda x.(x \bullet (\varphi\ \&\ \psi \supset \psi\ \&\ \varphi)))
\end{gathered}$$

Ce terme doit à présent être abstrait. Mais nous obtiendrions un terme illisible pour notre exemple. Nous allons donc considérer que les symboles suivants sont des variables libres dans $Abst(A)$:

$$\begin{array}{cccccc}
Ded & \bullet & \supset & \& & \mathcal{I}_{\supset} & \mathcal{E}_{\supset} \\
\mathcal{I}_{\&} & fst & snd & \varphi & \psi & @
\end{array}$$

Et nous allons appliquer le filtrage avec le terme représentant la forme anti-prénexe de la formule $(Typage \Rightarrow \exists t\ (t : \varphi' \times \psi' \rightarrow \psi' \times \varphi'))$, c'est-à-dire :

$$
\begin{array}{c}
B = \\
\forall(\lambda\Gamma\ t\ u\ \tau_1\ \tau_2\ .((Typ(\Gamma@x:\tau_1 \mid t:\tau_2)) \Rightarrow Typ(\Gamma \mid Abst(x,t):\tau_1 \to \tau_2)) \\
\wedge \\
\forall(\lambda\Gamma\ t\ u\ \tau_1\ \tau_2\ .((Typ(\Gamma \mid t:\tau_1) \wedge Typ(\Gamma \mid u:\tau_2)) \Rightarrow Typ(\Gamma \mid D[t,u]:\tau_1 \times \tau_2)) \\
\wedge \\
\forall(\lambda\Gamma\ t\ u\ \tau_1\ \tau_2\ .((Typ(\Gamma \mid t:\tau_1 \to \tau_2) \wedge Typ(\Gamma \mid u:\tau_1)) \Rightarrow Typ(\Gamma \mid Appl(t,u):\tau_2)) \\
\wedge \\
\forall(\lambda\Gamma\ t\ \tau_1\ \tau_2\ .(Typ(\Gamma \mid t:\tau_1 \times \tau_2) \Rightarrow Typ(\Gamma \mid \pi_1(t):\tau_1)) \\
\wedge \\
\forall(\lambda\Gamma\ t\ \tau_1\ \tau_2\ .(Typ(\Gamma \mid t:\tau_1 \times \tau_2) \Rightarrow Typ(\Gamma \mid \pi_2(t):\tau_2)) \\
\Rightarrow \\
\exists(\lambda t\ .(t:\varphi' \times \psi' \ \to\ \psi' \times \varphi'))
\end{array}
$$

Pour le problème de filtrage $\{Abst(A) \stackrel{?}{=} B\}$, nous obtenons, par l'algorithme de filtrage, la correspondance (unique) donnée dans le tableau ci-dessous où nous mettons en parallèle la correspondance obtenue par filtrage et la correspondance de Curry-Howard.

Analogie		Curry-Howard	
Correspondance $\mapsto$		Déduction naturelle	Typage
Ded	$\lambda z_1 z_2.Typ(z_1,z_2) =_\eta Typ$	Déduction	Typage
$\bullet$	$\lambda z_3 z_4.z_3 : z_4 =_\eta :$	Est une preuve de	Est un type de
$\supset$	$\lambda z_3 z_4.z_3 \to z_4 =_\eta \to$	Implication	Type fonctionnel
$\&$	$\lambda z_3 z_4.z_3 \times z_4 =_\eta \times$	Conjonction	Produit
$\mathcal{I}_\supset$	$\lambda z_3 z_4.Abst(z_3,z_4) =_\eta Abst$	Intro. de l'implication	Abstraction
$\mathcal{E}_\supset$	$\lambda z_3 z_4.Appl(z_3,z_4) =_\eta Appl$	Elim. de l'implication	Application
$\mathcal{I}_\&$	$\lambda z_3 z_4.Paire(z_3,z_4) =_\eta Paire$	Intro. de la conjonction	Formation d'un couple
fst	$\lambda z_3.\pi_1(z_3) =_\eta \pi_1$	Elim. de la conjonction	Première projection
snd	$\lambda z_3.\pi_2(z_3) =_\eta \pi_2$	Elim. de la conjonction	Seconde projection
φ	φ'	Symbole de proposition	Symbole de type
ψ	ψ'	Symbole de proposition	Symbole de type

Ainsi, une preuve de $A = \exists x\ (DedNat \Rightarrow x \bullet P)$ sera transformée en une preuve de $B = Typage \Rightarrow \exists t\ (x \bullet T)$ (et réciproquement). Mais nous pouvons aller plus loin. Le théorème de déduction pour la déduction naturelle peut s'énoncer

$$\Gamma, a \bullet A \vdash_N b \bullet B \Longleftrightarrow \Gamma \vdash_N \mathcal{I}_\supset(a,b) \bullet A \supset B$$

Ce qui s'exprime par la formule

$$\forall\Gamma\ a\ b\ A\ B\ (Ded(\Gamma@a \bullet A \mid b \bullet B) \Longleftrightarrow Ded(\Gamma \mid \mathcal{I}_\supset(a,b) \bullet A \supset B))$$

Soit, en forme anti-prénexe,

$$\forall\Gamma\ a\ b\ A\ B\ Ded(\Gamma@a \bullet A \mid b \bullet B) \Longleftrightarrow \forall\Gamma\ a\ b\ A\ B\ Ded(\Gamma \mid \mathcal{I}_\supset(a,b) \bullet A \supset B)$$

Une preuve de cette formule peut être transformée en preuve de la formule

$$\forall x\ u\ \tau_1\ \tau_2\ Typ(\Gamma@x : \tau_1 \mid u : \tau_2) \Longleftrightarrow \forall x\ u\ \tau_1\ \tau_2\ Typ(\Gamma \mid Abst(x, u) : \tau_1 \rightarrow \tau_2))$$

qui est la formule "correspondante" pour le typage. Notons enfin que dans l'énoncé des formules $DedNat$ et $Typage$, nous aurions pu permuter des hypothèses, le filtrage modulo AC aurait rétabli la correspondance. Nous aurions également pu ajouter des hypothèses inutiles. Mais alors, il aurait fallu élaguer la preuve de référence avant de procéder à l'analogie afin de les éliminer de façon automatique.

En résumé, à partir de ces deux systèmes d'inférence, codés en logique du premier ordre, nous pouvons transformer les preuves dans un système en preuve dans l'autre système. L'analogie est, dans ce cas, automatique. Notre but n'est bien sûr pas de transformer des preuves de typage en preuves de déduction naturelle, mais de montrer que là où il y a intuitivement similitude, une définition formelle de cette similitude peut être capable d'apporter une solution automatique. Dans le cas de Curry-Howard, la similitude va bien au delà de l'intuition, mais elle a l'avantage d'être bien connue. Ce n'est qu'ensuite, lorsque les formules ne peuvent plus être considérées comme similaires, que l'on peut se résoudre à faire intervenir des heuristiques. Le but du chapitre IV, outre d'introduire l'algorithme de calcul de différence entre deux formules, est de produire un algorithme de transformation de preuve pour une différence particulière. Dans un cadre restreint, mais ayant des applications, nous pouvons encore prétendre à un algorithme complet (qui transforme la preuve de A en une preuve de B si et seulement si B est valide). Par contre, lorsque l'on considère la preuve par analogie dans un cadre général, comme nous le ferons au chapitre V, l'automatisation n'est bien sûr plus possible en général. L'analogie consiste alors à réutiliser le mieux possible, toute l'information de la preuve de référence.

IV

Calcul de la différence et application aux preuves par expansion

Jusqu'à présent, les procédures de preuve par analogie que nous avons mises en œuvre concernaient deux formules dites similaires. Ce qui signifie qu'à un renommage près, elles étaient équivalentes du point de vue de la logique. Même s'il faut, pour s'en convaincre considérer les propriétés des quantificateurs et des connecteurs, ou la relation $\preceq$. A présent, la formule de référence et la formule objet considérées sont telles que la validité de la formule de référence n'implique plus la validité de la formule objet. Elle peuvent être alors considérées comme réellement différentes. Nous allons donner une définition formelle de la différence et un moyen de la calculer. Grossièrement, il s'agit de séparer, dans les formules, ce qui peut être similaire et ce qui ne peut pas l'être. Cette notion de différence est très importante, car c'est de l'information que l'on en tire dont dépend l'analogie, dans un cadre général. Une fois la différence définie, il faut savoir comment l'exploiter. Avec ce que nous avons vu jusqu'à présent, nous sommes capables de traiter les parties similaires, et forts de ces outils, nous allons pouvoir nous concentrer sur la différence entre les formules, puisque c'est là que se situe le problème essentiel. Nous allons tout d'abord regarder une différence extrêmement simple, puisqu'elle consiste à ne modifier qu'une constante dans la formule de référence. C'est l'occasion de regarder à la loupe le processus d'analogie. Il s'avère que si le théorème de référence est choisi arbitrairement, cette différence minime peut mener à un problème très difficile (qui revient à la preuve automatique classique). Au contraire, en imposant une condition simple sur ce théorème, nous pouvons produire une méthode complète pour résoudre ce problème. Cette complétude s'étend à une différence un peu plus large, et permet de traiter un exemple connu dans le domaine de l'analogie.
Enfin, puisque toutes les manipulations de formules et de preuves que nous avons définies jusqu'à maintenant sont spécifiques aux preuves par expansion, nous discutons brièvement ce que pourraient être les manipulations correspondantes pour des représentations du type déduction naturelle. Les problèmes rencontrés sont en fait les mêmes, mais tout ce qui est relatifs aux formules – la relation $\preceq$, les notions de similitude et la différence – reste valable, seules les manipulations de preuves sont à reconsidérer.

IV.1 Différence entre deux formules

La différence est l'information qui détermine la marche à suivre pour construire la preuve de la formule objet par un processus d'analogie, ou, à défaut, pour adapter toute l'information

recueillie dans la preuve de référence pour faciliter cette preuve. Cette notion est donc essentielle. Elle a été évoquée auparavant par Boy de la Tour et Caferra [BdlTC88]. En se plaçant dans le cadre de l'isomorphisme de Curry-Howard, ils ont proposé une méthode consistant à filtrer les types de termes, qui peut déboucher sur la génération de lemmes suffisant à la construction de la preuve de la formule objet. Le principe repose sur l'utilisation des échecs d'un algorithme de filtrage dérivé du filtrage d'ordre deux de Huet et Lang [HL78]. Nous partons de la même idée. Mais, d'une part, nous allons appliquer le filtrage sur les formules, et d'autre part, nous allons utiliser notre algorithme de filtrage qui intègre les propriétés AC des connecteurs et que nous nous sommes attachés à rendre efficace. Au chapitre V, nous verrons de façon précise comment utiliser la différence calculée pour générer des lemmes suffisants pour construire la preuve recherchée. Dans le seconde partie du présent chapitre, nous donnons une méthode complète d'analogie pour une différence dite *minime*. La complétude signifie que nous sommes capables de construire une preuve de la formule objet si celle-ci est valide, et d'affirmer qu'elle n'est pas valide sinon.

IV.1.1 Filtrage de différence

Si une formule B est similaire à A, nous pouvons dire qu'il n'existe pas de différence entre elles. Formulé en langage naturel, cela apparaît comme une lapalissade, mais nous devons nous attacher à donner des définitions formelles aux notions de similitude, différence, etc... parce que c'est justement là que réside une difficulté majeure de l'analogie. Il s'agit de formaliser des notions et des mécanismes très intuitifs. Nous avons défini la similitude entre deux formules et une façon de la calculer; nous allons voir maintenant comment déterminer la différence entre deux formules. D'après ce que nous avons vu, $B = q(b)$ est similaire à $A = p(a)$ puisque B peut être obtenue de A par simple renommage, à condition que les types coïncident. Par contre, $B = \forall x\ p(x) \Rightarrow q(b)$ n'est plus similaire à $A = \forall x\ p(x) \Rightarrow p(a)$. Nous pouvons l'affirmer parce qu'il n'existe pas de correspondance de B vers A. Ce qui nous intéresse maintenant est de savoir pourquoi, ou en quoi, ces formules sont différentes. Le filtrage est l'outil essentiel pour reconnaître la similitude entre deux formules, il en est de même pour la différence, à ceci près que nous allons adapter quelque peu l'algorithme de filtrage. Il s'agit de filtrer ce qui peut l'être, et de conserver le reste. Ce qui revient à envisager les cas d'échecs classiques sous un angle différent. Au lieu de stopper le processus, nous allons stocker les parties de termes qui ne coïncident pas. De cette façon, nous récupérerons, en sortie, d'une part une substitution qui constitue la similitude, et d'autre part, l'ensemble des parties qui ne filtrent pas. Voyons de façon plus précise comment cela peut être réalisé.

L'algorithme décrit dans le chapitre II utilise un couple $< S, \sigma >$ où S est un problème de filtrage et σ la partie déjà construite de la solution. Remplaçons cette paire par un triplet $< S, D, \sigma >$ où D contiendra la différence. Pour l'algorithme proprement dit, nous rajoutons simplement une règle (Diff) qui se charge de stocker l'information que nous devrons exploiter par la suite. Donnons d'abord les conditions d'application de la règle (Diff).

La règle (Diff) est appliquée pour une paire $s \stackrel{?}{=} t$ lorsque :

- $\mathcal{H}(s)$ et $\mathcal{H}(t)$ sont des constantes distinctes,
- ou lorsque l'algorithme de filtrage AC d'ordre un échoue.

$$\frac{< \{\lambda\overline{x}.a(\overline{s_n}) \stackrel{?}{=} \lambda\overline{x}.b(\overline{t_m})\}@S, D, \sigma >}{< S, \{\lambda\overline{x}.a(\overline{s_n}) \stackrel{?}{\neq} \lambda\overline{x}.b(\overline{t_m})\}@D, \sigma >} \quad \text{(Diff)}$$

De plus, lorsqu'une substitution est appliquée à S, elle doit l'être également à D.

La condition d'arrêt de l'algorithme est $S = \emptyset$.

Lorsqu'à l'issue du processus, $D_f = \emptyset$, cela signifie que la règle $Diff$ n'a pas été utilisée. Par conséquent, σ_f est un filtre.

Pour un problème de filtrage S, le filtrage de différence est défini par :

Définition 1 Filtrage de différence
Problème

- Un problème d'entrée $< \{Abst(A) \stackrel{?}{=} B\}, \{\}, \{\} >$

Déroulement

- L'application de l'algorithme de filtrage lorsque celui-ci ne provoque pas d'échec
- Pour une paire $s \stackrel{?}{=} t$, la règle $(Diff)$ est appliquée quand
  - $\mathcal{H}(t)$ et $\mathcal{H}(t)$ sont des constantes distinctes,
  - ou l'algorithme de filtrage AC d'ordre un est en échec.
- L'algorithme s'arrête lorsque $S = \emptyset$.

Solutions
Une solution $< \emptyset, D_f, \sigma_f >$ d'un problème de filtrage de différence $< \{Abst(A) \stackrel{?}{=} B\}, \{\}, \{\} >$ est acceptable si D_f ne contient que des termes clos.

Pour toute solution acceptable, toute paire $\{s \stackrel{?}{\neq} t\}$de D_f est transformée en $\{f(s) \stackrel{?}{\neq} f(t)\}$ tels que $f(s)$ et $f(t)$ sont les littéraux qui contiennent s et t. C'est-à-dire que pour une différence $s \stackrel{?}{\neq} t$, s apparaît dans $\sigma_f(Abst(A))$, dans un littéral $f(s)$. Ainsi, au lieu de considérer la différence $s \stackrel{?}{\neq} t$, nous considérons la différence $f(s) \stackrel{?}{\neq} f(t)$. □

Pour une solution acceptable, nous avons la propriété suivante :
Si $< \{Abst(A) \stackrel{?}{=} B\}, \{\}, \{\} > \Rightarrow^* < \emptyset, D_f, \sigma_f >$

$$\text{alors } (\sigma_f(Abst(A)))[\overline{g \mapsto d}] =_{AC} B$$

où $D_f = \{\overline{g \stackrel{?}{\neq} d}\}$. Pour faire le lien avec ce qui a été dit précédemment, σ_f reflète la partie similaire, et D_f la différence. En appliquant la correspondance, il suffit de remplacer les parties gauches de la différence par les parties droites pour obtenir la formule objet, ou une formule AC équivalente. Remarquons également que la terminaison de l'algorithme n'est pas remise en cause. En effet, là où nous avions un arrêt de l'algorithme de filtrage, nous poursuivons le processus avec une mesure de complexité plus petite, puisque nous ôtons une paire à S.

Exemple 2 Cet exemple est tiré de [BdlTK92], nous y reviendrons au paragraphe IV.2.2. La formule de référence est

$$A = \begin{array}{l} \forall xy\ (q(x,y) \Rightarrow \forall z\ (p(x,z) \Rightarrow p(y,z))) \wedge \\ q(a,b) \wedge (p(a,a) \vee p(b,b)) \Rightarrow \exists x\ p(b,x) \end{array}$$

et la formule objet,

$$B = \begin{array}{l} (p(c,c) \vee p(a,a)) \wedge q(b,c) \wedge q(a,b) \wedge \\ \forall xy\ (q(x,y) \Rightarrow \forall z\ (p(x,z) \Rightarrow p(y,z))) \Rightarrow \exists x\ p(c,x) \end{array}$$

l'abstraction de la formule A s'effectue par

$$Abst = \left\{ \begin{array}{l} p \mapsto \lambda x_1 x_2 . P(x_1, x_2) \\ q \mapsto \lambda x_1 x_2 . Q(x_1, x_2) \\ a \mapsto Z_1 \\ b \mapsto Z_2 \end{array} \right.$$

D'où la formule abstraite obtenue

$$C = Abst(A) = \begin{array}{l} \forall xy\ (Q(x,y) \Rightarrow \forall z\ (P(x,z) \Rightarrow P(y,z))) \wedge \\ Q(Z_1, Z_2) \wedge (P(Z_1, Z_1) \vee P(Z_2, Z_2)) \Rightarrow \exists x\ P(Z_2, x) \end{array}$$

On applique alors le filtrage de différence entre C et B. On obtient alors la solution

$$\begin{array}{l} < \{C \stackrel{?}{=} B\}, \{\}, \{\} > \\ \Longrightarrow^* < \{\neg q(a,c) \stackrel{?}{=} \neg q(a,b) \vee \neg q(b,c)\}, \{\}, \sigma > \\ \Longrightarrow_{(\mathrm{Diff})} < \{\}, \{\neg q(a,c) \stackrel{?}{\neq} \neg q(a,b) \vee \neg q(b,c)\}, \sigma > \end{array}$$

avec $\sigma = \left\{ \begin{array}{l} P \mapsto \lambda x_1 x_2 . p(x_1, x_2) \\ Q \mapsto \lambda x_1 x_2 . q(x_1, x_2) \\ Z_1 \mapsto a \\ Z_2 \mapsto c \end{array} \right.$ et donc, $\rho = \sigma(Abst) = \left\{ \begin{array}{l} p \mapsto \lambda x . p(x) \\ q \mapsto \lambda x . q(x) \\ a \mapsto a \\ b \mapsto c \end{array} \right.$

La différence obtenue est donc :

$$D = \{\neg q(a,c) \stackrel{?}{\neq} \neg q(a,b) \vee \neg q(b,c)\}$$

Nous reprendrons cet exemple dans la section IV.2.2, et nous verrons que la formule objet peut être prouvée automatiquement grâce à cette différence. ∎

Nous avons donc une méthode pour calculer la différence entre les formules. Celle-ci peut être utilisée de diverses façons. Dans un premier temps, nous allons voir comment, dans un cas simple, l'information délivrée par le filtrage de différence peut être utilisée. En regardant pourquoi le processus n'est pas complet en général, nous verrons qu'il suffit d'imposer une condition simple sur le théorème utilisé comme référence pour que la complétude soit atteinte. Il sera par la suite question d'utilisation plus générale de la différence.

IV.2 Processus d'analogie utilisant la différence

IV.2.1 Le cadre et la méthode d'analogie

Résumons d'abord le processus dont la différence est issue :

Calcul de la différence entre A et B

1- Abstraction de la formule de référence : $C = Abst(A)$

2- $< \{C \stackrel{?}{=} B\}, \{\}, \{\} > \Longrightarrow^{*}_{\text{filtrage de différence}} < \{\}, D_f, \sigma_f >$

$$\mathcal{D} = \{D_f, \sigma_f\} \text{ : différence entre } A \text{ et } B$$

On a alors $\sigma_f(C)[\overline{g \mapsto d}] =_{AC} B$ avec $\mathcal{D} = \{\overline{g \stackrel{?}{\neq} d}\}$. On notera $\Sigma(C) = \sigma_f(C)[\overline{g \mapsto d}]$. Il s'agit à présent d'utiliser cette information pour construire la nouvelle preuve.

Définition 3 Si D, la différence calculée est telle que :

$$\mathcal{D} = \{D, \sigma\} \text{ avec } D = \{(p(\overline{t_n}) \stackrel{?}{\neq} \vee_{i=1}^{k} p(\overline{t_n^k}))\}$$

où les $\overline{t_n}$ sont des symboles de constantes. Cette différence est dite minime. □

Nous allons produire un mécanisme qui transforme une preuve de A en une preuve de B si et seulement si la différence entre A et B est de ce type, et B est valide. Mais il nous semble important, avant cela, de montrer que ce type de différence correspond à des cas intéressants en pratique. Donnons donc un exemple rentrant dans ce cadre afin de pouvoir cerner les cas couverts par cette restriction sur la différence. Signalons par ailleurs que A. Felty et D. Howe, dans [FH94] utilisent ce type d'exemples - où une constante est substituée à une autre - pour illustrer leur méthode de généralisation, puis de réutilisation de tactiques de preuves.

