Intégration et implémentation de mécanismes de déduction naturelle dans les démonstrateurs utilisant la résolution

Gilles Chaminade

T H E S E

présentée par

Gilles Chaminade

pour obtenir le titre de DOCTEUR

de l'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE
(Arrêté ministériel du 23 novembre 1988)

en INFORMATIQUE

Intégration et implémentation de mécanismes de Déduction Naturelle dans les démonstrateurs utilisant la Résolution

---

Date de soutenance : Mardi 1er octobre 1991

Composition du jury :

Ricardo	Caferra	directeur
Hubert	Comon	examinateur
Hans-Jürgen	Ohlbach	rapporteur
Michaël	Rusinowitch	rapporteur
Jean-Pierre	Verjus	président

Thèse préparée au sein du Laboratoire d'Informatique Fondamentale
et d'Intelligence Artificielle de l'IMAG

Remerciements

Je tiens à remercier Jean-Pierre Verjus qui a bien voulu me faire l'honneur d'être le président du jury.

Je tiens à remercier également Michaël Rusinowitch et Hans-Jürgen Ohlbach qui, en acceptant le rôle difficile de rapporteur, ont bien voulu me faire bénéficier de leur grande compétence scientifique. Je suis particulièrement sensible à l'honneur que me fait Hans-Jürgen Ohlbach en acceptant de lire cette thèse, rédigée dans une langue qui n'est pas la sienne.

Je suis très reconnaissant à Hubert Comon de l'intérêt qu'il a toujours porté à mon travail et le remercie vivement de bien vouloir participer à ce jury.

Je remercie également tous les membres du projet ATINF de leur soutient. Les résultats de la première partie de cette thèse ont été le fruit de nombreuses discussions avec Thierry Boy de la Tour dont la connaissance de toutes les subtilités du renommage m'a été précieuse. Je suis également reconnaissant à Nicolas Zabel de m'avoir fait profiter de sa grande culture scientifique: lecteur infatigable des premières versions de cette thèse, il a su – par des remarques toujours judicieuses – contribuer à améliorer ce document.

Je suis reconnaissant à Philippe Jorrand d'avoir fait du LIFIA un lieu de travail agréable dans lequel j'ai pu poursuivre mes travaux dans d'excellentes conditions scientifiques et techniques.

Je tiens à remercier R. Caferra pour les exceptionnelles qualités humaines et scientifiques qu'il a toujours su montrer au cours de ces années. Je lui suis reconnaissant de m'avoir fait profiter de sa lucidité et de sa grande connaissance de l'existant dans le domaine de la démonstration automatique ainsi que d'un soutient sans faille. Son amitié indéfectible m'a été précieuse.

Introduction

Le *principe de Résolution* [Rob65] est un des calculs des prédicats les plus employés en déduction automatique. Comme le titre de l'article dans lequel il a été introduit l'indique ("A Machine-oriented Logic Based on the Resolution Principle"), ce calcul a été conçu pour être mis en œuvre sur machine. Il se caractérise par sa simplicité et son uniformité:

- *simplicité des formules manipulées*: ce calcul n'est applicable qu'à des formules sous une forme normale: la forme clausale.
- *simplicité des règles d'inférence*: soit résolution [Rob65], soit, plus classiquement, résolution binaire et factorisation [CL73] [Lov78], celles-ci peuvent être implémentées efficacement.
- *utilisation de l'unification*: fondé sur le théorème d'Herbrand, la méthode de Résolution cherche à établir l'insatisfaisabilité d'un ensemble de clauses en montrant l'insatisfaisabilité d'un ensemble fini d'instances de ces clauses. Le choix de ces instances est un problème crucial. L'utilisation de l'unification permet de répondre de manière générale et incrémentale à ce problème.

Les avantages de l'utilisation d'une forme normale ne se réduisent pas uniquement à des soucis d'implémentation: la forme clausale facilite aussi l'étude des stratégies.

Malgré l'existence de nombreuses stratégies de Résolution – D. Loveland en recense plus de 25 dans [Lov78] dont la plus part ont été trouvées dans les années 1965-1970 – il est apparu très vite que le caractère uniforme et uniquement dirigé par la syntaxe de la Résolution rendait cette méthode inopérante pour de nombreuses classes de problèmes.

Dans un article célèbre "Non Resolution Theorem-Proving" [Ble77], W.W. Bledsoe défend la thèse selon laquelle il est nécessaire de mettre en œuvre des procédures de preuve dans lesquelles le processus inférentiel est assuré non pas par un unique mécanisme de déduction (comme dans le cas de la Résolution) mais par l'intégration de plusieurs mécanismes de déduction spécialisés permettant d'intégrer des connaissances sur des théories particulières autrement que par la donnée d'un ensemble d'axiomes. D'autre part, W.W. Bledsoe insiste sur le fait que ces procédures

doivent opérer d'une manière qui reflète la manière de procéder des mathématiciens et ce, afin de pouvoir mettre en œuvre des démonstrateurs de théorèmes interactifs auxquels il sera possible de fournir des heuristiques et des suggestions issues de l'expérience humaine.