Exemple 4 Soit la formule de référence :

$\forall xyz\,(x > y \wedge y > z) \wedge \Rightarrow x > z \wedge$	Transitivité de l'ordre sur les entiers
$\forall xy\,(x > y) \Rightarrow \forall z\,(x + z > y + z) \wedge$	Conservation de l'ordre par l'addition d'un entier
$a > b$	Hypothèse close
$\Rightarrow \exists x\,(a + x > b + x)$	But

Pour le typage, nous avons $\tau(x) = \tau(y) = \tau(z) = \iota$, $\tau(>) = \iota \times \iota \rightarrow o$ et $\tau(+) = \iota \times \iota \rightarrow \iota$. Ce théorème se présente donc comme une conjonction d'hypothèses, formée de propriétés de l'ordre sur les entiers et de l'addition, puis d'une hypothèse close. Considérons maintenant la formule suivante :

$\forall xyz\,(x \supseteq y \wedge y \supseteq z) \wedge \Rightarrow x \supseteq z \wedge$	Transitivité de l'inclusion
$\forall xy\,(x \supseteq y) \Rightarrow \forall z\,(x \cup z \supseteq y \cup z) \wedge$	Conservation de l'inclusion par union d'un ensemble
$b \supseteq c \wedge a \supseteq b$	Hypothèse close
$\Rightarrow \exists x\,(a \cup x \supseteq c \cup x)$	But

Le typage est le suivant : $\tau(x) = \tau(y) = \tau(z) = \iota$, $\tau(\supseteq) = \iota \times \iota \to o$ et $\tau(\cup) = \iota \times \iota \to \iota$.

La différence entre ces deux formules, si l'on abstrait A et que l'on applique les techniques décrites auparavant, est simplement :

$$\{a \supseteq c \stackrel{?}{\neq} a \supseteq b \vee b \supseteq c\}$$

Cela signifie que cet exemple rentre dans le cadre fixé d'une part, puisqu'il suffit de prendre $a = \supseteq$, $t_1 = a$, $t_2 = c$, $t_1^1 = a$, $t_1^2 = b$, $t_2^1 = b$ et $t_2^2 = c$, et que nous serons capables, possédant une preuve de A de valider automatiquement toutes les formules du type :

$$\begin{array}{l} \forall xyz\,(x \supseteq y \wedge y \supseteq z) \wedge \Rightarrow x \supseteq z \wedge \\ \forall xy\,(x \supseteq y) \Rightarrow \forall z\,(x \cup z \supseteq y \cup z) \wedge \\ C \\ \Rightarrow \exists x\,(a \cup x \supseteq c \cup x) \end{array}$$

où C est une conjonction d'hypothèses closes relatives au prédicat $\supseteq$. Dans le cas où la formule objet n'est pas valide, l'arbre d'expansion construit ne pourra être couvert par une connexion; nous serons alors capables d'affirmer qu'il n'en existe pas de preuve (cf. lemme 10). ■

Nous voyons donc que si le cadre fixé n'est qu'un cas particulier d'analogie, il permet néanmoins d'utiliser une preuve de propriété sur les entiers pour construire automatiquement une preuve sur des ensembles. De plus, tous les théorèmes utilisant les mêmes propriétés, mais avec des instances différentes seront prouvés s'ils sont valides, et invalidés sinon. On peut par exemple appliquer le même type de méthode pour des propriétés générales d'un programme dont on veut prouver les propriétés avec différentes données.
Abordons maintenant la reconstruction de la preuve. Nous allons montrer que le cas de base, qui consiste en une différence vide se traite comme une correspondance. Rappelons que nous travaillons toujours ici avec des preuves par expansion.

Lorsque la différence est du type donné plus haut, et que le théorème de référence possède une propriété simple, énoncée plus loin, la méthode adoptée est très simple. Il suffit de provoquer une sorte d'explosion des expansions, telle que si B est valide, il existe une connexion – donc une connexion minimale – qui l'affirme. Ensuite, comme la preuve obtenue a été élargie, nous appliquerons l'élagage de preuves décrit au chapitre I. Ainsi, nous obtiendrons une preuve tout à fait raisonnable de la formule B. Si B n'est pas valide, il sera impossible de trouver de connexion couvrant l'arbre. Mais comme le problème est décidable, nous aurons de toute façon une réponse. Précisons encore une fois que le théorème de référence possède une propriété spéciale, sans quoi ce qui vient d'être dit est faux en général. Cette propriété peut être vue intuitivement de la façon suivante. Si un théorème peut être vu comme la disjonction de deux théorèmes différents, celui-ci ne rentre pas dans le cadre de cette méthode. Par exemple, si le théorème de référence est $F \vee A \vee \neg A$, toute preuve de ce théorème peut être en fait, soit une preuve de F, soit une preuve de $A \vee \neg A$. Dans le deuxième cas, il est absurde de procéder à l'analogie si la différence entre cette formule et la formule à montrer porte sur F. On peut par exemple imaginer que F soit le théorème de Fermat ! La preuve de $A \vee \neg A$ risque de ne pas nous être d'un grand secours. Par contre, nous pouvons dire que si nous possédons une preuve par expansion du théorème de référence, l'élagage de cette preuve nous dira que la formule prouvée est en réalité $A \vee \neg A$ et non pas F. Donnons à présent la définition formelle de la méthode.

Définition 5 Pour une différence entre les formules A et B du type

$$\{\rho p(\overline{t_n}) \stackrel{?}{\neq} \vee_{i=1}^{k} p(\overline{t_n^k}))\}$$

où les $\overline{t_n}$ sont des symboles de constantes, et $\rho(p(\overline{t_n})) = p(\rho(\overline{t_n}))$, soient $\{\overline{s_n}\}$ l'ensemble des littéraux tels que $(p(\overline{s_n}), p(\overline{t_n})) \in \mathcal{M}$. On défini l'ensemble $\mathcal{J}$ par :

$$\mathcal{J} = \{j \mid s_j \in \{\overline{s_n}\} \text{ et } s_j \text{ est un terme d'expansion}\}$$

□

Définition 6 Explosion
Pour chaque $j \in \mathcal{J}$, appliquer toutes les expansions possibles dans l'arbre avec les termes d'expansion t_j^i pour $i = 1, \ldots, k$. □

Il est évident que l'explosion va, en général, engendrer des expansions superflues. Mais nous disposons d'un algorithme d'élagage qui va les effacer. Globalement, la reconstruction de la preuve s'exprime ainsi

Construction d'une preuve respectivement à une différence

- Soit A une formule telle que $(Q, \mathcal{M}) \vdash A$ avec $\mathcal{M}$ une connexion minimale, et soit B une formule dont la différence avec B, obtenue par abstraction de A et filtrage de différence, est $\mathcal{D} = \{D, \sigma\}$ avec $D = \{p(\overline{t_n}) \stackrel{?}{\neq} \vee_{i=1}^{k} p(\overline{t_n^k}))\}$ où les $\overline{t_n}$ sont des symboles de constantes,
- Q' est le résultat de l'explosion de Q respectivement à D et $Q'' = Q'[p(\overline{t_n}) \mapsto \vee_{i=1}^{k} p(\overline{t_n^k}))]$,
- Soit $\mathcal{M}'$ une connexion minimale couvrant Q'' si elle existe, et $R = TR(E(Q''), B, B)$.

Alors, $(R, \mathcal{M}') \vdash B$ si $\mathcal{M}'$ existe.

Avant de prouver la correction de cette méthode et d'en discuter la complétude, nous allons développer un exemple complet.

IV.2.2 Un exemple complet d'application de cette technique

Nous allons développer un exemple qui a déjà été utilisé pour la preuve par analogie, mais que notre méthode permet d'automatiser totalement. Cet exemple a été introduit par Boy de la Tour et Kreitz [BdlTK92], et a été repris comme exemple par Melis dans [Mel93b]. Il s'agit des formules introduites dans l'exemple 2. La formule de référence est

$$\begin{aligned} A = \quad & \forall xy\ (q(x,y) \Rightarrow \forall z\ (p(x,z) \Rightarrow p(y,z))) \wedge \\ & q(a,b) \wedge (p(a,a) \vee p(b,b)) \Rightarrow \exists x\ p(b,x) \end{aligned}$$

et le nouveau théorème à prouver,

$$\begin{aligned} B = \quad & (p(c,c) \vee p(a,a)) \wedge q(b,c) \wedge q(a,b) \wedge \\ & \forall xy\ (q(x,y) \Rightarrow \forall z\ (p(x,z) \Rightarrow p(y,z))) \Rightarrow \exists x\ p(c,x) \end{aligned}$$

On considère, la preuve par expansion $(Q, \mathcal{M})$ pour A, symbolisée par la figure IV.1.

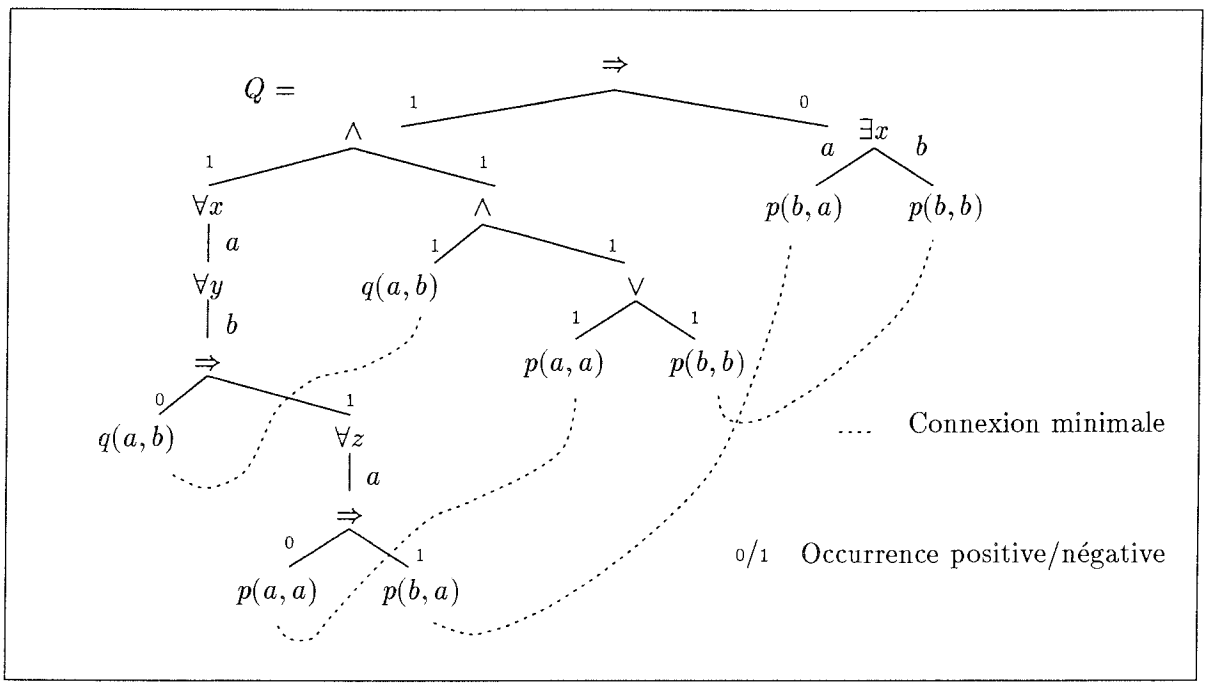


Figure IV.1: Preuve de la formule de référence

Rappelons que par filtrage de différence, on obtient :

$$
\begin{aligned}
&< \{C \stackrel{\triangleleft}{=} B\}, \{\}, \{\} > \\
\Longrightarrow^{*} &< \{\neg q(a,c) \stackrel{\triangleleft}{=} \neg q(a,b) \vee \neg q(b,c)\}, \{\}, \sigma > \\
\Longrightarrow_{(\mathrm{Diff})} &< \{\}, \{\neg q(a,c) \stackrel{\triangleleft}{\neq} \neg q(a,b) \vee \neg q(b,c)\}, \sigma >
\end{aligned}
$$

où C est le théorème abstrait.

$$
\text{avec } \sigma = \begin{cases} P \mapsto \lambda x_1 x_2 . p(x_1, x_2) \\ Q \mapsto \lambda x_1 x_2 . q(x_1, x_2) \\ Z_1 \mapsto a \\ Z_2 \mapsto c \end{cases} \quad \text{et donc, } \rho = \sigma(Abst) = \begin{cases} p \mapsto \lambda x . p(x) \\ q \mapsto \lambda x . q(x) \\ a \mapsto a \\ b \mapsto c \end{cases}
$$

la différence et σ ont les propriétés requises pour appliquer l'explosion. Dans la connexion de la preuve de référence, le littéral $l = \neg q(a,b)$ tel que $\rho(l) = \neg q(a,c)$ est connecté au littéral $q(a,b)$ où a et b sont des termes d'expansion. Par conséquent, l'explosion va être effectuée avec les deux arguments des deux littéraux apparaissant dans la partie droite de la différence, c'est à dire a, b et c. Enfin, la partie droite de la différence est substituée à la partie gauche. On obtient l'arbre d'expansion de la figure IV.2.

Après explosion, une connexion minimale est trouvée, et nous élaguons l'arbre respectivement à cette connexion. On obtient alors la preuve par expansion de B de la figure IV.3.

Elle correspond à la preuve que nous aurions construite à la main, sans se référer à celle de A.

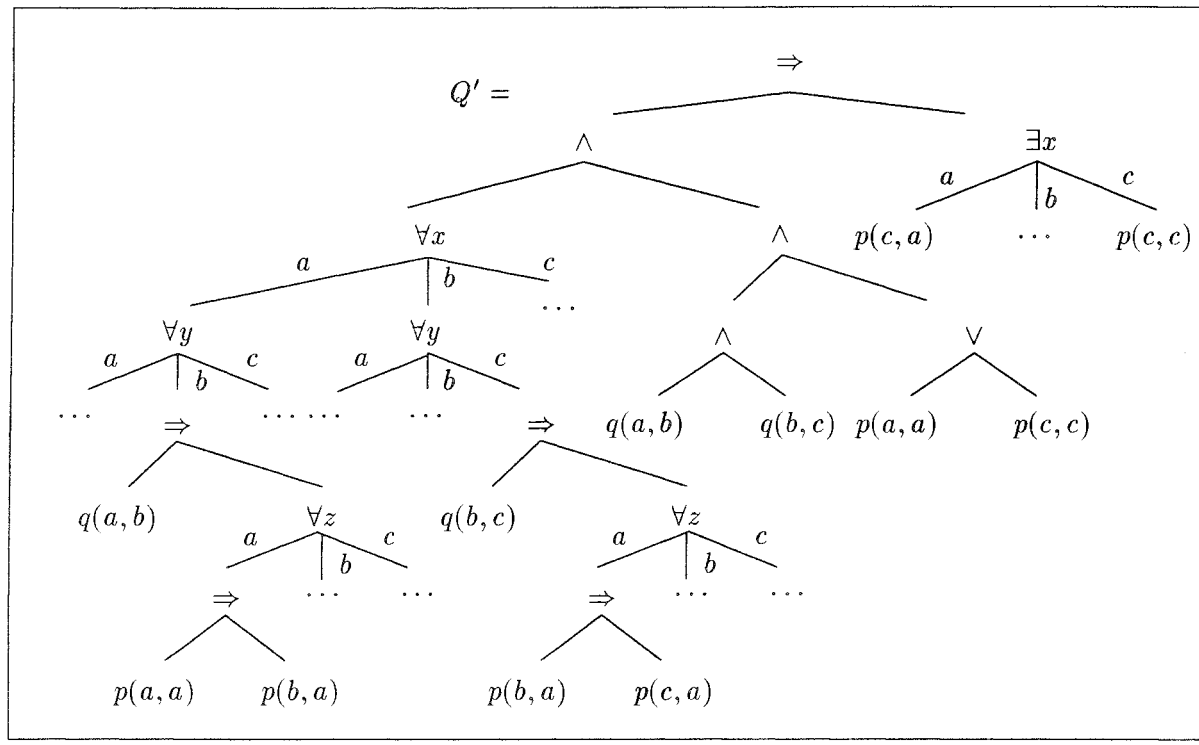


Figure IV.2: Arbre d'expansion de B obtenu par explosion

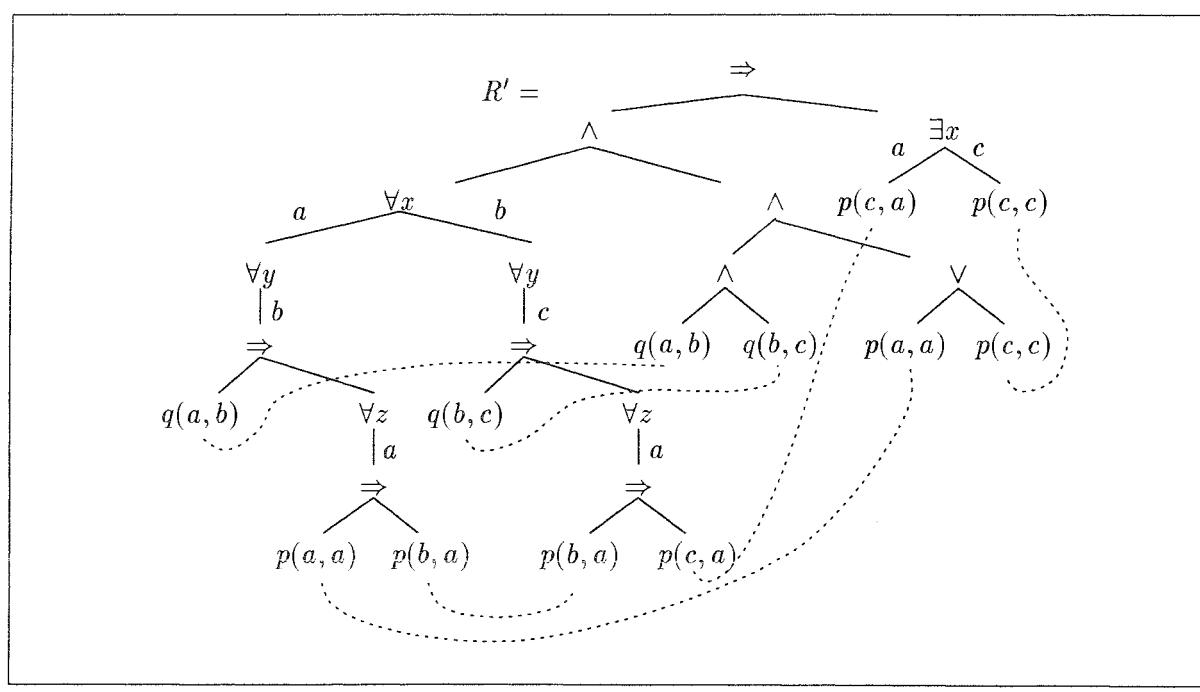


Figure IV.3: Preuve de B construite par analogie à celle de A

IV.3 Discussion de la complétude pour une différence minime

La méthode de l'explosion n'est malheureusement pas complète pour une différence minime. Il existe en effet des cas où nous ne pourrons pas construire une preuve de la formule objet alors que celle-ci est valide (cf. exemple 7). Mais nous pouvons comprendre, sur un cas de base, quelles en sont les raisons. En regardant de près comment construire une preuve de complétude pour un cas de base, nous allons voir qu'il suffit que le théorème pris comme référence possède une propriété simple pour que la complétude soit atteinte. En effet, nous allons montrer que cette méthode est complète lorsque le théorème de référence ne peut pas s'écrire comme la disjonction de deux formules valides différentes.

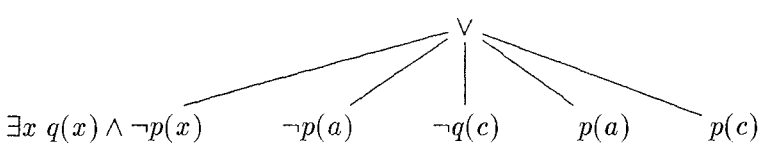
Nous avons vu au chapitre I, qu'un théorème peut être épuré en prenant sa preuve et en élagant les parties qui ne sont pas utilisées par la connexion minimale. Mais ceci n'est pas encore suffisant. Plaçons-nous dans le cas qui doit être le plus simple. Nous avons une formule A, et une preuve par expansion $(Q, \mathcal{M}) \vdash A$ telle que $\mathcal{M}$ est minimale. Soit B, une formule telle que la différence entre A et B porte simplement sur une constante d'ordre un. C'est-à-dire que B est A où l'on a remplacé une constante a par une constante b; ou encore, cela correspond au cas $k = n = 1$ de la restriction sur la différence énoncée plus haut. Pour ce cas de figure, le lemme 9 affirme que si la méthode d'explosion n'est pas complète, il existe une formule $A_1 \vee A_2$ équivalente à A et telle que A_1 et A_2 sont toutes deux valides. En d'autres termes, A est la disjonction de deux théorèmes.

Nous allons voir, sur un exemple simple, en quoi il est normal que cette méthode ne soit pas complète dans le cas général. Le simple fait de remplacer une constante par une autre pouvant transformer un processus d'analogie en une recherche automatique de preuve tout à fait générale.

Exemple 7 Prenons la formule

$$A = (\forall x\ (q(x) \Rightarrow p(x)) \wedge p(a) \wedge q(c)) \Rightarrow p(a) \vee p(c)$$

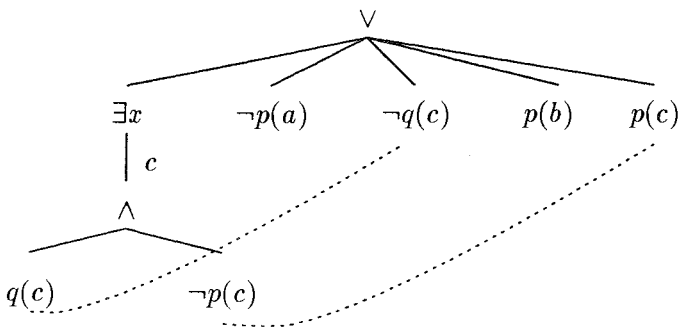
L'arbre d'expansion suivant, muni de la connexion $\mathcal{M} = \{(\neg p(a), p(a))\}$ est une preuve par expansion de A.



Substituons à présent a par b dans l'occurrence positive de $p(a)$. Nous obtenons la formule

$$B = (\forall x\ q(x) \Rightarrow p(x) \wedge p(a) \wedge q(c)) \Rightarrow p(b) \vee p(c)$$

qui est une formule valide. Mais l'explosion n'en fournira pas une preuve, puisque c'est par c qu'il faut expanser x, et que l'explosion ne fera qu'expanser x par b. Observons une preuve par expansion de B.



Cet exemple est suffisamment simple, pour se rendre compte qu'en fait, les deux preuves données ne prouvent pas le même théorème. En effet, la première preuve est relative à la formule $p(a) \Rightarrow p(a)$, alors que la seconde est relative à la formule $(\forall x\ (q(x) \Rightarrow p(x)) \wedge q(c)) \Rightarrow p(c)$. Nous pouvons alors remarquer que la formule A peut s'exprimer par $A = A_1 \vee A_2$ avec $A_1 = p(a) \Rightarrow p(a)$, et $A_2 = (\forall x\ (q(x) \Rightarrow p(x)) \wedge q(c)) \Rightarrow p(c)$. Notons au passage que c'est la connexion qui nous permet de différencier les deux éléments de cette disjonction. ■

Lemme 8 *Soit a un symbole de constante d'ordre un apparaissant dans une formule A avec $(Q, \mathcal{M}) \vdash A$ et telle que $f(a)$ est la plus grande sous-formule de A et où ni $f(a)$ ni aucun de ses sous-termes n'apparaît dans $\mathcal{M}$. Alors $A \Leftrightarrow A_1 \vee f(a)$ avec $\models A_1$ et $f(a) \notin A_1$.*

Preuve Toutes les clauses complètes de Q sont couvertes par $\mathcal{M}$. Pour toutes les clauses complètes c de Q contenant $f(a)$, nous avons donc $c = c' \vee f(a)$ où $f(a) \notin c'$ (nous ne pouvons pas avoir $c = c' \wedge f(a)$ puisque $f(a)$ est le plus grand sous-terme non connecté, sinon, c ne pourrait être couverte par $\mathcal{M}$). Ce qui implique, pour la conjonction des clauses complètes qu'il existe C' telle que $\wedge\mathrm{cc}(Q) = C \vee f(a)$ où $f(a) \notin C$. On en déduit donc qu'il existe A_1 telle que $A = A_1 \vee f(a)$ avec $\models A_1$ et $f(a) \notin A_1$. □

Lemme 9 *Soient A et B deux formules telles que $A[p(a) \mapsto p(b)] = B$, avec $(Q, \mathcal{M}) \vdash A$, et a et b des symboles de constantes. Soit Q' un arbre d'expansion obtenu de Q par explosion respectivement à la différence $\{p(a) \stackrel{?}{\neq} p(b)\}$.*

S'il existe $\mathcal{M}'$ couvrant Q', alors $\models B$ et $(Q', \mathcal{M}') \vdash B$,
sinon, ou bien $\not\models B$, ou bien $A = A_1 \vee A_2$ avec $\models A_1$ et $\models A_2$.

Preuve Si $p(a) \notin \mathcal{M}$, alors tout remplacement de a par b dans A donne une preuve de B, puisque, par le lemme 8, il existe A_1 telle que $A = A_1 \vee p(a)$ avec $\models A_1$ et $p(a) \notin A_1$.
Si $p(a) \notin \mathcal{M}$, et s'il existe $\mathcal{M}'$ couvrant Q', puisque nous ne manipulons que des constantes, $<_{Q'}$ reste acyclique, et par correction des preuves par expansion, nous avons $(Q', \mathcal{M}') \vdash B$ implique $\models B$.
Le problème se pose lorsqu'il n'existe pas de connexion $\mathcal{M}'$ couvrant Q'. Si B n'est pas valide, $\not\models B$ implique $\nexists \mathcal{M}'$ telle que $(Q', \mathcal{M}') \vdash B$. Par contre, si nous avons $\models B$ et qu'il n'existe pas de couverture de Q', cela signifie qu'il existe une façon d'expanser Q' en Q'' tel qu'il existe $\mathcal{M}''$ qui couvre Q'', par complétude des preuves par expansion. Nous sommes dans l'hypothèse où $p(a) \in \mathcal{M}$. Il existe donc au moins une clause complète c de Q telle que

$$c = \cdots \vee p(a) \vee \cdots \vee \neg p(a) \vee \cdots$$

où $\{p(a), \neg p(a)\}$ est le seul couple de $\mathcal{M}$ couvrant c. Voyons ce qu'est devenue cette clause complète dans Q''. Le symbole a contenu dans $\neg p(a)$ ne peut être un terme d'expansion. Dans le cas contraire nous aurions

$$c' = \cdots \vee p(a) \vee \cdots \vee \neg p(a) \vee \neg p(b) \vee \cdots$$

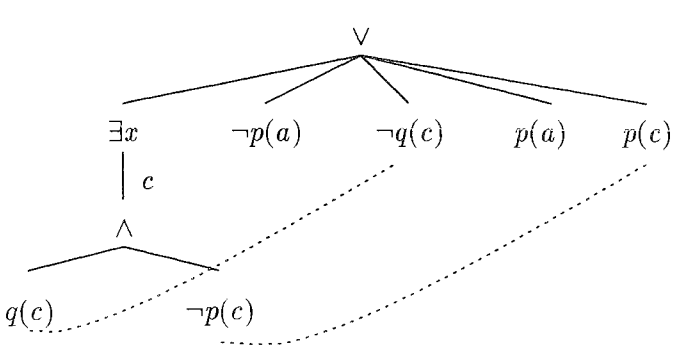
en conséquence de l'explosion. Il existerait donc $\mathcal{M}'$ telle que $(Q'\mathcal{M}') \vdash B$. Les deux occurrences de a sont donc des constantes dans A. Nous avons donc deux cas pour la nouvelle clause c'' de Q''.

- $c'' = \cdots \vee p(a) \vee \cdots \vee p(b) \vee \cdots \vee \neg p(a) \vee \cdots$. Dans ce cas, nous pouvons dire qu'il existe une preuve de A qui n'utilise pas (dans sa connexion) l'occurrence de $p(a)$ que nous avions dans c et qui a été remplacée par $p(b)$. Par conséquent, il existe A_1 vérifiant $A = A_1 \vee p(a)$ et $\models A_1$ où $p(a) \notin A_1$, et d'autre part, il existe A_2 où $p(a)$ et $\neg p(a)$ apparaissent, telle que $\models A_2$ (il suffit de prendre $A_2 = A$), et $A = A_1 \vee A_2$.
- $c'' = \cdots \vee \neg l \vee l \vee \cdots \vee p(b) \vee \cdots \vee \neg p(a) \vee \cdots$. Alors, il existe A_1 vérifiant $A = A_1 \vee p(a) \vee \neg p(a)$ avec $\models A_1, p(a) \vee \neg p(a) \notin A_1$ et A_2 où apparaissent $p(a)$ et $\neg p(a)$ (il suffit encore de prendre $A_2 = A$), et $A_1 \vee A_2 = A$.

□

Revenons sur l'exemple 7. La leçon essentielle que nous devons tirer de cet exemple est qu'un théorème peut en cacher un autre. Nous pensions prouver A, alors que nous n'en avons prouvé qu'une partie suffisante. En remplaçant a par b, nous avons transformé la formule en un théorème très différent, pour lequel la preuve de A, telle qu'elle était donnée ne peut pas être d'une grande utilité. $p(a) \Rightarrow p(b)$ n'étant pas valide, la preuve de B doit être une preuve de A_2. Dire qu'il peut y avoir analogie entre la preuve de A_1 et la preuve de B n'a aucun sens. Le problème que nous pensions être un problème d'analogie devient alors un problème de preuve automatique classique, sans analogie. Dès lors, il n'est plus pensable d'avoir un processus complet pour résoudre ce genre de problèmes.

Cet exemple montre encore l'importance de la formule choisie comme référence. Ce que nous avons décrit au chapitre I, est toujours valable ici. Les outils qui pouvaient paraître naïfs au premier abord trouvent ici tout leur sens. En effet, si nous considérons la preuve de A, et que nous lui appliquons l'élagage relativement à sa connexion, nous obtenons le théorème de référence $p(a) \Rightarrow p(a)$, c'est-à-dire A_1. Il ne fait aucun doute que l'analogie entre la preuve de cette formule et celle de B ne nous viendrait pas à l'esprit. Par contre, si la preuve de A avait été



l'élagage aurait donné la formule A_2. Or, $A_2 \preceq B$. L'analogie aurait pu être résolue. Il est donc essentiel de procéder à l'élagage en premier lieu, avant de tenter de prouver par analogie. Malheureusement, ceci ne peut résoudre tous les problèmes, car il existe sans doute des preuves par expansion où une connexion minimale est telle que l'élagage ne permette pas de dissocier les deux théorèmes cachés en un seul. C'est pourquoi la condition imposée n'est pas d'avoir une preuve élaguée, mais un théorème qui n'est pas une disjonction de deux théorèmes.

Nous avons étudié le cas simple d'une constante substituée afin de bien comprendre les rouages de l'analogie, et de voir qu'un problème qui peut sembler évident à première vue, se ramène en réalité à un problème de déduction automatique classique. Voyons maintenant que la complétude que nous avons obtenue pour ce cas particulier de théorème de référence peut s'étendre à la différence donnée plus haut comme cas particulier.

Lemme 10 *Soient A et B deux formules telles que $A[p(\overline{t_n}) \mapsto \vee_{i=1}^{k} p(\overline{t_n^k})] = B$, avec $(Q, \mathcal{M}) \vdash A$, et $\overline{t_n}$ et $\overline{t_n^k}$ des symboles de constantes. Soit Q' un arbre d'expansion obtenu de Q par explosion respectivement à la différence $D = \{p(\overline{t_n}) \stackrel{?}{\neq} \vee_{i=1}^{k} p(\overline{t_n^k}))\}$ S'il existe $\mathcal{M}'$ couvrant Q', alors $\models B$ et $(Q', \mathcal{M}') \vdash B$.*
Sinon, soit $\not\models B$, soit $A = A_1 \vee A_2$ avec $\models A_1$ et $\models A_2$.

Preuve Le cas $k = n = 1$ est couvert par le lemme 9. Nous allons donc faire un induction sur k et sur n.
$n = 1$: le lemme est vérifié pour $D = \{p(a) \stackrel{?}{\neq} p(a_1) \vee \cdots \vee p(a_k)\}$. Dans ce cas, soit il existe un $p(a_i)$ connecté dans les clauses complètes qui étaient couvertes par $\{p(a), \neg p(a)\}$, soit le problème revient à la différence $D' = \{p(a) \stackrel{?}{\neq} p(a_k)\}$ (lemme 9).
$k = 1$: le lemme est vérifié pour $D = \{p(a_1, \ldots, a_n) \stackrel{?}{\neq} p(b_1, \ldots, a_n)\}$. Si $D = \{p(a_1, \ldots, a_n, a_{n+1}) \stackrel{?}{\neq} p(b_1, \ldots, b_n, b_{n+1})\}$. Il suffit alors de prendre $p(a_1, \ldots, a_{n+1}) = p_1(a_1, \ldots, a_n) \wedge p_2(a_{n+1})$, et le problème se ramène à une différence $D' = \{p_2(a_{n+1}) \stackrel{?}{\neq} p_2(b_{n+1})\}$.
$k > 1$ et $n > 1$: par combinaison des deux cas précédents. Si le lemme est vérifié pour tout k, le raisonnement de $k = 1$ s'applique, et réciproquement, si le lemme est vérifié pour tout n, le raisonnement pour $n = 1$ s'applique. □

Nous obtenons donc, pour ce type de différences, et pour un théorème ne pouvant s'écrire sous forme d'une disjonction de formules valides, un résultat de complétude. Mais l'enseignement essentiel est que pour faire de la preuve par analogie, il est très important de se pencher autant sur les formules que sur les preuves. Tout du moins si on envisage d'automatiser au mieux le processus. Il est maintenant clair que nous avons beaucoup plus de chance de réussite si le point de départ est une preuve qui utilise complètement le théorème qu'elle prouve. D'autant qu'il est relativement simple (cf. chapitre I) d'obtenir une preuve d'une formule plus grande par la relation $\preceq$. Nous sommes donc confortés dans l'idée d'analyser les formules avant toute autre chose. Enfin, remarquons que dans l'exemple de la section IV.2.2, le théorème de référence ne peut pas s'écrire comme la disjonction de deux théorèmes. L'analogie aurait donc été un succès quelle que soit la preuve par expansion de cette formule.

IV.4 Extension à d'autres systèmes de preuve

Nous nous proposons dans cette section, de discuter l'adaptation des méthodes que nous avons données et qui sont spécifiques aux preuves par expansion, à d'autres représentations de preuves.

IV.4.1 Analogie pour des preuves non analytiques

La dénomination de preuves analytiques pour les preuves par expansion et non-analytiques pour les preuves du type déduction naturelle est due à Pfenning [Pfe84].
Nous n'allons pas passer en revue les différents types de représentation de preuves. Nous allons nous contenter de voir ce qui peut se passer lorsque l'on essaie d'appliquer des manipulations identiques à celles que nous avons décrites dans un formalisme du type déduction naturelle. Bien sûr, tout ce qui concerne les formules est encore valide, et cela constitue un des gros avantages de s'intéresser aux formules avant les preuves. Quel que soit le système de représentation des preuves, le relation $\preceq$ peut être exploitée sur les formules prouvées, et la différence peut être utilisée. Nous le verrons d'ailleurs dans le chapitre V. Par contre, la manipulation des preuves ne s'applique plus. Au lieu de manipuler des arbres, il nous faudrait modifier des inférences. Or, lorsque nous pratiquons l'élagage des preuves par expansion, nous nous appuyons sur l'information délivrée par les connexions. Il n'est pas évident, a priori, de savoir quelles sont les branches ou les hypothèses d'une preuve de déduction naturelle qui sont superflues. Considérons à nouveau l'exemple de la section précédente. La première preuve de A, en déduction naturelle s'écrit :

$$\dfrac{\dfrac{\dfrac{[\forall x\ (q(x) \Rightarrow p(x)) \wedge q(c) \wedge p(a)]}{p(a)}\ E_\wedge}{p(a) \vee p(c)}\ I_\vee}{\forall x\ (q(x) \Rightarrow p(x)) \wedge q(c) \wedge p(a) \Rightarrow p(a)}\ I_\Rightarrow$$

Or, cette preuve sera difficilement réutilisable pour prouver $B = \forall x\ (q(x) \Rightarrow p(x)) \wedge q(c) \wedge p(a) \Rightarrow p(b) \vee p(c)$. En fait, le problème qui se pose est le même que pour les preuves par expansion. Mais nous ne disposons pas, a priori, de techniques simples pour déterminer quelles sont les parties de A qui sont utiles dans la preuve, et quelles sont les parties superflues. Par contre, aller de la preuve de la formule élaguée, vers une preuve d'une formule plus grande par la relation $\preceq$ semble plus aisé. Il suffirait d'ajouter des étapes structurelles ($E_\wedge$, $E_\vee$, $I_\wedge$, $I_\vee$) aux endroits appropriés et de modifier les hypothèses pour reconstituer la formule objet. Ainsi, pour la preuve suivante de $A_2 = \forall x\ (q(x) \Rightarrow p(x)) \wedge q(c) \Rightarrow p(c)$,

$$\dfrac{\dfrac{\dfrac{[\forall x\ q(x) \Rightarrow p(x)]}{q(c) \Rightarrow q(c)}\ E_\forall \qquad [q(c)]}{p(c)}\ E_\Rightarrow}{\forall x\ (q(x) \Rightarrow p(x)) \wedge q(c) \Rightarrow p(c)}\ I_\Rightarrow$$

en rajoutant $p(a)$ aux hypothèses et deux règles structurelles, nous obtenons :

$$\dfrac{\dfrac{\dfrac{\dfrac{[\forall x\ q(x) \Rightarrow p(x) \wedge p(a)]}{\forall x\ q(x) \Rightarrow p(x)}\ E_\wedge}{q(c) \Rightarrow p(c)}\ E_\forall \qquad [q(c)]}{p(c)}\ E_\Rightarrow}{p(c) \vee p(a)}\ I_\vee}{\forall x\ (q(x) \Rightarrow p(x)) \wedge q(c) \wedge p(a) \Rightarrow p(c) \vee p(a)}\ I_\Rightarrow$$

qui est une preuve de la formule B. Tout comme $p(a)$ était la seule hypothèse réellement utilisée pour la preuve de A, seules, les hypothèses $\forall x\ q(x) \Rightarrow p(x) \wedge p(a)$ et $q(c)$ sont utili-

sées ici. Nous voyons que des problèmes très semblables se posent avec ce formalisme. Il semble donc raisonnable de penser que des techniques similaires à celles que nous utilisons peuvent être définies. Le principal problème serait de trouver l'équivalent des connexions. Cette définition doit sans doute passer par la formalisation de ce qu'est une hypothèse utile ou pertinente dans une preuve de déduction naturelle. Pour les techniques ne faisant pas appel aux connexions, comme l'utilisation des propriétés des quantificateurs, les correspondances ou les propriétés AC des connecteurs, elles doivent s'appliquer à la déduction naturelle. Tout comme nous pouvons inverser deux nœuds de sélection dans un arbre d'expansion, il est possible de permuter deux inférences $E_\forall$ dans une déduction; l'application d'une correspondance doit avoir le même effet sur une déduction naturelle que sur un arbre d'expansion, etc...

En conclusion, nous avons utilisé les preuves par expansion, parce qu'elles possèdent des avantages certains dans leur présentation. En particulier l'information détenue par les connexions. Mais il semble possible d'adapter les transformations de preuves par expansion en transformations d'autres types de preuves. Par contre, la constante, quelle que soit la représentation des preuves est le travail sur les formules. Pour prouver B à partir d'une preuve de A, quel qu'en soit le moyen implique que l'on connaisse la relation qui les lie. Cette relation pouvant être d'ordre propositionnel, ou bien une correspondance, etc.. ou, dans le cas général, une différence. C'est cette différence qui va maintenant nous guider dans la recherche de la nouvelle preuve dans un cadre général et indépendant de la représentation choisie.

V
Analogie dans un cadre général

Ce chapitre contient des concepts simples pour l'analogie dans un cas général pour réutiliser au maximum toute la preuve de référence. Ces techniques mènent à des processus automatiques dans certains cas, et à la génération de lemmes dans d'autres. Nous allons donner une transformation générale de la preuve de référence, basée sur la correspondance et la différence. La preuve obtenue n'est pas correcte, en général. En vérifiant étape par étape la preuve obtenue, nous nous trouvons dans l'un des cas de figure évoqués : analogie immédiate, analogie par saut, analogies horizontales ou analogie incomplète. Pour chacun de ces cas, nous donnons une méthode pour tenter de le résoudre. Cette méthode mène automatiquement à une preuve de la formule objet pour les analogies immédiates et par saut, ou génère des lemmes pour les analogies horizontales et incomplètes (mais il ne faut pas oublier que la formule objet n'est pas nécessairement valide). Pour chacune de ces méthodes, nous donnons un ou plusieurs exemples. Ceux-ci sont, le plus souvent, empruntés à des travaux sur l'analogie. Ces exemples font appel à des formalismes variés. En effet, l'un des buts de ce chapitre est de montrer que les concepts proposés sont généraux et consistent en fait à réutiliser au mieux la preuve de référence, indépendamment du contexte. Ainsi, nous nous penchons sur la synthèse de programmes logiques, l'induction, la réécriture, l'analyse sur les réels. Encore une fois, il ne s'agit pas de faire de la preuve de théorèmes, mais de produire des méthodes de réutilisation optimales de la preuve de référence, afin justement, de réduire au maximum la preuve de théorème, puisqu'il s'agit d'un problème très difficile. Les exemples indiquent que ces méthodes générales et simples produisent au moins les mêmes résultats que des systèmes spécialisés. De plus, il est possible d'y adjoindre les notions de similitudes, d'enchâssement etc.. que nous avons décrites. Nous terminons ce chapitre par une synthèse donnant la marche à suivre pour appliquer au mieux toutes les techniques développées, et en comparant notre démarche et celles des travaux voisins.

V.1 Situation dans les travaux sur l'analogie

Nous allons, pour commencer, situer les travaux les plus importants dans le domaine de l'analogie. Ce qui nous permettra de positionner ce que nous avons présenté jusqu'à maintenant, et d'introduire ce qui va suivre. Les travaux précurseurs dans ce domaine, sont ceux de Kling [Kli71], et de Plaisted [Pla81], bien que plus de dix années les séparent. Ce n'est qu'à la fin des années 80 que quelques nouveaux travaux ont été publiés. De façon générale, ces travaux s'inscrivent entre deux extrêmes dont l'un est l'automatisation, et l'autre la généralité. Les travaux de Plaisted [Pla81] s'inscrivent du coté de l'"automatisation". Il y décrit des méthodes –

généralisation et abstraction – permettant de résoudre des problèmes par analogie dans le cadre de la résolution. Ces méthodes sont, pour la plupart, complètes. Ainsi, Plaisted utilise une notion d'abstraction qui englobe la transformation d'une formule en une instance close, en une formule propositionnelle, en une formule dont on a permuté ou effacé certains arguments, ou dont on a renommé les symboles de fonctions et de prédicats. C'est à ces travaux que se rattache le mieux ce que nous avons décrit jusqu'à présent. Si l'on considère, par exemple, le renommage, il s'agit d'un sous-ensemble de ce que nous pouvons faire par correspondance. Plaisted fournit alors une méthode pour adapter la résolution appliquée à la formule de référence, pour qu'elle devienne une résolution de la formule objet.

L'autre extrémité est moins nette, mais plus dense. Il y a en effet un certain nombre de travaux qui tentent, de façon très générale, de résoudre le problème du raisonnement par analogie. On peut trouver, par exemple dans [Mel93b, Mel93a, Owe90] des heuristiques et des exemples d'applications à des théorèmes mathématiques complexes. On peut trouver par exemple dans [Mel93a], une étude de l'utilisation de l'analogie dans les preuves présentées dans "Halbgruppen und Automaten" qui semble être un ouvrage fréquemment utilisé dans les études d'informatique théorique dans les universités allemandes.

Bien sûr, ces deux extrémités aspirent à se rejoindre, puisqu'on essaie de généraliser ce qui est automatique, et d'automatiser ce qui est général. Si nous avons choisi la première approche, c'est simplement parce qu'elle nous paraît la mieux adaptée à la machine. Il nous a semblé plus approprié de partir de choses simples et sûres, constituant une base d'outils manipulables par la machine, pour aller vers des problèmes plus complexes.

Il existe également des travaux qui se situent entre ces deux points. Principalement, nous avons déjà eu l'occasion de citer les travaux de Boy de la Tour, Caferra et Kreitz [BdlTC87, BdlTC88, BdlTK92] dont le cadre est l'isomorphisme de Curry-Howard. Ainsi, travailler sur les preuves revient à manipuler des types. Ils obtiennent un bon compromis entre automatisation et généralité. On peut remarquer que ce que nous avons présenté possède un certain nombre de points communs avec ces travaux. Par exemple l'utilisation du filtrage d'ordre deux, mais il y a plus particulièrement dans [BdlTC88], l'émergence de l'idée essentielle du filtrage de différence, et de l'exploitation de celui-ci pour la génération de lemmes. Nous y reviendrons dans la suite.

D'autre part, des travaux sur l'analogie ont été élaborés pour des cadres spécifiques. Par exemple, on trouve dans [BCP88] une application intéressante du concept d'analogie à un système de déduction naturelle pour des théorèmes sur les réels. Il s'agit d'une façon de spécialiser l'analogie à une application. Enfin, dans le domaine très actif de la preuve par induction, les travaux de Bundy et al. (par exemple [Bun91]) peuvent, en un certain sens, rentrer dans le cadre de l'analogie. L'outil de base : le "rippling" consiste en effet à retrouver dans les processus d'induction des schémas connus, et à leur appliquer un traitement qui, en général, guide vers la solution. Citons également pour l'induction, des travaux récents [KW94] qui tentent de réunir des techniques comme l'abstraction à la Plaisted et des concepts beaucoup plus généraux d'analogie.

Si nous essayons de voir ce qui se recoupe dans tous ces travaux – particulièrement dans ceux qui ne s'éloignent pas trop de l'automatisation –, nous pouvons remarquer un certain nombre de constantes.

- Tout d'abord, les techniques d'analogies sont suggérées par des exemples. Ceci est à peu près général.
- Il s'agit de suggérer des étapes de preuves pour la formule objet en fonction de la preuve de référence.

- Dans beaucoup de cas, [BdlTC87, BdlTC88, BdlTK92, Mel93b, KW94], le filtrage est un outil prépondérant lors du processus.
- L'analogie est basée sur des heuristiques.

Lorsque les heuristiques échouent, une échappatoire consiste à tenter de générer des lemmes aidant à la résolution du problème [BdlTC88, KW94]. Dans la suite, nous allons utiliser la différence comme outil de base. Notre approche diffère légèrement des travaux évoqués. Notre but est de réutiliser au mieux le contenu de la preuve de référence pour construire une preuve de la formule objet. Lorsque cela n'est pas suffisant, nous essaierons de donner des conditions suffisantes, sous forme de lemmes, pour montrer celle-ci. La différence avec ce qui avec ce qui précède est qu'au lieu de se poser la question : "comment montrer B?", nous allons nous poser le problème :"jusqu'à quel point peut-on réutiliser la preuve de A pour celle de B?", sans rien tenter – pour l'instant – qui n'appartienne vraiment à l'analogie, comme l'appel d'un démonstrateur classique pour combler les trous restant dans la preuve. En somme, nous ne nous intéressons qu'à la partie réellement "analogie".

V.2 Principe de l'analogie générale

Par "analogie générale", il ne faut pas voir le concept général et universel de l'analogie, mais l'extension de ce que nous avons fait jusqu'ici dans un cadre plus général général de preuve. Nous présentons donc ici une façon générale d'appliquer l'analogie, en utilisant la différence. Selon le résultat de cette opération, nous nous placerons dans l'un des cas de figures décrits dans les sections suivantes. Ces analogies seront détaillées sur des exemples. Ces exemples sont tirés de domaines assez différents, afin de montrer, d'une part, que les preuves par expansion n'étaient qu'une façon pratique d'aborder le sujet, et, d'autre part, de montrer que l'analogie peut s'appliquer à beaucoup de systèmes basés sur la preuve, quel qu'en soit le but. Voyons d'abord comment utiliser la différence.

L'idée de base est extrêmement simple. Nous allons appliquer le même type d'opération que sur les preuves par expansion, mais cette fois-ci dans un contexte aussi général que possible. C'est-à-dire, en particulier, que nous ne devons plus nous préoccuper de la représentation des preuves. Nous voyons la preuve comme une succession d'étapes menant au but visé.

Définition 1 Pour obtenir la correspondance et la différence entre les deux formules A et B, A étant la formule de référence, nous procédons comme d'habitude, c'est-à-dire par abstraction de A, puis par application du filtrage de différence entre $Abst(A)$ et B. Nous obtenons ainsi une correspondance ρ et une différence D, comme cela a été défini au chapitre IV. Nous notons Σ, la paire formée de cette correspondance et de cette différence. $\Sigma(A)$ signifie $(\rho(A))[\overline{g \mapsto d}]$ pour $D = \{\overline{g \mapsto d}\}$ où tous les termes contenus dans D sont clos. □

Le principe général d'analogie consiste à appliquer cette correspondance et cette différence dans toute la preuve de la formule de référence, et à vérifier la validité des étapes de preuves obtenues pour B.

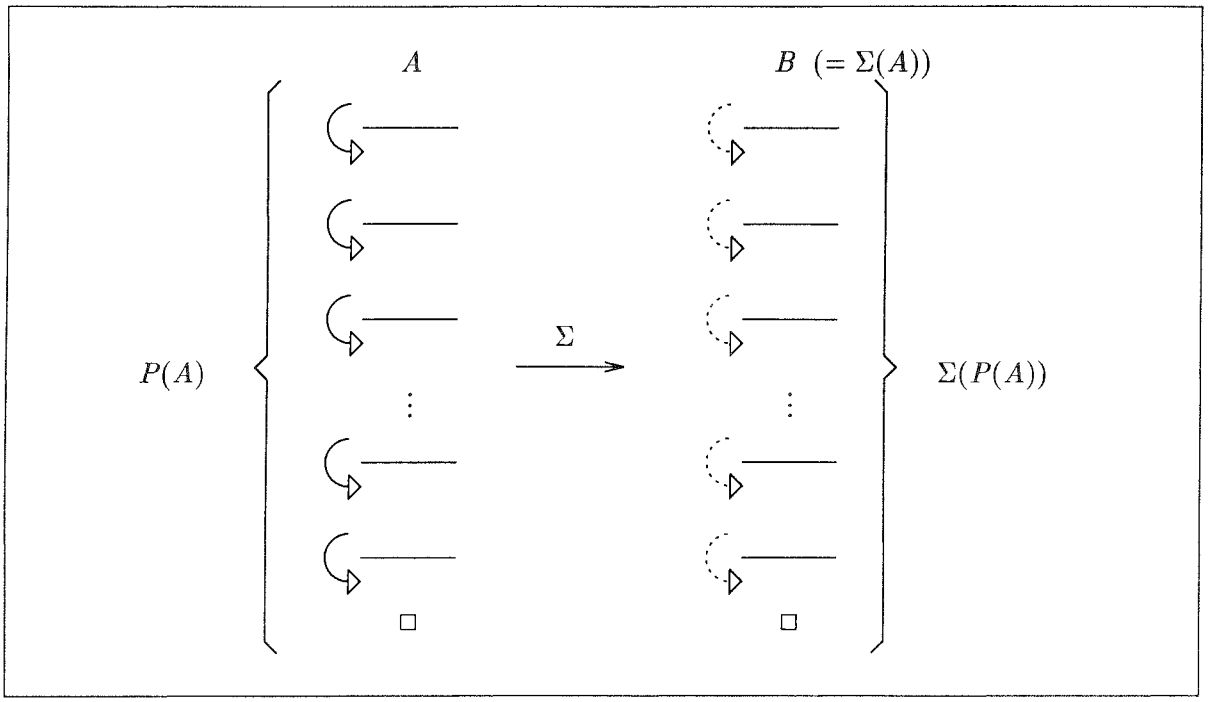


Figure V.1: Principe appliqué en général

Ce qui est obtenu n'est évidemment pas une preuve de B en général. En fonction des étapes valides, nous allons nous trouver dans un des cas de figures traités dans les sections suivantes. Nous allons distinguer quatre types d'analogies. Il s'agit d'une sorte de classification des cas de figures pouvant apparaître. Pour chacun de ces cas de figures, nous allons développer un exemple. Comme jusqu'à présent, nous nous sommes presque toujours restreints aux preuves par expansion, nous allons donner des exemples qui se situent dans des cadres tous différents. Nous voulons montrer par là que l'analogie peut s'appliquer à tout processus de preuve, que celui-ci ait pour but la synthèse de programmes, la validation de propriétés, ou même des preuves mathématiques (mais les exemples restent modestes dans ce domaine).

V.3 Analogie immédiate

Le cas le plus simple apparaît lorsque toutes les étapes de la preuve de B sont valides. C'est ce que nous appelons l'analogie immédiate. Ce type d'analogie est proche de la notion de similitude par correspondance. Mais nous pouvons avoir ici une différence non vide. La représentation de ce type d'analogies est simplement celui de la figure V.2.

Il est possible de la rencontrer en particulier lorsque l'on veut appliquer l'analogie entre deux structures de données voisines. Comme ce cas de figure est relativement simple, nous allons l'appliquer à un domaine qui l'est moins : la synthèse de programmes.

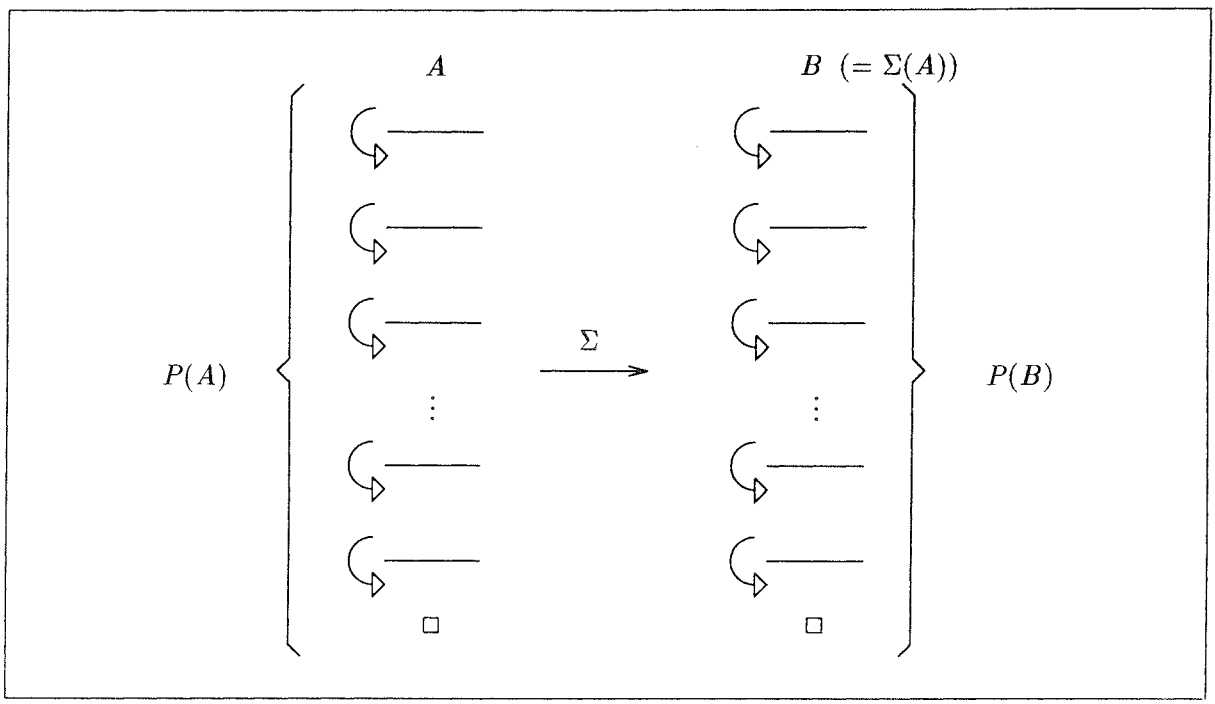


Figure V.2: Analogie immédiate

V.3.1 Analogie immédiate pour la synthèse de programmes

L'exemple que nous développons ici est inspiré de Bundy et Al. [BSW90]. Nous allons prendre comme référence une synthèse de prédicat donnée dans [BSW90], et développer à partir de là une analogie. La synthèse de programme logique consiste à transformer une spécification du premier ordre non-exécutable en un programme logique. Cette transformation résulte de la preuve de la spécification. Nous allons décrire la transformation donnée par Bundy et Al. qui construit un programme ôtant une élément d'une liste. A partir de là, nous appliquerons l'analogie pour construire un programme ôtant un élément d'un ensemble.

Exemple 2 La première étape est donc la description du processus permettant de passer d'une spécification à un programme. La démarche consiste à passer de la spécification d'une fonction qui supprime un élément d'une liste :

$$\begin{array}{l} \vdash \quad \forall x\, \forall K, L \quad \exists B \\ (\exists L_1, L_2 \quad L = L_1 @ L_2 \ \wedge \ K = L_1 @ [x|L_2]) \ \Leftrightarrow B \end{array}$$

à un programme logique réalisant cette opération :

$$\begin{array}{rcl} supp(x, [], L) & \Leftrightarrow & faux \\ supp(x, [Hd|Tl], L) & \Leftrightarrow & (x = Hd \ \wedge \ Tl = L) \vee \\ & & (\exists L' \quad \wedge \ L = [Hd|L'] \ \wedge \ supp(x, Tl, L')) \end{array}$$

Le programme logique est issu de la preuve de la spécification dont on extrait des témoins qui décrivent les différentes parties de ce programme. Le théorème à prouver est donc

Théorème A

$$\vdash \quad \forall x\, \forall K, L \quad \exists B$$
$$(\exists L_1, L_2 \quad \wedge\ L = L_1 @ L_2 \ \wedge\ K = L_1 @ [x|L_2]) \quad \Leftrightarrow B$$

où @ traduit la concaténation de deux listes, x est de type élément, K, L, L_1 et L_2 sont de type liste et B est un booléen. Au cours de la preuve, l'instanciation de B par *vrai* ou $faux$ permet de passer d'une proposition à un prédicat. Dans le déroulement de la preuve ci-dessous, ce que Bundy et al. appellent le *terme extrait* qui sera finalement la définition du prédicat recherché. Le prédicat à construire s'appelle *supp*, et la preuve se fait par induction (c'est le cadre habituel des travaux de Bundy et al.). Sans vouloir détailler leur démarche, nous utilisons partiellement leurs notations, dans l'unique but de faciliter la lecture. Dans ce qui suit, la colonne de gauche constitue les étapes de la preuve, la colonne de droite la construction du programme.

Preuve

Cas de base :	
$\vdash_\phi \exists x, B \quad (\exists L_1, L_2 \quad l = L_1 @ L_2$	Terme extrait :
$\wedge\ [] = L_1 @ [x|L_2]) \Leftrightarrow B$	$supp(x, [], L) \Leftrightarrow \phi$

L'indice ϕ signifie que la preuve de ce cas de base sera extrait le terme qui formera le membre droit de la formule $supp(x, [], L) \Leftrightarrow \phi$. Le cas inductif s'exprime comme ceci :

Cas inductif :$\forall L \quad \exists B$	
$\exists x, B \quad (\exists L_1, L_2 \quad L = L_1 @ L_2$	
$\wedge\ tl = L_1 @ [x|L_2]) \Leftrightarrow B$	Terme extrait :
$\vdash_\psi \exists B \quad (\exists L_1, L_2 \quad l = L_1 @ L_2$	$supp(x, [Hd|Tl], L) \Leftrightarrow \psi$
$\wedge\ [hd|tl] = L_1 @ [x|L_2]) \Leftrightarrow B$	

Résolution du cas de base :

Le cas de base se réduit à	
$\exists B \quad B \Leftrightarrow faux$	Terme extrait pour le cas de base
Puisqu'aucune liste L_1 ou L_2 ne peut satisfaire	$supp(x, [], L) \Leftrightarrow faux$
$[] = L_1 @ [x|L_2]$	

Résolution du cas inductif :

Il existe deux cas :	
$L_1 = [] \vee \exists Hd_1, Tl_1 \quad (L_1 = [Hd_1|Tl_1])$	
Pour le premier cas, la conclusion de l'induction se réduit à:	
$\exists B \quad (\exists L_2 \quad l = L_2 \wedge [hd|tl] = [x|L_2]) \Leftrightarrow B$	
soit, en réécrivant $[]@L \rightarrow L$, et en instanciant L_2 par l	Terme extrait :
$\exists B \quad (hd = x \ \wedge\ tl = l) \Leftrightarrow B$	
Ce qui est prouvé en considérant les quatre sous-cas	$supp(x, [Hd|Tl], L) \Leftrightarrow$
correspondant à $hd = x \ \vee\ hd \neq x$ et $tl = l \ \vee\ tl \neq l$	$(x = Hd \ \wedge\ Tl = L)$
où seul $hd = x \ \wedge\ tl = l$ valide $hd = x \ \wedge\ tl = l$	
ce qui clos le premier cas.	

Pour le second cas, l'induction se réduit à :

$\exists B \quad (\exists Hd_1, Tl_1, L_2$ $(l = [Hd_1 \mid Tl_1 @ L_2] \wedge [hd \mid tl] = [Hd_1 \mid Tl_1 @ [x \mid L_2]]) \Leftrightarrow B)$ qui se sépare en deux cas par $l = [] \vee \exists H', L' \quad l = [H' \mid L']$ Le premier instancie B par $faux$, puisque $[] = [Hd_1 \mid Tl_1 @ L_2]$ ne peut être vrai. L'autre cas se réduit à $\exists B \quad (\exists Tl_1, L_2 \quad L' = Tl_1 @ L_2$ $\wedge\, hd = Hd_1 \wedge tl = Tl_1 @ [x \mid L_2]) \Leftrightarrow B$ des deux cas $hd = Hd_1 \vee hd \neq Hd_1$, seul le premier est valide on obtient $\exists B \quad (\exists Tl_1, L_2 \quad L' = Tl_1 @ L_2$ $\wedge\, tl = Tl_1 @ [x \mid L_2]) \Leftrightarrow B$ qui filtre avec l'hypothèse d'induction en instanciant L par L' et renommant L_1 en Tl_1	L'utilisation de l'hypothèse d'induction suggère le terme $supp(x, tl, L)$ et le terme $\exists L'$ $L = [Hd \mid L'] \wedge$ $supp(x, Tl, L')$ pour tout le second cas

Globalement, pour le cas inductif, le terme extrait est :

$$(hd = x \wedge l = tl) \vee (\exists L' \quad l = [hd \mid L'] \wedge supp(x, tl, L'))$$

et la formule définissant le prédicat :

$$\begin{aligned} supp(x, [], L) &\Leftrightarrow faux \\ supp(x, [Hd \mid Tl], L) &\Leftrightarrow (x = Hd \wedge Tl = L) \vee \\ &\quad (\exists L' \quad L = [Hd \mid L'] \wedge supp(x, Tl, L')) \end{aligned}$$

Cette synthèse n'est pas triviale. Le fait de pouvoir la réutiliser pour des cas voisins a un intérêt évident. Envisageons de produire une démarche semblable pour un prédicat *supp'* effaçant un élément d'un ensemble. Le théorème correspondant pour les ensembles peut s'écrire :

Théorème B

$$\begin{aligned} &\vdash \quad \forall x \, \forall E, F \quad \exists B \\ &(\exists E_1, E_2 \quad E = E_1 \cup E_2 \wedge F = E_1 \cup adj(x, E_2)) \quad \Leftrightarrow B \end{aligned}$$

Par abstraction du théorème de référence et application du filtrage de différence entre le schéma obtenu et le théorème 2, nous obtenons les correspondances et différences suivantes, en utilisant, comme nous l'avons déjà dit, le type *bool* pour le booléen B, et le type ι pour les autres :

Correspondance	Différence
$@ \mapsto \lambda x_1 x_2 . x_1 \cup x_2$	$[] \not\subseteq \emptyset$
	$\lambda xz.[x \mid z] \not\subseteq \lambda xz.adj(x, z)$

En appliquant la correspondance et en remplaçant les parties gauches de la différence par les parties droites, la preuve transformée devient (nous avons renommé les variables libres, afin d'être plus en accord avec les ensembles, mais cela n'a aucune incidence sur la preuve) :

Cas de base :
$\vdash_\phi \exists x, B \quad (\exists E_1, E_2 \quad e = E_1 \cup E_2$
$\wedge \emptyset = E_1 \cup adj(x, E_2)) \Leftrightarrow B$
Cas inductif :$\forall E \quad \exists B$
$\exists x, B \quad (\exists E_1, E_2 \quad E = E_1 \cup E_2$
$\wedge s = E_1 \cup adj(x, E_2)) \Leftrightarrow B$
$\vdash_\psi \exists B \ (\exists E_1, E_2 \quad e = E_1 \cup E_2$
$\wedge adj(el, s) = E_1 \cup adj(x, E_2)) \Leftrightarrow B$
Le cas de base se réduit à
$\exists B \quad B \Leftrightarrow faux$
Puisqu'aucun ensemble E_1 ou E_2 ne peut satisfaire
$\emptyset = E_1 \cup adj(x, E_2)$
Il existe deux cas :
$E_1 = \emptyset \vee \exists El_1, S_1 \quad (E_1 = adj(El_1, S_1))$
Pour le premier cas, la conclusion de l'induction se réduit à:
$\exists B \quad (\exists E_2 \quad e = E_2 \wedge adj(el, s) = adj(x, E_2))$
soit, en réécrivant $\emptyset \cup E \rightarrow E$, et en instanciant E_2 par e
$\exists B \quad (el = x \wedge s = e) \Leftrightarrow B$
Ce qui est prouvé en considérant les quatre sous-cas
correspondant à $el = x \vee el \neq x$ et $s = e \vee s \neq e$
où seul $el = x \wedge s = e$ valide $el = x \wedge s = e$
ce qui clos le premier cas.

Terme extrait :
$supp'(x, \emptyset, E) \Leftrightarrow \phi$

Terme extrait :
$supp'(x, adj(El, S), E) \Leftrightarrow \psi$

Terme extrait pour
le cas de base
$supp'(x, \emptyset, E) \Leftrightarrow faux$

$\Leftrightarrow B$
Terme extrait :
$supp'(x, adj(El, S), E) \Leftrightarrow$
$(x = El \wedge S = E)$

$\exists B \quad (\exists El_1, S_1, E_2$
$(e = adj(El_1, S_1 \cup E_2) \wedge adj(el, s) = adj(El_1, S_1 \cup adj(x, E_2)))$
$\Leftrightarrow B)$
qui se sépare en deux cas par
$e = \emptyset \vee \exists H', E' \quad e = adj(H', E')$
Le premier instancie B par $faux$, puisque $\emptyset = adj(El_1, S_1 \cup E_2)$
ne peut être vrai. L'autre cas se réduit à
$\exists B \quad (\exists S_1, E_2 \quad E' = S_1 \cup E_2$
$\wedge el = El_1 \wedge s = S_1 \cup adj(x, E_2)) \Leftrightarrow B$
des deux cas $el = El_1 \vee el \neq El_1$, seul le premier est valide
on obtient $\exists B \quad (\exists S_1, E_2 \quad E' = S_1 \cup E_2$
$\wedge s = S_1 \cup adj(x, E_2)) \Leftrightarrow B$
qui filtre avec l'hypothèse d'induction en instanciant E par E'
et renommant E_1 en S_1

E'utilisation de
l'hypothèse d'induction
suggère le terme
$supp'(x, s, E)$
et le terme
$\exists E'$
$E = adj(El, E') \wedge$
$supp'(x, S, E')$
pour tout le second cas

Globalement, pour le cas inductif, le terme extrait est :

$$(el = x \wedge e = s) \vee (\exists E' \quad e = adj(el, E') \wedge supp'(x, s, E'))$$

et la formule définissant le prédicat :

$$\begin{aligned} supp'(x, \emptyset, E) &\Leftrightarrow faux \\ supp'(x, adj(El, S), E) &\Leftrightarrow (x = El \wedge S = E) \vee \\ &\quad (\exists E' \quad E = adj(El, E') \wedge supp'(x, S, E')) \end{aligned}$$

■

Nous pouvons vérifier toutes les étapes de cette synthèse, toutes sont correctes. Il n'y a donc rien à faire, nous avons directement une preuve de la formule objet et le texte du prédicat correspondant. C'est pour cette raison que ce type d'analogie s'appelle analogie immédiate. On pourrait même être tenté de l'appeler analogie triviale. Mais ce serait une erreur, pour la simple raison que la différence entre les formules n'est pas vide, par conséquent, il n'est pas possible de dire a priori si la formule objet est valide, et de plus toutes les propriétés des listes ne sont pas forcément valables pour les ensembles et réciproquement. De plus, nous fabriquons automatiquement la proposition de preuve.

V.4 Analogie par saut

Nous abordons ici le deuxième type d'analogies. L'opération de base est celle que nous avons décrit comme principe général. Ce que nous appelons type d'analogie dépend en fait du résultat de cette opération. Dans la section précédente, toutes les étapes de la preuve ainsi construite étaient valides. Ce n'est plus le cas ici. Par contre, il est possible, en *sautant* les étapes qui ne sont pas valides, de reconstituer une preuve valide. Le schéma correspond à la figure V.3.

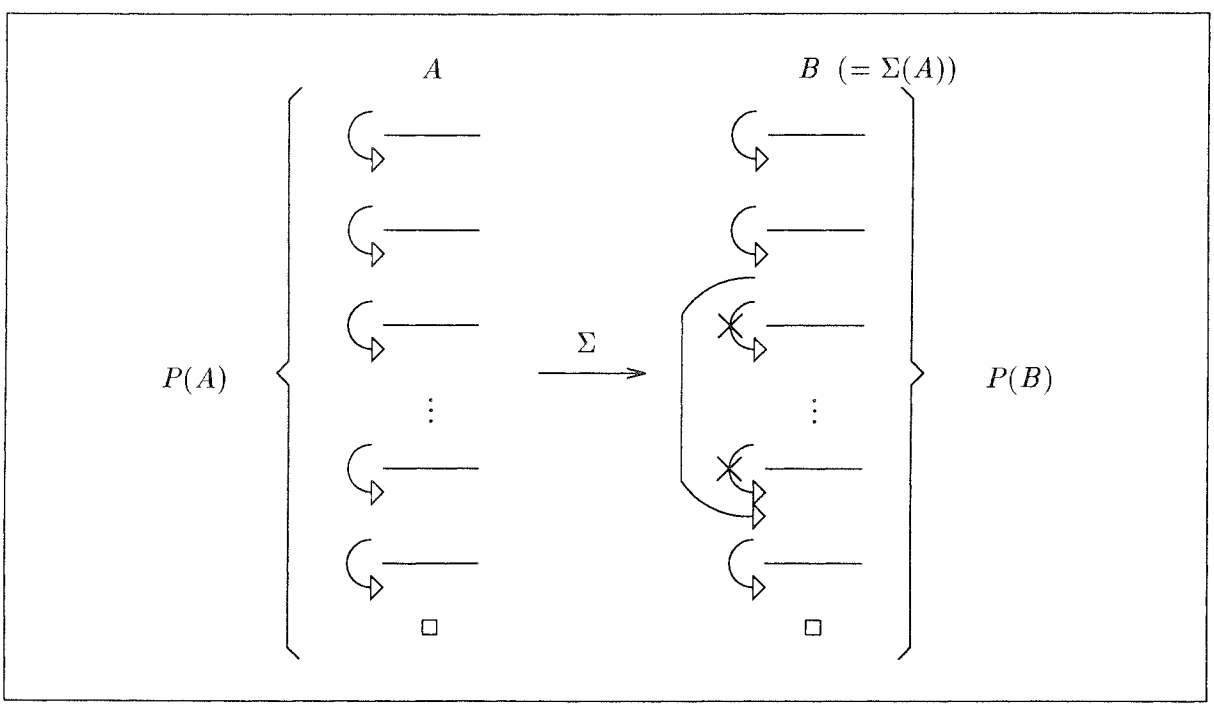
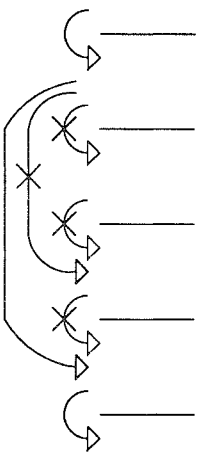


Figure V.3: Analogie par saut

En appliquant la correspondance et la différence à la preuve de référence, nous obtenons une séquence d'étapes de preuve dont un certains nombre ne sont pas valides. Mais une déduction valide peut enjamber les déductions erronées. C'est pourquoi nous l'appelons analogie par saut. Dans ce cas, nous voyons qu'il ne convient pas seulement de vérifier les étapes de la preuve

obtenue, mais, lorsque celle-ci n'est pas valide, il est nécessaire d'en faire un peu plus. Lorsque nous rencontrons une déduction qui n'est pas valide, nous essayons de sauter cette étape, puis, s'il y a toujours échec, l'étape suivante, etc... On peut exprimer cela par la figure suivante :



□

Un cas probable de ce cas de figure est, lorsque l'on applique l'analogie à un cas moins complexe que le théorème de référence. "Moins complexe" étant encore une notion intuitive. Nous allons voir dans l'exemple qui suit que cela peut se produire lorsque l'on transforme une preuve utilisant le produit en une preuve utilisant la somme.

V.4.1 Analogie par saut pour l'analyse

Nous allons nous placer dans le cadre de preuves d'analyse dans le domaine des réels. Le théorème qui nous servira de référence est tiré de travaux sur l'analogie [BCP88]. L'analogie que nous allons traiter est également utilisée avec succès dans l'application décrite dans [BCP88]. Mais d'une part, leur système d'analogie est spécialisé pour les preuves de ce type – analyse sur les réels –, et d'autre part, la réalisation est assez différente. En effet, alors que les étapes de la preuve de référence leur fait suggérer des étapes de la preuve objet, nous allons opérer la transformation en une seule étape, après avoir recueilli l'information sur les formules. A aucun moment, nous ne recherchons de nouvelles étapes de preuve. Nous tentons simplement de réappliquer celles de la preuve de référence, et, lorsqu'il y a un problème, de réagir en fonction de sa nature. De plus, plusieurs points diffèrent avec la façon qu'ont Brock Cooper et Pierce d'aborder le problème. En premier lieu, leur reconnaissance de l'analogie est basée sur l'expérience, à l'aide d'associations. Ensuite, le principe qu'ils utilisent est de proposer des étapes de preuves. Enfin, lorsqu'un lemme est généré, ils peuvent – récursivement en quelque sorte – le résoudre par analogie. Pour notre part, nous tentons de réutiliser l'information de la preuve de départ en générant automatiquement et en une seule étape une preuve – imparfaite – de la formule à prouver. De plus notre façon de reconnaître l'analogie ne dépend pas du cadre dans lequel nous nous plaçons, ni de l'expérience.

Exemple 3 Le théorème de référence est le suivant :

$$\lim_{x\to\infty} f(x) = l \ \wedge \ \lim_{x\to\infty} g(x) = l' \Rightarrow \lim_{x\to\infty} f.g(x) = l.l'$$

avec la définition d'une limite en l'infini donnée par :

$$\lim_{x\to\infty} f(x) = l \Leftrightarrow \forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |f(x) - l| < \varepsilon$$

Le théorème ci-dessus se traduit donc par la formule logique suivante :

Théorème A

$$\begin{array}{c}(\forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |f(x) - l| < \varepsilon \\ \wedge \forall\varepsilon \quad \exists n' \quad \forall x \quad x < n' \vee |g(x) - l| < \varepsilon) \\ \Rightarrow \forall\varepsilon \quad \exists n'' \quad \forall x \quad x < n'' \vee |f.g(x) - l.l'| < \varepsilon\end{array}$$

Une preuve de ce théorème peut être :

$$\begin{array}{c}(\forall\varepsilon \quad \exists n' \quad \forall x \quad x < n' \vee |f(x) - l| < \varepsilon \\ \wedge \forall\varepsilon \quad \exists n'' \quad \forall x \quad x < n'' \vee |g(x) - l| < \varepsilon)\end{array}$$

(1) $$\Rightarrow \forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |f(x) - l| \,.\, |g(x) - l'| < \varepsilon.\varepsilon$$

(2) $$\Rightarrow \forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |(f(x) - l).(g(x) - l')| < \varepsilon.\varepsilon$$

(3) $$\Rightarrow \forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |f.g(x) - l.l' - (l.(f(x) - l') + l'.(f(x) - l))| < \varepsilon.\varepsilon$$

$$\text{avec } \forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |l.(f(x) - l') + l'.(f(x) - l)| < l.\varepsilon + l'.\varepsilon$$

$$\text{or } |A| \leq |A + B| + |B|$$

(4) $$\Rightarrow \forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |f.g(x) - l.l'| < \varepsilon.(\varepsilon + l + l')$$

(5) $$\Rightarrow \forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |f.g(x) - l.l'| < \varepsilon$$

Nous avons étiqueté les implications comme des étapes de preuve. Le théorème relatif à la somme des limites s'exprime par :

$$\lim_{x\to\infty} f(x) = l \wedge \lim_{x\to\infty} g(x) = l' \Rightarrow \lim_{x\to\infty} f + g(x) = l + l'$$

ce qui s'écrit :

Théorème B

$$\begin{array}{c}(\forall\varepsilon \quad \exists n \quad \forall x \quad x < n \vee |f(x) - l| < \varepsilon \\ \wedge \forall\varepsilon \quad \exists n' \quad \forall x \quad x < n' \vee |g(x) - l| < \varepsilon) \\ \Rightarrow \forall\varepsilon \quad \exists n'' \quad \forall x \quad x < n'' \vee |f + g(x) - (l + l')| < \varepsilon\end{array}$$

Par abstraction du théorème A et application du filtrage de différence avec le théorème B, nous obtenons :

Correspondance	Différence
$f \mapsto \lambda x.f(x)$	$\lambda x_1 x_2.x_1.x_2 \overset{?}{\neq} \lambda x_1 x_2.x_1 + x_2$
$g \mapsto \lambda x.g(x)$	

Nous pouvons en effet considérer les opérateurs, comme la valeur absolue ou l'ordre, comme faisant partie du langage. Dans le cas contraire, il y aurait correspondance, et le résultat serait le même. En appliquant la différence à la preuve, nous obtenons (il suffit de remplacer les produits par des sommes) :

$$\begin{array}{l}(\forall\varepsilon \ \ \exists n' \ \ \forall x \ \ x < n' \vee \ |f(x) - l| < \varepsilon \\ \wedge\, \forall\varepsilon \ \ \exists n'' \ \ \forall x \ \ x < n'' \vee \ |g(x) - l| < \varepsilon)\end{array}$$

$$(1) \qquad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f(x) - l| + |g(x) - l'| < \varepsilon + \varepsilon$$

$$(2) \qquad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f(x) - l + g(x) - l'| < \varepsilon + \varepsilon$$

$$(3) \quad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f + g(x) - l + l' - (l + (f(x) - l') + l' + (f(x) - l))| < \varepsilon + \varepsilon$$

$$\text{avec } \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |l + (f(x) - l') + l' + (f(x) - l)| < l + \varepsilon + l' + \varepsilon$$

$$\text{or } \ |A| \leq |A + B| + |B|$$

$$(4) \qquad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f + g(x) - l + l'| < \varepsilon + (\varepsilon + l + l')$$

$$(5) \qquad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f + g(x) - l + l'| < \varepsilon$$

Par vérification des étapes de preuve, on se rend compte que seules les étapes (1) et (2) restent valides. Par contre, de la prémisse de l'étape (3), nous pouvons sauter directement à la conclusion de l'étape (5). Ce qui donne la preuve :

$$\begin{array}{l}\forall\varepsilon \ \ \exists n' \ \ \forall x \ \ x < n' \vee \ |f(x) - l| < \varepsilon \\ \wedge\, \forall\varepsilon \ \ \exists n'' \ \ \forall x \ \ x < n'' \vee \ |g(x) - l| < \varepsilon)\end{array}$$

$$(1) \quad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f(x) - l| + |g(x) - l'| < \varepsilon + \varepsilon$$

$$(2) \quad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f(x) - l + g(x) - l'| < \varepsilon + \varepsilon$$

$$(3') \qquad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f + g(x) - l + l'| < \varepsilon$$

■

La preuve du second théorème apparaît en fait comme un affaiblissement de la preuve de référence. Mais encore un fois, ce qui est valable pour le produit ne l'est pas nécessairement pour la somme. Or, nous voyons dans cet exemple que l'information contenue dans la preuve de référence est suffisante pour montrer la preuve objet. C'est donc un cas où l'analogie doit être utilisée. Elle rend la machine capable de passer de la preuve du théorème de référence à la preuve objet sans aide extérieure, et sans utiliser un prouveur. Cette transformation est uniquement une transformation par analogie. Il n'y a pas construction d'une nouvelle preuve, mais simplement adaptation de la preuve de référence. Pour l'exemple donné, la question qui vient naturellement à l'esprit est : "que se passe t-il dans l'autre sens, lorsque l'on essaie de passer de la preuve pour la somme à la preuve pour le produit?" Il est facile de voir que la preuve du théorème 2 ne suffira pas à montrer le théorème 1. Par conséquent, l'analogie ne peut pas être suffisante pour

ce processus. Nous entendons par là l'analogie seule, sans intervention du prouveur auquel elle doit être associée. Un processus d'analogie va immanquablement laisser un trou dans la preuve qui devra être comblé, par exemple en générant un lemme, qui pourra alors être prouvé, soit par analogie avec un théorème présent dans la base de théorèmes et de preuves à disposition, soit par un processus classique de démonstration automatique, ou encore grâce à une intervention intelligente, donc humaine.

V.5 Analogies horizontales

Nous appelons ce type d'analogies *horizontales*, parce qu'elles utilisent une éventuelle relation d'implication entre les formules ρA et B (où ρ est la correspondance qui lie les formules lorsque $(\rho A)[\overline{l \mapsto r}] = B$ pour D, la différence). Dans les autres cas, nous observons uniquement la preuve transformée et regardons comment la modifier afin qu'elle devienne une preuve correcte de B. Une condition suffisante pour la validité de B est de montrer que $A \Rightarrow B$ est une formule valide. Si nous possédons une preuve de cette formule, l'application d'une règle de type coupure doit constituer une preuve de B. Mais la formalisation de la construction de cette preuve dépend quelque peu du formalisme de représentation des preuves que l'on utilise. Nous pouvons grossièrement représenter cette construction par la figure V.4.

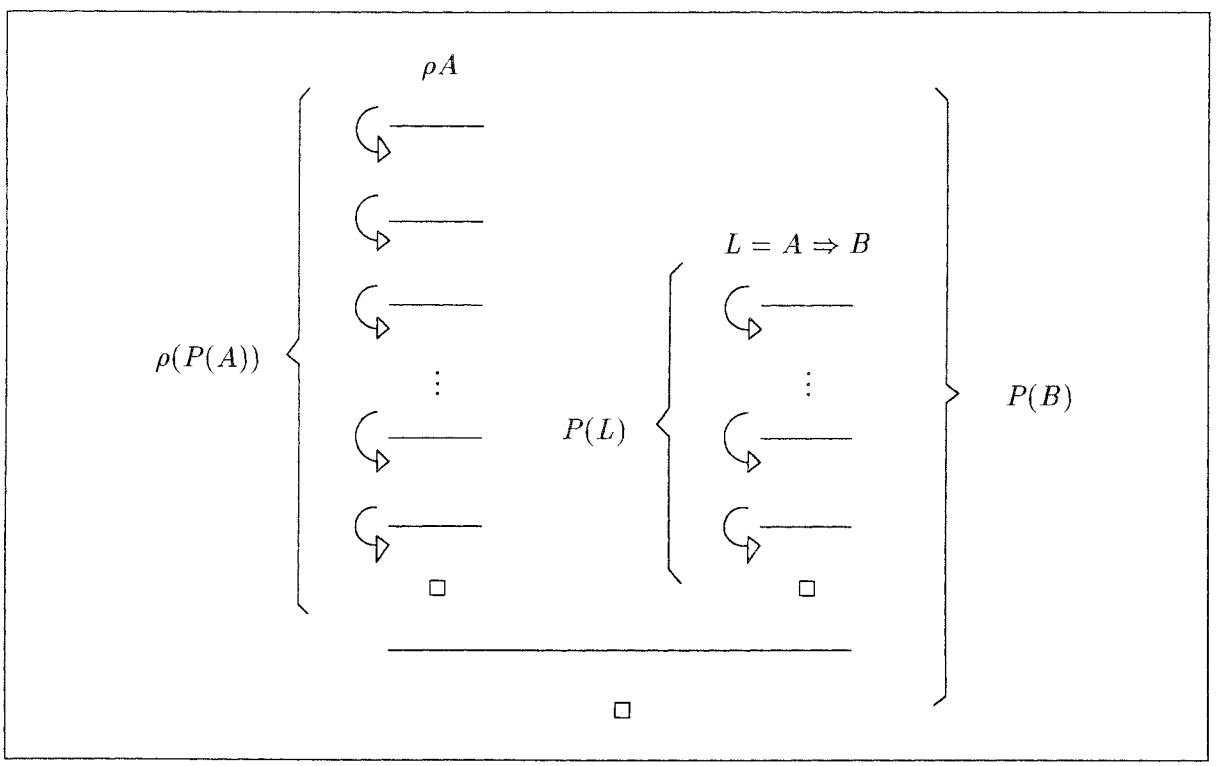


Figure V.4: Analogies horizontales

Pour la première fois, nous introduisons un lemme (L), dont la preuve sert à construire la preuve objet. Nous allons voir deux façons d'envisager ce problème. Elles ne sont pas incompatibles, et peuvent être essayée successivement. La première consiste à considérer globalement le lemme $\rho A \Rightarrow B$, la seconde à générer des lemmes plus fins tirés de la différence.

V.5.1 Analogie horizontale structurelle

Cette façon de faire consiste à générer directement le lemme $\rho A \Rightarrow B$. Il est bien sûr possible que ce lemme soit faux. Mais s'il est valide, sa preuve associée à celle de ρA forme une preuve de B. En particulier, en utilisant ce que nous avons fait auparavant, nous savons que si $A \preceq B$, alors $A \Rightarrow B$. Par conséquent, si nous avons une dérivation $A \rightarrow^* B$ montrant que $A \preceq B$, nous pouvons construire une preuve de B (dans le cadre des preuves par expansion). Le principal problème qui se pose ici est celui du formalisme utilisé. Il faut transcrire la preuve de $A \preceq B$ dans le formalisme adéquat, et utiliser une règle de coupure qui se rapporte à ce même formalisme. Mais cette analogie permet de généraliser une partie de ce que nous avons évoqué au chapitre I. On peut en effet envisager de montrer automatiquement tout B tel que $A \preceq B$ à partir d'une preuve de A, moyennant une adaptation au système de preuve utilisé. Si nous ramenons le problème $A \Rightarrow B$ à $A \preceq B$ qui n'en est qu'un cas particulier, c'est simplement parce qu'en général, le lemme $A \Rightarrow B$ peut être plus compliqué que B elle-même. Par contre, si nous tombons dans ce cas particulier, il est possible de donner un processus automatique. Sinon, le lemme $A \Rightarrow B$ peut être généré. Voyons à présent une méthode plus fine utilisant la différence.

V.5.2 Analogie horizontale par la différence

Il s'agit, comme précédemment, de générer un ou plusieurs lemmes suffisants pour construire une preuve de B. Mais cette fois, nous allons utiliser l'information de la différence. Prenons par exemple $D = \{A' \stackrel{?}{\neq} B'\}$, où A' et B' apparaissent à des occurrences positives respectivement dans ρA et B. Le lemme $A' \Rightarrow B'$ est alors suffisant pour montrer $\rho A \Rightarrow B$, et pour construire une preuve de B. Si A' et B' apparaissent à des occurrences négatives, c'est le lemme $B' \Rightarrow A'$ qui doit être généré. Notons au passage que pour $\{l \stackrel{?}{\neq} k\} \in D$, l et k apparaissent nécessairement à des occurrences de mêmes signes. Nous allons dans un premier temps définir les lemmes de différence et montrer que leur validité implique celle de B, puis, nous montrerons comment reconstruire une preuve de B.

A Lemmes de différence

Signalons, avant de donner cette définition, que l'idée d'utiliser les échecs du filtrage pour générer des lemmes a déjà été évoquée par Boy de la Tour et Caferra dans [BdlTC88]. Mais là encore, dans notre cas, le filtrage s'effectue sur les formules non pas sur les types – donc les preuves, puisqu'ils travaillent dans le cadre de l'isomorphisme de Curry-Howard. La définition des lemmes de différence s'exprime ainsi :

Définition 4 A tout couple $\{l \stackrel{?}{\neq} r\} \in D$, est associé un lemme de différence défini par :

- $l \Rightarrow r$ si l et r apparaissent à des occurrences positives respectivement dans ρA et dans B,
- $r \Rightarrow l$ sinon.

□

La validité de tous les lemmes de différence ajoutée à la validité de A implique la validité de B. C'est ce qu'exprime le lemme suivant.

Lemme 5 *Soit A une formule valide, et B une formule telle que $\Sigma(A) = B$ avec $\Sigma = (\rho, D)$ où ρ est une correspondance et D une différence résultant du filtrage de différence entre Abst(A) et B. Soient $\{\overline{L_n}\}$ les n lemmes de différence formés à partir des n couples de D. Alors,*

$$\models A \text{ et } \models L_i \quad \textit{pour } i = 1, \ldots, n \textit{ implique } \models B$$

Preuve Par induction sur n.

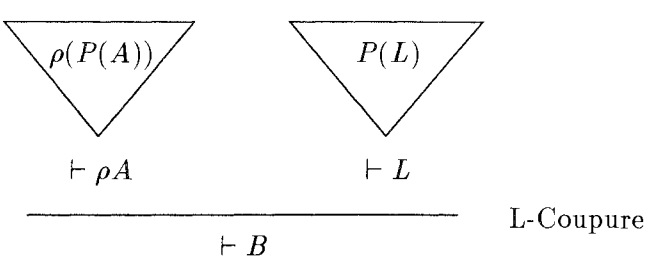
- $n = 0$: $\models A$ implique $\models \rho A$ est déduit du lemme 6 (du chapitre III).
- $n = 1$: $D = \{l \stackrel{?}{\neq} r\}$, et $\Sigma(A) = (\rho A)[l \mapsto r]$.
  - l et r apparaissent à des occurrences positives. Par induction structurelle sur A : pour $A = l$, $\models l$ implique $\models r$ lorsque $l \Rightarrow r$ est trivial. Il en est de même pour $A = A_1 \vee l$ et $A = A_1 \wedge l$. Pour le cas inductif, l'hypothèse d'induction s'applique immédiatement pour $A = A_1 \vee A_2$ lorsque l apparaît à une occurrence positive dans A_i, et pour $A = A_1 \wedge A_2$ lorsque l apparaît à une occurrence positive dans A_i.
  - l et r apparaissent à des occurrences négatives. Est directement déduit du fait que $(l \Rightarrow r) \Leftrightarrow (\neg r \Rightarrow \neg l)$.
- $\Sigma(A) = (\rho A)[\overline{l_n \mapsto r_n}][l_{n+1} \mapsto r_{n+1}]$ et $\models (\rho A)[\overline{l_n \mapsto r_n}]$. Ce cas est identique au cas de base.

□

Passons maintenant à la construction de la preuve de B.

B Construction d'une nouvelle preuve

Cette construction dépend nécessairement de la représentation des preuves qui est utilisée. Nous en utilisons une, arbitrairement, qui peut être adaptée. Pour un lemme de différence, nous pouvons voir les choses ainsi :



Cette règle est bien sûr très pratique, puisqu'elle traduit directement ce que nous attendions. Le problème qui se pose, est de trouver un équivalent n'utilisant que les règles habituelles. On tombe alors sur un problème similaire à celui de l'élimination des coupures au sens classique. Nous n'allons pas le développer ici, nous allons simplement donner un exemple.

Exemple 6 Cet exemple est tiré de [BdlTC87].

Formule de référence : $A = (p(a) \vee q(b)) \wedge \forall x\ (p(x) \Rightarrow q(x)) \Rightarrow \exists x\ q(x)$

Formule objet :

$B = (p(a) \vee q(b)) \wedge \forall x\ (p(x) \Rightarrow r(x)) \wedge \forall x\ (r(x) \Rightarrow q(x)) \Rightarrow \exists x\ q(x)$

Par abstraction de A et application du filtrage de différence entre le schéma obtenu et B, il vient :

Correspondance	Différence
$p \mapsto \lambda x.p(x)$	$\forall x\ (p(x) \Rightarrow q(x))$
$q \mapsto \lambda x.q(x)$	$\not\neq$
$a \mapsto a$	$\forall x\ (p(x) \Rightarrow r(x)) \wedge \forall x\ (r(x) \Rightarrow q(x))$
$b \mapsto b$	

Or, cette différence correspond à des occurrences négatives. Le lemme généré sera donc

$$L = (\forall x\ (p(x) \Rightarrow r(x)) \wedge \forall x\ (r(x) \Rightarrow q(x))) \ \Rightarrow\ \forall x\ (p(x) \Rightarrow q(x))$$

La preuve de B sera alors, en utilisant la règle *L-coupure*,

$$\dfrac{\dfrac{\vdots}{\vdash (p(a) \vee q(b)) \wedge \forall x\ (p(x) \Rightarrow q(x)) \ \Rightarrow\ \exists x\ q(x)} \qquad \dfrac{\vdots}{\vdash (\forall x\ (p(x) \Rightarrow r(x)) \wedge \forall x\ (r(x) \Rightarrow q(x))) \ \Rightarrow\ \forall x\ (p(x) \Rightarrow q(x))}}{\vdash (p(a) \vee q(b)) \wedge \forall x\ (p(x) \Rightarrow r(x)) \wedge \forall x\ (r(x) \Rightarrow q(x)) \ \Rightarrow\ \exists x\ q(x)}\ \textit{L-coupure}$$

ou, avec la coupure classique (notée Cut) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\vdash (p(a) \vee q(b)) \wedge \forall x\ (p(x) \Rightarrow q(x)) \ \Rightarrow\ \exists x\ q(x)}}{(p(a) \vee q(b)), \forall x\ (p(x) \Rightarrow r(x)), \forall x\ (r(x) \Rightarrow q(x)) \vdash \exists x\ q(x)}\ E_{\Rightarrow} \qquad \dfrac{\dfrac{\vdots}{\vdash (\forall x\ (p(x) \Rightarrow r(x)) \wedge \forall x\ (r(x) \Rightarrow q(x))) \ \Rightarrow\ \forall x\ (p(x) \Rightarrow q(x))}}{\forall x\ (p(x) \Rightarrow r(x)), \forall x\ (r(x) \Rightarrow q(x)) \vdash \forall x\ (p(x) \Rightarrow q(x))}\ E_{\Rightarrow}}{(p(a) \vee q(b)) \wedge \forall x\ (p(x) \Rightarrow r(x)) \wedge \forall x\ (r(x) \Rightarrow q(x)) \vdash \exists x\ q(x)}\ \textit{Cut}}{\vdash (p(a) \vee q(b)) \wedge \forall x\ (p(x) \Rightarrow r(x)) \wedge \forall x\ (r(x) \Rightarrow q(x)) \ \Rightarrow\ \exists x\ q(x)}\ I_{\Rightarrow}$$

■

Il est important de noter ici que la preuve du lemme de différence ne tient plus de l'analogie. En effet, il n'est pas possible de puiser dans les informations de la preuve de référence pour prouver ce lemme. Par contre, ce lemme pourrait être prouvé par analogie avec un autre théorème existant dans une base de théorèmes, puisqu'il ne s'agit que de la transitivité de l'implication. En utilisant un processus d'analogie, cette preuve ne doit donc pas poser de problème.

Pour ce type d'analogie, nous pouvons voir que si l'idée intuitive est simple, la mise en pratique n'est pas évidente s'il faut éliminer la règle L – *coupure*. Cette méthode doit être appliquée de façon spécifique à chaque représentation.

V.6 Analogie incomplète

Nous abordons maintenant la dernière forme d'analogie – selon notre façon de voir – que nous appelons analogie incomplète. Il s'agit en fait du minimum que l'on puisse faire par analogie.

C'est-à-dire que l'on va appliquer le processus général, puis, pour les parties de la preuve générée qui ne sont pas correctes, extraire les lemmes correspondants, dont les preuves éventuelles peuvent se glisser dans la preuve de B pour la compléter. Le schéma représentant ce cas de figure est le suivant :

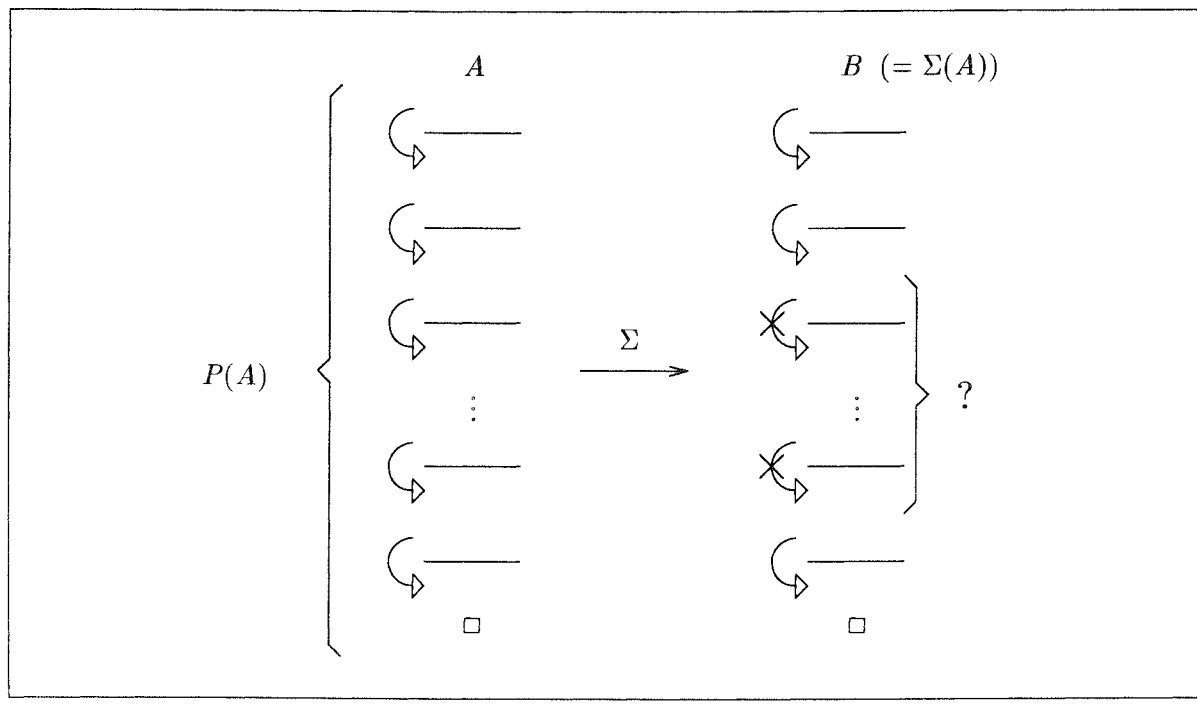


Figure V.5: Analogie incomplète

La différence avec l'analogie par saut, est qu'il n'existe pas de saut possible pour enjamber les étapes qui ne sont pas valides. Par conséquent, pour ces étapes incorrectes, un lemme est généré. La définition des lemmes générés est la suivante.

Définition 7 Soit A une formule, $P(A)$ une preuve de cette formule, et B une formule telle que $\Sigma(A) = B$. Pour toute séquence incorrecte $D_1 \rightarrow D_2 \rightarrow \cdots \rightarrow D_n$ de $P' = \Sigma(P(A))$, un lemme permettant de passer de D_1 à D_n est généré. Sa forme dépend du type de déductions. □

Cette définition est dépendante du type de déduction. Nous allons appliquer ce type d'analogie à une preuve équationnelle. Une fois n'est pas coutume, nous allons utiliser une formule objet non valide. Par conséquent, les lemmes générés ne seront pas valides. De plus, nous allons voir que l'information tirée de tels échecs peut être intéressante.

Exemple 8 Considérons l'ensemble E, contenant e, muni de l'opérateur $+$. $G = (E, +)$ est un groupe, est défini par les axiomes suivants :

$$Ax = \begin{array}{lrcll} (1) & x + e & = & x & \text{Elément neutre} \\ (2) & x + i(x) & = & e & \text{Inverse} \\ (3) & (x + y) + z & = & x + (y + z) & \text{Associativité} \end{array}$$

Une preuve du théorème $Ax \Rightarrow e + x = x$ peut être

$$\begin{array}{lll}
(1) & e+x & = e+(x+e) \\
(2) & & = e+(x+(i(x)+i(i(x)))) \\
(3) & & = e+((x+i(x))+i(i(x))) \\
(2) & & = e+(e+i(i(x))) \\
(3) & & = (e+e)+i(i(x)) \\
(1) & & = e+i(i(x)) \\
(2) & & = (x+i(x))+i(i(x)) \\
(3) & & = x+(i(x)+i(i(x))) \\
(2) & & = x+e \\
(1) & & = x
\end{array}$$

Or, il est connu que si l'on commute le membre gauche de l'un des deux premiers axiomes, on passe d'un groupe à un système "gauche-droite", pour lequel le théorème énoncé n'est plus vrai. Nous allons tenter de montrer cette formule pour un système gauche-droite par analogie et regarder les lemmes générés. $G' = (E, +)$ est donc défini par les axiomes :

$$\begin{array}{llrcll}
 & (1') & x+e & = & x & \text{Elément neutre} \\
Ax' = & (2') & i(x)+x & = & e & \text{Inverse} \\
 & (3') & (x+y)+z & = & x+(y+z) & \text{Associativité}
\end{array}$$

Nous allons donc tenter l'analogie pour la formule $Ax' \Rightarrow e + x = x$. Par abstraction, puis filtrage de différence, la différence obtenue est :

$$\lambda x.x + i(x) \overset{?}{\neq} \lambda x.i(x) + x$$

En remplaçant les occurrences de $x + i(x)$ par $i(x) + x$, nous obtenons,

$$\begin{array}{lll}
(1') & e+x & = e+(x+e) \\
(2') & & = e+(x+(i(i(x))+i(x))) \\
(3') & X & = e+((i(x)+x)+i(i(x))) \\
(2') & & = e+(e+i(i(x))) \\
(3') & & = (e+e)+i(i(x)) \\
(1') & & = e+i(i(x)) \\
(2') & & = (i(x)+x)+i(i(x)) \\
(3') & X & = x+(i(i(x))+i(x)) \\
(2') & & = x+e \\
(1') & & = x
\end{array}$$

où les marques X étiquettent les étapes non valides. Par conséquent, les deux lemmes générés sont

$$L_1 \ : \ e+(x+(i(i(x))+i(x))) = e+((i(x)+x)+i(i(x)))$$
$$\text{et } L_2 \ : \ (i(x)+x)+i(i(x)) = x+(i(i(x))+i(x))$$

En fait, ils se ramènent simplement à

$$e+(x+e) = e+(e+i(i(x)))$$
$$\text{et } e+i(i(x)) = x+e$$

■

On voit que si $x = i(i(x))$, alors le théorème est trivial. Or, cette propriété est vraie pour les groupes, mais pas pour les systèmes gauche-droite. D'autre part, on peut remarquer que la validité de l'un des deux lemmes aurait rendu l'autre immédiat. Notons au passage que si les lemmes générés sont proches du théorème à prouver, c'est simplement parce que le théorème fait presque intervenir des formes normales du système complété, puisque dans le système canonique des groupes, figurent les axiomes $x+e \to x$ et $e+x \to x$. On pourrait donc difficilement avoir des lemmes plus simples. D'autre part, le fait que les lemmes générés ne sont pas valides, n'implique en aucun cas que la formule objet n'est pas valide. Ils constituent des conditions suffisantes, mais pas forcément nécessaires. Cet exemple nous a servi à montrer comment utiliser l'analogie pour des preuves équationnelles, mais également à observer des cas d'échecs de l'analogie lorsque la formule objet n'est pas valide. Regardons maintenant, les cas où la formule objet est un théorème.

Nous allons traiter un exemple issu de [KW94]. Le système construit par Kolbe et Walther est spécialisé dans les preuves par induction. Il consiste grossièrement à abstraire la preuve comme nous le faisons pour la formule de référence, puis à l'instancier. Lorsque l'instanciation mène à des formules qui ne sont pas valides, ils procèdent à une analyse plus fine de la preuve. C'est le cas pour l'exemple qui suit. Pour notre part, nous allons voir que notre méthode d'analogie débouche sur un lemme trivial. Nous ne modifions que la forme de la preuve en la présentant comme une réécriture des termes au lieu d'écrire des égalités successives.

Exemple 9 Pour le théorème de référence, nous possédons le système équationnel suivant :

$$E_1 = \begin{array}{lrcl} (1) & plus(0,y) & = & y \\ (2) & plus(s(x),y) & = & s(plus(x,y)) \\ (3) & sum(empty) & = & 0 \\ (4) & sum(add(n,x)) & = & plus(n,sum(x)) \\ (5) & app(empty,y) & = & y \\ (6) & app(add(n,x),y) & = & add(n,app(x,y)) \\ (7) & plus(plus(n,x),y) & = & plus(n,plus(x,y)) \end{array}$$

Le théorème de référence est

$$\varphi \ : \ E_1 \vdash plus(sum(x),sum(y)) = sum(app(x,y))$$

La preuve se découpe en deux parties : le cas de base, et le cas inductif.
Cas de base :

$$\begin{array}{ccc} plus(sum(empty),sum(y)) & = & sum(app(empty,y)) \\ \downarrow (3) & & \downarrow (5) \\ plus(0,sum(y)) & & sum(y) \\ \downarrow (1) & & \\ sum(y) & & \end{array}$$

et le cas inductif :

$$\begin{array}{ccc} plus(sum(add(n,x)),sum(y)) & = & sum(app((add(n,x),y)) \\ \downarrow (4) & & \downarrow (6) \\ plus(plus(n,sum(x)),sum(y)) & & sum(add(n,app(x,y))) \\ \downarrow (7) & & \downarrow (4) \\ plus(n,plus(sum(x),sum(y))) & & plus(n,sum(app(x,y))) \\ & & \downarrow (HI) \\ & & plus(n,plus(sum(x),sum(y))) \end{array}$$

où (HI) est l'application de l'hypothèse d'induction. La formule à prouver par analogie fait intervenir la fonction *len* au lieu de *plus*. On peut donc écrire les choses ainsi :

$$E_2 = \begin{array}{lrcl} (1') & plus(0,y) & = & y \\ (2') & plus(s(x),y) & = & s(plus(x,y)) \\ (3') & len(empty) & = & 0 \\ (4') & len(add(n,x)) & = & s(len(x)) \\ (5') & app(empty,y) & = & y \\ (6') & app(add(n,x),y) & = & add(n,app(x,y)) \\ (7') & plus(plus(n,x),y) & = & plus(n,plus(x,y)) \end{array}$$

où (3') et (4') est la définition donnée pour *len*. Le théorème à prouver est :

$$\psi \ : \ E_2 \vdash plus(len(x),len(y)) = len(app(x,y))$$

Le filtrage de différence entre φ et ψ donne :

Correspondance	Différence
$sum \mapsto \lambda x.len(x)$	$\lambda nx.plus(n,len(x)) \not\stackrel{?}{=} \lambda nx.s(len(x))$

Cette différence est due aux axiomes (4) et (4'). Le premier réflexe doit être de tenter l'analogie horizontale (l'analogie immédiate est en échec). Mais le lemme $\lambda nx.plus(n,len(x)) \longrightarrow^* \lambda nx.s(len(x))$ n'est pas valide. L'analogie incomplète donne, pour le cas de base :

$$\begin{array}{ccc} plus(len(empty),len(y)) & = & len(app(empty,y)) \\ \downarrow (3') & & \downarrow (5') \\ plus(0,len(y)) & & len(y) \\ \downarrow (1') & & \\ len(y) & & \end{array}$$

qui est valide. Pour le cas inductif, on obtient :

et le cas inductif :

$$\begin{array}{ccc} plus(len(add(n,x)),len(y)) & = & len(app((add(n,x),y)) \\ \downarrow (4') & & \downarrow (6') \\ plus(s(len(x)),len(y)) & & len(add(n,app(x,y))) \\ \downarrow (7) & & \downarrow (4') \\ X & & s(len(app(x,y))) \\ & & \downarrow (HI') \\ & & s(plus(len(x),len(y))) \end{array}$$

Remarquons en passant la chose suivante : pour un étape $A \rightarrow^{(n)} B$ de la preuve de référence, si l'on applique la différence à B, alors le règle $A' \rightarrow^{(n')} B$ s'applique. D'autre part, si l'on applique la règle (n') à A, on obtient B'. Les deux façons de faire mènent au même résultat. Mais dans les deux cas, il y a échec, à l'étape étiquetée par X. Le lemme généré va être l'état de l'égalité à ce moment, c'est-à-dire :

$$plus(s(len(x)),len(y)) = s(plus(len(x),len(y)))$$

Or, ce lemme est trivial, puisque du membre gauche, on obtient directement le membre droit par $(2')$. Mais cela implique tout de même que cette analogie n'est pas automatique. Mais nous comparons là un processus général avec un système spécialisé dans ce type de preuves. ■

Exemple 10 Reprenons maintenant l'exemple qui nous a servi pour l'analogie par saut – relatif aux limites –, et essayons l'analogie dans l'autre sens. C'est-à-dire si nous essayons de montrer

$$\lim_{x\to\infty} f(x) = l \ \wedge \ \lim_{x\to\infty} g(x) = l' \Rightarrow \lim_{x\to\infty} f.g(x) = l.l'$$

à partir de la preuve de

$$\lim_{x\to\infty} f(x) = l \ \wedge \ \lim_{x\to\infty} g(x) = l' \Rightarrow \lim_{x\to\infty} f + g(x) = l + l'$$

Dans ce cas, le lemme généré sera :

$$\begin{array}{l} \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |(f(x) - l).(g(x) - l')| < \varepsilon.\varepsilon \\ \quad \Rightarrow \forall\varepsilon \ \ \exists n \ \ \forall x \ \ x < n \vee \ |f.g(x) - l.l'| < \varepsilon \end{array}$$

■

Intuitivement, la preuve de ce lemme constitue effectivement ce qu'il faut rajouter à la preuve pour la somme, pour obtenir la preuve pour le produit. La réutilisation de la preuve de référence ne peut aller plus loin sans intervention, ou utilisation d'un nouveau processus d'analogie pour ce lemme. L'analogie "pure" ne peut résoudre ce problème. Nous avons tenu à séparer clairement ce qui est réutilisation de la preuve du reste, afin de comprendre ce qui peut être vraiment attribué à l'analogie. Bien sûr, le lemme généré peut être prouvé par analogie, mais c'est un processus indépendant, qui utilisera une nouvelle preuve de référence. Enfin, reprenons l'exemple que nous avons introduit pour l'analogie horizontale.

Exemple 11 Dans [BdlTC87], la preuve complète, donnée pour le théorème de référence est la suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p(a)\vdash p(a)\quad \dfrac{q(a)\vdash q(a)}{q(a)\vdash \exists x\ q(x)}\exists R}{p(a),p(a)\Rightarrow q(a)\vdash \exists x\ q(x)}\Rightarrow L}{p(a),\forall x\ (p(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\forall L \quad \dfrac{\dfrac{q(b)\vdash q(b)}{q(b)\vdash \exists x\ q(x)}\exists R}{q(b),\forall x\ (p(x)\Rightarrow q(x))\vdash \exists x\ q(x)}ThinL}{(p(a)\vee q(b)),\forall x\ (p(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\vee L}{(p(a)\vee q(b))\wedge \forall x\ (p(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\wedge L}{\vdash (p(a)\vee q(b))\wedge \forall x\ (p(x)\Rightarrow q(x))\Rightarrow \exists x\ q(x)}\Rightarrow R$$

Rappelons que le filtrage de différence entre les deux formules donne

Correspondance	Différence
$p\mapsto \lambda x.p(x)$	$\forall x\ (p(x)\Rightarrow q(x))$
$q\mapsto \lambda x.q(x)$	$\not\equiv$
$a\mapsto a$	$\forall x\ (p(x)\Rightarrow r(x))\wedge \forall x\ (r(x)\Rightarrow q(x))$
$b\mapsto b$	

Par application de ce résultat à la preuve, nous obtenons :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{p(a)\vdash p(a)\quad \dfrac{q(a)\vdash q(a)}{q(a)\vdash \exists x\ q(x)}\exists R}{p(a),p(a)\Rightarrow r(a)\wedge \forall x\ (r(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\Rightarrow L^*}{p(a),\forall x\ (p(x)\Rightarrow r(x))\wedge \forall x\ (r(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\forall L \quad \dfrac{\dfrac{q(b)\vdash q(b)}{q(b)\vdash \exists x\ q(x)}\exists R}{q(b),\forall x\ (p(x)\Rightarrow r(x))\wedge \forall x\ (r(x)\Rightarrow q(x))\vdash \exists x\ q(x)}ThinL}{(p(a)\vee q(b)),\forall x\ (p(x)\Rightarrow r(x))\wedge \forall x\ (r(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\vee L}{(p(a)\vee q(b))\wedge \forall x\ (p(x)\Rightarrow r(x))\wedge \forall x\ (r(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\wedge L}{\vdash (p(a)\vee q(b))\wedge \forall x\ (p(x)\Rightarrow r(x))\wedge \forall x\ (r(x)\Rightarrow q(x))\Rightarrow \exists x\ q(x)}\Rightarrow R$$

La seule inférence qui n'est pas valide est

$$\dfrac{p(a)\vdash p(a)\quad q(a)\vdash \exists x\ q(x)}{p(a),p(a)\Rightarrow r(a)\wedge \forall x\ (r(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\Rightarrow L^*$$

Or, cette déduction s'obtient aisément par :

$$\dfrac{\dfrac{\dfrac{\dfrac{p(a)\vdash p(a)\quad \dfrac{r(a)\vdash r(a)\quad q(a)\vdash \exists x\ q(x)}{r(a),(r(a)\Rightarrow q(a))\vdash \exists x\ q(x)}\Rightarrow L}{p(a),p(a)\Rightarrow r(a),(r(a)\Rightarrow q(a))\vdash \exists x\ q(x)}\Rightarrow L}{p(a),p(a)\Rightarrow r(a),\forall x\ (r(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\forall L}{p(a),p(a)\Rightarrow r(a)\wedge \forall x\ (r(x)\Rightarrow q(x))\vdash \exists x\ q(x)}\wedge L$$

que l'on peut insérer à l'endroit de la déduction qui n'était pas valide, pour obtenir une preuve complète de B. ∎

Nous pouvons regarder la différence, avec ce qui est fait dans [BdlTC87]. Notre méthode est automatique jusqu'à la génération du lemme que nous devons prouver "à la main", ou par tout autre processus. Au contraire, dans [BdlTC87], la partie non automatique consiste à positionner des variables dans les termes de preuves aux endroits où les preuves doivent être différentes. Puis, par un processus de filtrage, ils obtiennent une preuve pour un lemme tout à fait comparable à celui que nous avons généré. Ceci laisse à penser qu'une combinaison des deux méthodes pourrait mener à un processus complètement automatique, au moins dans ce cas.

V.7 Synthèse et comparaison avec les travaux existants

Nous allons à présent donner une vue globale des processus d'analogie que nous avons proposés, et les comparer avec les travaux existants. De façon synthétique, les méthodes d'analogie présentées peuvent être essayées séquentiellement. C'est-à-dire que l'on peut tout d'abord essayer les processus complets et, en cas d'échec, tenter les méthodes plus générales. La séquence est représentée par la figure V.6. Nous mentionnons sur cette figure les méthodes qui ont été présentées spécifiquement pour les preuves par expansion ou les formules en forme normale négative, et qui nécessiteraient une adaptation afin d'être utilisées dans d'autres formalismes.

Chapitre	Type d'analogie ou transformation	Nécessite une adaptation	
I	Elagage de la preuve de référence $\rightarrow$ théorème plus simple	X	A
III	Mise en forme normale anti-Prénexe	X	B
I	Preuve de toute formule plus grande par la relation $\preceq$ preuves réduites $\leftrightarrow$ preuves par expansion	X	C
IV	Filtrage de différence		D
III	Différence vide : -correspondance -propriétés des quantificateurs -propriétés AC des connecteurs	(X)	E
III	Différence = $(a(\overline{t_n}) \stackrel{?}{\neq} \vee_{i=1}^{k} a(\overline{t_n^i}))$	X	F
V	Différence non vide -Analogie immédiate -Analogie par saut -Analogies horizontales -Analogie incomplète		G

Figure V.6: Séquence des méthodes d'analogie

Cette vue synoptique va nous aider à situer tout ceci parmi les travaux existants. Passons en revue les différentes parties (symbolisées par une lettre dans la colonne de droite de la figure V.6). Il semble que les seuls travaux comparables à la partie A sont ceux de D. Plaisted [Pla81]. Par abstraction, Plaisted peut obtenir un ensemble de clauses plus simple que l'ensemble de départ, par exemple en transformant toutes les occurrences d'un littéral ou d'une sous-formule par une même variable propositionnelle. Mais il ne s'agit pas d'éliminer les parties inutiles d'un théorème comme les preuves par expansion nous ont permis de le faire.

Pour la partie B, des transformations similaires à la mise en forme anti-prénexe ont déjà été utilisées dans [Bib74, BdlTCC88, Egl94]. Pour" antiprénexe", on trouve également la dénomination "miniscoping", puisqu'il s'agit d'optimiser le champ d'application des quantificateurs avant de skolémiser les formules, afin de faciliter la résolution. Le but poursuivi n'est donc pas le même, puisque le nôtre est d'obtenir une forme normale unique pour une formule donnée. L'intérêt étant de définir une façon de passer d'une preuve d'une formule à une preuve de toute formule ayant la même forme normale.

En ce qui concerne la partie C, le fait de pouvoir prouver une formule dans laquelle la formule de référence est enchâssée n'a pas, jusqu'à présent, suscité grand intérêt. Il semble pourtant que ce soit une bonne base pour l'analogie, puisqu'alliée à l'élagage, cette transformation permet d'une part de poser le problème de la pertinence d'un théorème de référence, et d'autre part, de prouver toutes les formules dans lesquelles elle est enchâssée. L'intégration de ce processus dans un prouveur peut être un atout non négligeable et lui éviter bien des preuves redondantes.

Pour le filtrage de différence – partie D –, l'idée apparaît dans [BdlTC88]. Le filtrage d'ordre deux est maintenant assez répandu dans les processus d'analogie, et l'idée d'aller plus loin, en utilisant les échecs du filtrage devrait s'élargir, car elle constitue un outil naturel pour définir une différence. Pour notre part, nous en avons fait la base de toute les méthodes d'analogie. D'autre part, nous nous sommes attachés à produire des méthodes aussi générales que possible, mais il serait très intéressant de voir si ce que peuvent donner ces méthodes dans le cadre de l'isomorphisme de Curry-Howard utilisé par Boy de la Tour, Caferra et Kreitz. Ce formalisme est séduisant pour l'analogie, puisque formules et preuves y sont intimement liées. La théorie des types est également utilisée, mais de façon plus intensive dans les travaux de E. Gunter [Gun93] pour la généralisation de preuves.

Pour la partie E, à notre connaissance, seule la correspondance a déjà été évoquée pour l'analogie. En commençant par Plaisted, qui a défini le renommage – un sous ensemble de la correspondance – comme une abstraction possible. Par ailleurs, en utilisant Curry-Howard, le nom des symboles n'a pas d'importance, puisque l'on s'attache plus particulièrement à leur type. Ensuite, Kolbe et Walther, dans [KW94], abstraient leurs preuves de façon semblable à ce que nous faisons pour la formule de référence. Ainsi, le schéma obtenu peut être instancié par des symboles quelconques. Enfin, on peut trouver dans [Owe90], des méthodes basées sur des heuristiques pour construire des correspondances entre symboles. Ces méthodes sont utilisées dans le cadre d'"analogie de haut niveau", c'est-à-dire pour la planification de preuves et de raisonnement, plutôt que pour la réutilisation automatique de preuves. Quant à ce qui se rapporte aux propriétés des quantificateurs et des connecteurs, il semble que rien n'y fasse référence dans le cadre de l'analogie.

Dans la partie F, il s'agit d'approcher les limites de l'analogie automatique, et de montrer que la notion de différence s'impose comme une notion fondamentale pour l'analogie. Dans [FH94], Felty et Howe donnent une méthode de généralisation de de tactiques de preuve qui peuvent être réutilisées. Un des exemples donnés consiste à ré-appliquer une tactique pour une formule dont on a substitué un symbole de constante par un autre. C'est un exemple que nous pouvons traiter de façon complète, comme nous l'avons montré au chapitre IV si le théorème de référence

n'est pas une disjonction de deux théorèmes, mais Felty et Howe se placent dans un contexte beaucoup plus puissant, puisqu'ils utilisent l'ordre supérieur autorisé par λ-Prolog. Nous pouvons dès à présent remarquer qu'un certain nombre de notions n'avaient pas attiré l'attention jusqu'à maintenant. C'est sans doute parce que des similitudes qui semblaient trop simples ont été délaissées pour des cas plus complexes. Excepté pour Plaisted, qui s'est penché sur des processus essentiellement complets. De la même façon, notre axe de conduite a été de produire des outils de base pour l'analogie. Ceux-ci doivent être automatiques, autant que possible. Il peut, par exemple, être problématique de mettre sur pieds des méthodes d'analogie très complexes s'il n'est pas possible de voir que deux formules sont équivalentes modulo la position de leurs quantificateurs, et de transformer la preuve de l'une en preuve de l'autre. Le revers est que pour montrer la complétude et la terminaison de ces processus de base, nous avons dû nous placer dans le cadre d'une représentation particulière de preuves.

La partie G étant la partie générale, la comparaison peut s'étendre à de nombreux travaux. Tout d'abord, une différence assez marquée avec le qui existe est que nous nous sommes efforcés d'appliquer les concepts exposés à des domaines assez différents. Mais cela est dû aussi au fait qu'ils ne sont pas illustrés par une implantation, qui spécialiserait immanquablement les applications. La seconde originalité est que l'origine des processus d'analogie que nous avons présentés se trouve toujours dans les formules. La clef de voûte reste la différence entre ces formules. Il est en réalité très difficile de comparer les différentes méthodes d'analogie, puisque chacune d'entre elles s'applique à un cadre précis, ou est spécialisée dans un type de preuve particulier, et comporte des étapes non automatiques. Prenons par exemple ce qui est fait par Brock et al. dans [BCP88]. Nous avons réutilisé leur exemple principal pour illustrer l'analogie par saut. Dans les deux cas, l'analogie peut être automatisée. Mais lorsque nous essayons de montrer la propriété des limites pour le produit à partir de cette propriété pour le produit, nous arrivons inévitablement sur un lemme. Pour leur part, ils affirment pouvoir construire la preuve. Mais ce n'est plus là le fruit de l'analogie, puisqu'ils font intervenir un prouveur classique qui, de plus, est spécialisé pour ce domaine. De même, si l'on regarde l'exemple de Kolbe et Walther que nous avons évoqué, nous aboutissons sur un lemme. L'utilisation d'un système de preuve nous permettrait de prouver facilement, puisqu'il suffit d'appliquer une seule règle, qui est la seule à pouvoir s'appliquer. Mais nous nous sommes volontairement limités à l'analogie "pure", afin de mieux pouvoir en cerner les possibilités et les limites.

V.8 Conclusion sur l'analogie générale

Tous les exemples que nous avons développés dans ce chapitre montrent en premier lieu que le filtrage de différence s'impose comme un outil de base pour l'analogie. Dans chacun des cas évoqués, c'est l'information délivrée par la différence qui détermine le type d'action à mettre en œuvre. Si nous nous sommes attachés à donner des mécanismes spécifiques aux preuves par expansion dans un premier temps, afin de traiter un maximum de cas de façon automatique, il s'avère que le filtrage de différence peut être utilisé tel quel dans un grand nombre de formalismes. Ce qui signifie également que l'approche consistant à analyser les formules avant de passer aux preuves est intéressante.

Ensuite, ces exemples donnent un éventail de possibilité pour traiter l'analogie. De l'analogie immédiate qui est automatique, à l'analogie incomplète qui peut dans certain cas, générer le théorème de départ comme lemme suffisant. Il n'y a pas une façon d'appliquer l'analogie; celle-ci

doit nécessairement tenir compte du cas de figure. Néanmoins, dans un cadre général, l'analogie est une méthode incertaine.

Conclusion

L'objet de cette thèse n'est pas de modéliser le raisonnement par analogie, mais simplement de fournir des outils utilisables pour la preuve par analogie. Les techniques basées sur l'analogie sont appelées à se développer pour aider les systèmes de déduction automatique. Nous avons donc décidé d'aborder ce sujet par ses bases, en construisant une collection de méthodes adaptées à des types d'analogies définis formellement. Ces méthodes se veulent une base de travail pour progresser dans le domaine de l'analogie.

Du point de vue des apports, nous pouvons distinguer trois composantes.

- Nous avons défini, à travers un nouvel algorithme de filtrage AC d'ordre deux, une base qui nous semble nécessaire pour les processus d'analogie. L'adaptation de cet algorithme à la recherche de similitude et/ou de différence montre bien qu'il permet de répondre – en partie, évidemment – à la question "quelle est la différence entre ces deux formules?". Il constitue en quelque sorte le dénominateur commun des méthodes énoncées par la suite. De plus, nous nous sommes attachés à le rendre efficace en réduisant considérablement le nombre de solutions calculées, tout en préservant la complétude. Sans considérer le fait que cet algorithme peut trouver des applications dans d'autres domaines que l'analogie, nous pouvons penser à intégrer d'autres critères de similitude, comme celui relatif aux quantificateurs. Signalons également que la version naïve de l'algorithme a été implémentée. La version efficace, quant à elle est en cours d'implémentation. A partir de là, la calcul de la différence devrait suivre aisément, pour constituer une base d'implémentation.

- Nous avons également proposé un certain nombre de méthodes complètes et automatiques s'appliquant à des cas élémentaires d'analogie. D'une part, directement par l'utilisation du filtrage AC, la reconnaissance des similitudes est aisée. Les méthodes de transformation de preuves qui l'accompagnent permettent d'appliquer l'analogie de façon assez satisfaisante.

  Par ailleurs, de façon plus faible, nous considérons d'abord un critère de similitude d'ordre propositionnel. Celui-ci permet, dans un premier temps, de transformer la preuve d'un théorème en une preuve d'un théorème plus simple, et ce, grâce aux connexions des preuves par expansion. Ensuite, nous avons montré que nous sommes capables, de transformer cette preuve en preuve de toute formule plus grande par la relation $\preceq$ qui est approximativement une relation d'enchâssement.

  Par la mise en forme anti-prénexe – dont l'intérêt est d'être unique pour une formule donnée – nous avons défini une transformation de preuve capable de passer de la preuve d'une formule à la preuve de toute formule possédant la même forme anti-prénexe – ou

ayant une forme prénexe en commun avec elle –. Cette prise en compte des propriétés de quantificateurs est très importante, car elle permet de considérer comme similaires des formulations différentes d'un même théorème. Ce qui n'est pas évident a priori.

Toujours dans le cadre des méthodes automatiques et complètes, à l'aide du calcul de la différence, nous avons défini une transformation permettant de résoudre le problème d'une différence élémentaire. La complétude signifie alors que nous sommes capables de fournir une preuve de la formule visée si et seulement si celle-ci est valide. Toutefois, il existe une restriction sur le théorème utilisé comme référence.

- Le troisième point constitue une étude plus prospective de ce qui peut être réalisé dans le cadre de l'analogie, utilisant les outils que nous avons construits. Nous montrons comment, dans un cadre général, il est possible de classer les cas de figure de l'analogie et y apporter des éléments de réponses. Ces types d'analogies sont traités par des méthodes allant d'un processus automatique pour les cas simples, à la génération de lemmes suffisants. Ces lemmes peuvent d'ailleurs ne pas être valides, puisque rien n'indique que la formule visée est un théorème.

Quant aux perspectives, de par la puissance potentielle de l'analogie, elles sont nombreuses. Un de nos buts est de poursuivre l'étude des preuves par expansion en étendant les méthodes décrites à l'ordre supérieur. C'est une suite logique de ces travaux, puisque l'une de nos motivations, pour utiliser les preuves par expansion est justement qu'elles supportent très bien le passage à l'ordre supérieur. Par conséquent, un certain nombre de mécanismes doivent s'y appliquer. Mais nous souhaitons également étudier d'autres systèmes de représentation de preuves. Les principes d'analogie sont généraux et doivent pouvoir trouver une application dans tout type de représentation. Par exemple, l'isomorphisme de Curry-Howard, utilisé par Boy de la Tour et al. semble être un formalisme très intéressant pour l'analogie, puisqu'il lie étroitement la formule et sa preuve. L'un des principaux problèmes, réside dans le fait que les connexions dont nous avons largement tiré parti, semblent difficiles à adapter dans ce formalisme.

Par ailleurs, l'implantation des méthodes fournies, à partir de l'implantation de l'algorithme de filtrage, fournira une vérification pratique de l'apport de l'analogie dans le cadre de la déduction automatique. Il est évident que cette validation par la pratique est nécessaire. D'autre part, intégrer au filtrage les manipulations de formules nécessaires à la reconnaissance de similitudes relatives aux quantificateurs pourrait fournir un outil très puissant. Mais il n'est pas sûr que le problème soit décidable. Ensuite, les critères d'analogie que nous avons utilisés sont loin d'être exhaustifs. Il en existe certainement bien d'autres qui méritent d'être étudiés, par exemple des critères de nature plus sémantique. La notion d'analogie est suffisamment riche pour dégager bien des façons de l'exploiter. Enfin, et peut-être à plus long terme, il faut envisager de constituer une base de théorèmes et de preuves afin de les utiliser pour l'analogie. Nous avons eu l'occasion de montrer que la forme du théorème utilisé comme référence, et même de sa preuve peuvent avoir une importance déterminante. Il s'agit certainement de problèmes à prendre en compte pour la constitution d'une telle base. Nous pouvons aussi nous demander comment choisir, dans une telle base, le théorème adéquat pour pratiquer l'analogie avec une conjecture donnée. Nous pourrions alors envisager d'appliquer des techniques d'analogie à de véritables systèmes de déduction, et montrer son intérêt en pratique.

Liste des Figures

I.1 Notre vue sur la preuve par analogie 14
I.2 Transformations réalisées dans le cadre de l'analogie simple 16
I.3 Arbres d'expansion 25
I.4 Exemple de preuve par expansion 28
I.5 Elagage des preuves par expansion 34
I.6 Exemple d'élagage d'une preuve par expansion en une preuve réduite 34
I.7 Transformation preuve par expansion réduite $\rightarrow$ preuve par expansion 36
I.8 Transformation d'une preuve réduite en une preuve par expansion pour une formule plus grande par la relation $\preceq$ 37

II.1 Combinaison naïve du filtrage d'ordre deux et d'une théorie 55

III.1 Règles de mise en forme anti-prénexe 80
III.2 Remontée d'un quantificateur existentiel pour la mise en forme prénexe d'une preuve 87
III.3 Transformation de preuves : anti-prénexe $\rightarrow$ prénexe 89

IV.1 Preuve de référence pour une analogie utilisant la différence 108
IV.2 Arbre d'expansion obtenu par explosion 109
IV.3 Preuve obtenue par analogie à partir de la différence 109

V.1 Analogie dans un cadre général 120
V.2 Analogie immédiate 121
V.3 Analogie par saut 125
V.4 Analogies horizontales 129
V.5 Analogie incomplète 133
V.6 Synthèse des transformations et méthodes d'analogie proposées 139

Bibliographie

[ACM82] Andrews (P. B.), Cohen (E. L.) et Miller (D. A.). – *A look at TPS*, pp. 50–69. – Springer-Verlag, 1982, *Lecture Notes in Computer Science*, volume 138.

[And76] Andrews (P. B.). – Refutations by Matings. *IEEE Transactions and Computers*, vol. C-25, n° 8, 1976, pp. 801–807.

[And79] Andrews (P. B.). – General Matings. *In: Proceedings 4th Workshop on Automated Deduction, Austin (Tex., USA)*, pp. 19–25. – 1979.

[And86] Andrews (P. B.). – On connections and higher-order logic. *In: Proceedings 8th International Conference on Automated Deduction, Oxford (UK).* – 1986.

[And87] Andrews (P. B.). – Transforming Mmatings into Natural Deduction Proofs. *In: Proceedings 5th International Conference on Automated Deduction, Les Arcs (France)*, pp. 281–292. – 1987.

[BCP88] Brock (B.), Cooper (S.) et Pierce (W.). – Analogical Reasoning and Proof Discovery. *In: Proceedings 9th International Conference on Automated Deduction, Argonne (Ill., USA).* pp. 454–468. – Springer-Verlag, 1988.

[BdlTC87] Boy de la Tour (T.) et Caferra (R.). – Proof analogy in interactive theorem proving: A method to express and use it via second order pattern matching. *In: proceedings of AAAI'87.* – 1987.

[BdlTC88] Boy de la Tour (T.) et Caferra (R.). – A formale approach to some usually informal techniques used in mathematical reasoning. *In: ISSAC'88.* pp. 402–406. – Lecture Notes in Computer Science, 1988.

[BdlTCC88] Boy de la Tour (Thierry), Caferra (Ricardo) et Chaminade (Gilles). – Some tools for an inference laboratory (atinf). *In: Proccedings of the 9th International Conference on Automated Deduction*, éd. par Lusk (E.) et Overbeek (R.). pp. 744–745. – Springer Lecture Notes in Computer Science 310, 1988.

[BdlTK92] Boy de la Tour (T.) et Kreitz (C.). – Building proofs by analogy via the curry-howard isomorphism. *In: Proceedings of the 1st International Conference on Logic Programming and Automated Reasoning, St. Petersburg (Russia).* pp. 202–213. – Springer-Verlag, 1992.

[Bib74] Bibel (W.). – An approach to a systematic theorem proving procedure in first-order logic. *Computing*, vol. 12, 1974, pp. 43–55.

[Bib79] Bibel (W.). – Tautology testing with a generalized matrix reduction method. *Theoretical Computer Science*, vol. 8, 1979, pp. 31–44.

[Bib81] Bibel (W.). – On matrices with connections. *Journal of the ACM*, vol. 28, 1981, pp. 633–645.

[BKN87] Benanav (D.), Kapur (D.) et Narendran (P.). – Complexity of matching problems. *Journal of Symbolic Computation*, vol. 3, n° 1 & 2, avril 1987, pp. 203–216.

[BSW90] Bundy (A.), Smaill (A.) et Wiggins (G.). – *The synthesis of Logic Programs from Inductive Proofs.* – Springer Verlag, 1990. J. Lloyd ed., Computationnal Logic, pp 135–149.

[BT88] Breazu-Tannen (V.). – Combining algebra and higher-order types. *In : Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, pp. 82–90. – 1988.

[Bun91] Bundy (A.). – A Science of Reasoning. *In : Computational logic : essays in honor of Alan Robinson*, éd. par Press (MIT), chap. 6, pp. 178–198. – Lassez, J. L. and Plotkin, G. Ed., 1991.

[BW92] Basin (D.) et Walsh (T.). – Difference Matching. *In : Proceedings 11th International Conference on Automated Deduction, Saratoga Springs (N.Y., USA).* – 1992.

[Cur93] Curien (R.). – Second order E-matching as a tool for automated theorem proving. *In : Proceedings of EPIA'93 (Portugese Conference on Artificial Intelligence) Porto, Portugal.* – 1993.

[Cur92] Curien (R.). – *Combining Second Order Matching and First Order E-Matching.* – Rapport technique n° 92-R-178, CRIN, 92.

[Dav63] Davis (M.). – Eliminating the irrelevant from mathematical proofs. *In : Experimental arithmetic, High Speed Computing and Mathematics, Proceeding of Symposia in Applied Mathematics XV.* American Mathematical Society, pp. 15–30. – 1963.

[Dow94] Dowek (G.). – Third order matching is decidable. *Annals of Pure and Applied Logic*, vol. 69, 1994, pp. 135–155.

[Egl94] Egly (U.). – On the value of antiprenexing. *In : Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning, LPAR'94*, éd. par Pfenning (F.), pp. 69–83. – 1994.

[FH94] Felty (A.) et Howe (D.). – Generalization and reuse of tactic proofs. *In : Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning, LPAR'94*, éd. par Pfenning (F.), pp. 1–15. – 1994.

[Gol81] Goldfarb (D.). – The undecidability of the second order unification problem. *Theoretical Computer Science*, vol. 13, 1981, pp. 225–230.

[Gun93] Gunter (E.). – A calculus for proofs and proof generalization. – 1993. Unpublished.

[HL78] Huet (G.) et Lang (B.). – Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, vol. 11, 1978, pp. 31–55.

[HS86] Hindley (J. Roger) et Seldin (Johnathan P.). – *Introduction to Combinators and Lambda-calculus.* – Cambridge University, 1986.

[Hue76] Huet (G.). – *Résolution d'equations dans les langages d'ordre 1,2, ...,ω.* – Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.

[Kli71] Kling (R. E.). – A Paradigm for Reasinong by Analogy. *Artificial Intelligence*, vol. 2, 1971, pp. 147–178.

[KW94] Kolbe (T.) et Walther (C.). – Reusing proofs. *In: ECAI'94, 11th European Conference on Artificial Intelligence*, éd. par Cohn (A.), pp. 80 – 84. – 1994.

[Mel93a] Melis (E.). – *Analogies between proofs - a case study.* – Rapport technique, Universität des Saarlandes, 1993.

[Mel93b] Melis (E.). – *Change of Representation in Theorem Proving by Analogy.* – Rapport technique n° SR-93-07, Universität des Saarlandes, 1993.

[Mil87] Miller (D.). – A compact representation of proofs. *Studia Logica*, vol. XLVI, n° 4, 1987, pp. 347–370.

[MW94] Müller (O.) et Weber (F.). – Theory and practice of modular higher-order E-unification. *In: Proceedings 12th International Conference on Automated Deduction, Nancy (France)*, éd. par Springer-Verlag, pp. 650–664. – 1994.

[Nip91] Nipkow (T.). – Combining matching algorithms: The regular case. *Journal of Symbolic Computation*, 1991, pp. 633–653.

[NQ91] Nipkow (T.) et Qian (Z.). – Modular higher-order E-unification. *In: Proceedings 4th Conference on Rewriting Techniques and Applications, Como (Italy).* pp. 200–214. – Springer-Verlag, 1991.

[Owe90] Owen (S.). – *Analogy for Automated reasoning.* – Academic Press, 1990, *Perspectives in Artificial Intelligence.*

[Pfe84] Pfenning (F.). – Analytic and Non-analytic Proofs. *In: Proceedings 7th International Conference on Automated Deduction, Napa Valley (Calif., USA)*, pp. 394–413. – 1984.

[Pfe87] Pfenning (F.). – *Proof Transformation in Higher-Order Logic.* – Pittsburgh, Pennsylvania, Thèse de PhD, Department of Mathematics, Carnegie-Mellon University, 1987.

[Pla81] Plaisted (D. A.). – Theorem Proving with Abstraction. *Artificial Intelligence*, vol. 16, 1981, pp. 47–108.

[Pre94] Prehofer (C.). – Decidable Higher-Horder Unification Problems. *In : Proceedings 12th International Conference on Automated Deduction, Nancy (France)*, pp. 635–649. – 1994.

[QW92] Qian (Z.) et Wang (K.). – Higher-order E-unification for arbitrary theories. *In : LOGIC PROGRAMMING : Proceedings of 1992 joint international conference and symposium on logic programming*, éd. par Apt (K.). – 1992.

[Sny88] Snyder (W.S.). – *Complete Sets of Transformations for General Unification.* – Thèse de PhD, University of Pennsylvania, 1988.

[Soc87] Socher (R.). – *Boolean Simplification of First-Order Formulae.* – SEKI-Report n° SR-87-04, Universität Kaiserslautern, 1987.

Index

$\curlywedge$cc 26
$\preceq$ 18
$\mathcal{FV}$ 47
$\mathcal{M}$ 25
$\mathcal{H}$ 47
$\models$ 18
$\mathcal{ESEM}$ 54
$\vdash$ 27
λ-abstraction 47
λ-calcul 47
α-conversion 47
η-expansion 48
η-réduction 48
β-réduction 47
λ-terme 46
    simplement typé 46
λ-terme 84
$<_Q$ 26
$\vdash_R$ 32

abstraction 54, 74, 139
analogie 13
    horizontale par la différence 130
    horizontale structurelle 130
    immédiate 120
    immédiate pour la synthèse de prog. 121
    incomplète 132
    par saut 125
    par saut pour l'analyse 126
    pour des preuves non-analytiques 114
    pour la déduction naturelle 114
    simple 16
    utilisant la différence 105
    via $\preceq$ 28
arbre
    d'expansion 24, 25
    d'expansion réduit 31

complétude 59, 72, 110
connecteurs 17, 77
connexion 21, 25, 32, 111
    AC-connexion 77
    minimale 26, 28
correction 40, 44, 58, 69
correspondance 74, 140
couverture 26
    minimale 26, 39

décomposition 51
déduction naturelle 94
différence 101, 141
    minime 105, 110

élagage 107, 113, 140
    de formule 29
    de preuve 29, 32
enchâssement 18
explosion 106

fcm 31
filtrage 50
    de différence 103
    équationnel 53
    syntaxique 49
forme normale
    β-forme normale 48
    $\beta\eta$-forme normale longue 48
    anti-prénexe 79, 83, 140
forme prénexe 79, 89
formule 17, 78, 84
    atomique 17
    formule commune minimale 30
    superficielle 24, 33

généralisation 140

génération de lemme 130, 133

heuristique 119, 140

imitation 51
isomorphisme de Curry-Howard 94, 140

L-coupure 131
lemme de différence 130
littéral 18

mating 26
miniscoping 74, 79, 140

nœud
    d'expansion 25, 87
    de sélection 25
    superflu 32

occurrence 28
ordre
    d'un λ-terme 47
    d'un type 46

position 64
    aplatie 64
    indépendante 64
preuve
    équationnelle 133
    par expansion 21, 26
    par expansion réduite 31
    par induction 122, 135
projection 51

quantificateurs 17, 78
    effacement des quantificateurs 85
    permutation des quantificateurs 84

relation
    $\preceq$ 18
    $<_Q$ 40
    de dépendance 26
renommage 27, 140

signature 46
similitude 18, 73, 91
    par AC-correspondance 77
    propositionnelle 18
    quantificateurs 78
simplification
    de formule 29, 32
    de preuve 32
skolémisation 26
substitution 47, 53
    normalisée 50
synthèse de programmes 121

tautologie 28
terme 78
    λ-terme 46
    aplati 64
    atomique 47
    coupure 65
    coupure élémentaire 65
    d'expansion 25
    flexible 51
    rigide 51
    sous-terme étranger maximal 54
terminaison 29, 37, 40, 60, 80, 103
théorème de référence 35, 110, 140
théorie 53, 60
    AC 53, 63
    régulière 53, 57
transformation
    E 29
    TR 35
type 46
    élémentaire 46
    individuel 46
    propositionnel 46

UBV 63
unification 50

variable
    de sélection 25
    expansée 25
    libre 47
    liée 63

Nom : **CURIEN**

Prénom : **Régis**

DOCTORAT de l'UNIVERSITE HENRI POINCARE, NANCY-I

en INFORMATIQUE

VU, APPROUVÉ ET PERMIS D'IMPRIMER

Nancy, le 30 JAN. 1995 n°47

Le Président de l'Université

UNIVERSITÉ HENRI POINCARÉ NANCY 1 LE PRESIDENT

J.P. FINANCE

3C-F9

UNIVERSITÉ HENRI POINCARÉ - NANCY 1
24-30, rue Lionnois B.P.3069 54013 Nancy cedex
Tél : 83 32 81 81 Fax : 83 32 95 90
Adresse électronique : _____ @presn1.u-nancy.fr

Résumé

Le but de cette thèse est de fournir des outils permettant à la déduction automatique de réutiliser les résultats déjà obtenus. En effet, la preuve par analogie consiste à construire de nouvelles preuves à partir de preuves existantes. Il faut dans un premier temps reconnaître que le problème à résoudre est semblable à un problème déjà résolu, puis, transformer la solution existante, pour obtenir une solution du nouveau problème. L'approche adoptée consiste à définir formellement des relations liant deux formules logiques du premier ordre – celle dont nous possédons une preuve est appelée référence – pour en déduire une méthode automatique de transformation de la preuve de référence en une preuve de la nouvelle formule. Le concept d'analogie est très puissant, mais aussi très intuitif. Ainsi, pour le formaliser, nous l'avons réduit à des concepts plus simples afin de les automatiser. Nous avons défini quatre relations liant les formules, que nous appelons similitudes. Ces similitudes considèrent les propriétés de la logique propositionnelle, les propritétés des quantificateurs, le renommage des fonctions et prédicats et les propriétés Associatives-Commutatives des connecteurs logiques.

Un outil fondamental pour la reconnaissance des ces similitudes est un algorithme de filtrage du second ordre modulo AC. La complétude et la terminaison de cet algorithme montrent la décidabilité du problème de filtrage AC du second ordre. Les transformations de preuves correspondant à ces similitudes sont données pour les preuves par expansion introduites par Miller et Pfenning. Cette représentation possède des propriétés très intéressantes pour l'analogie.

Pour dépasser le stade des similitudes, nous utilisons le calcul de différence, qui utilise les échecs du filtrage. Nous montrons que lorsque la différence entre les deux formules considérées est simple, nous pouvons espérer une méthode complète d'analogie. Enfin, nous montrons, dans le cas général, et à partir d'exemples, comment l'analogie peut être envisagée par l'étude de la différence.

Mots-clés : déduction automatique, analogie, réutilisation de preuves, filtrage, similitude, différence.

Abstract

This thesis aims at producing tools for automated deduction in order to reuse established results: proof by analogy consists in building new proofs from existing ones. We first have to recognize the similarity between an already solved problem and the current one. Then, we transform the existing solution to obtain a solution for the new problem. The aim of our approach is to formally define relations between first order logical formulas – the one for which we have a proof is called a reference – in order to deduce an automatic transformation of the reference proof into a proof of the new formula. The analogy concept is very powerful, but also very intuitive. Therefore, we have reduced it to simpler concepts to enable their automation. We have defined four relations on formulas that we call similarities. These relations take into account propositional logic properties, quantifier properties, function and predicate renaming, and the connectors' Associative-Commutative properties.

A fundamental tool for recognizing similarities between formulas is a second-order AC-matching algorithm. The completeness and termination of this algorithm show the decidability of the second-order AC-matching problem. Relative proof transformations are given for expansion-tree proofs, introduced by Miller and Pfenning: this representation has some relevant properties for analogy.

In order to go beyond the similarity stage, we introduce the difference calculus, which uses matching failures. We show that when the difference between formulas is simple, we may hope for a complete analogy process. Finally, we show, in a general setting and using examples, how analogy can be viewed by studying differences.

Keywords: automated deduction, analogy, reusing proofs, matching, similarity, difference.