Dans les vingt dernières années, de nombreux travaux ont porté sur l'intégration à la Résolution de procédures de raisonnement spécialisées. Le traitement de l'égalité a, par exemple, fait l'objet de nombreuses études : introduction de nouvelles règles d'inférence (démodulation [WR67], paramodulation [WRCS70], ...) intégration de règles de simplification [Sla74] utilisation de résultats issus des techniques de réécriture à la Résolution [Rus87] [Bac89] [BG90], l'utilisation d'algorithmes d'unification paramétrés par des théories équationnelles [Sie84] [JK90]... Les travaux visant à prendre en compte les ordres totaux ou partiels ainsi qu'une théorie naïve des ensembles [Sla72] ou encore des théories taxonomiques [Wal83] [SS86] [Coh87] et enfin l'utilisation de la Theory-Resolution [Sti85] sont d'autres exemples des efforts mener pour pallier l'uniformité de la Résolution.

Malgré ces améliorations, la Résolution souffre toujours d'un inconvénient majeur: l'utilisation d'une forme normale – et tout particulièrement de la forme clausale – conduit à manipuler des formules qui ne reflètent pas la structure et donc l'intention voulue ("intended interpretation") de la formule initiale. Ceci ne permet pas de tirer parti de la structure de la formule initiale dans la recherche d'une réfutation. De plus, la mise sous forme clausale standard est une transformation qui peut, dans le pire des cas, produire une formule dont la taille est exponentiellement plus grande que celle de la formule initiale. En règle générale, l'élimination des équivalences et l'utilisation des lois de distributivité conduisent souvent à de nombreuses redondances dans la forme clausale. Il en résulte pour certaines classes de formules une explosion combinatoire qui empêche une recherche efficace par Résolution de preuves de théorèmes "simples" [Ble77] [And81].

D'autre part, les preuves obtenues par Résolution étant très peu lisibles, il est difficile de fournir à l'utilisateur une justification compréhensible de la validité d'un théorème ou de la fausseté d'une conjecture. Ce manque de lisibilité empêche aussi l'utilisateur de guider efficacement un démonstrateur de théorèmes utilisant la Résolution dans la recherche d'une réfutation. Or, une recherche purement automatique de preuves de théorèmes difficiles étant pour l'instant illusoire, il est nécessaire de mettre en œuvre des calculs permettant une interaction aisée avec l'utilisateur.

Ces considérations ont conduit à un regain d'intérêt pour des méthodes de preuve

qui ne nécessitent pas une quelconque forme normale et dont les règles d'inférence reflètent autant que possible des étapes élémentaires de raisonnement apparaissant dans les preuves mathématiques usuelles. Ce sont les méthodes dites de *Déduction Naturelle*, héritées des travaux de Gentzen [Gen69], Prawitz [Pra65] et Smullyan [Smu68]. Ces méthodes ont été utilisées par plusieurs systèmes de démonstration de théorèmes [Ble71] [BB74] [OS88] [KZ90].

D'autres méthodes non clausales utilisant l'unification telles la méthode des matings [And81], la méthode des connexions [Bib81] ou la Résolution non clausale [Mur82] ont été proposées. Comme le remarque P. Andrews [And81],

> Of course, the fact that natural deduction provides a congenial format for communicating proofs dœs not necessarily mean that it provides the best context for discovering them. One can envision theorem proving systems which use a variety of methods to discover the essential ingredients of a proof and then construct proofs in whatever style is most congenial to the reader.

Dans [And80] il est montré comment extraire des matings les informations nécessaires à la construction d'une preuve en Déduction Naturelle. La construction de telles preuves peut être considérablement facilitée par l'utilisation des "expansion tree proofs" [Mil84], [Mil87]. Cette structure de données est une représentation compacte des preuves à partir de laquelle il est aisé de produire une preuve dans divers systèmes de Déduction Naturelle. Dans [Pfe84], il est montré comment construire un arbre d'expansion à partir des informations contenues dans une preuve par Résolution. Ceci permet d'envisager une intégration entre Résolution et Déduction Naturelle [MF86].

Cependant, l'approche retenue dans [Pfe84] n'est pas totalement satisfaisante:

- lors de la construction d'une preuve utilisant les arbres d'expansion, de nombreuses informations contenues dans la preuve par Résolution – et en particulier sa *structure* – sont perdues. Comme le remarque D. Miller, ces informations ne sont pas nécessaires à la présentation d'une preuve dans un système de Déduction Naturelle. Cependant, la perte de ces informations empêche de comparer dans un système donné de Déduction Naturelle *la structure de deux preuves* par Résolution. Un telle comparaison est importante si l'on veut pouvoir juger du caractère naturel d'une stratégie de Résolution.

- d'autre part, la traduction présentée par F. Pfenning n'est applicable que si

la réfutation par Résolution obeit à certaines restrictions. Ces restrictions conduisent à des preuves par Résolution possédant des caractéristiques peu souhaitables. Par exemple, la réfutation de la formule $\neg[(A \vee \neg A) \wedge (A \vee \neg A)]$ devra contenir des tautologies et des clauses contenant deux occurrences d'un même littéral ne pouvant être factorisées. En fait ces restrictions sont liées à l'utilisation de la mise sous forme clausale standard.

Dans cette thèse, nous nous proposons de contribuer à l'intégration et à l'implémentation de mécanismes de Déduction Naturelle dans des démonstrateurs utilisant la Résolution.

Dans une première partie, nous montrerons que l'utilisation d'une forme clausale non standard permet d'établir une correspondance naturelle entre les réfutations par Résolution et les déductions d'un calcul des séquents proche du calcul LK de Gentzen [Gen69]. Une telle correspondance permet d'envisager de tirer profit des avantages respectifs des deux approches de la démonstration de théorèmes – l'une automatique ("machine oriented"), l'autre interactive ("human oriented")– en construisant efficacement par Résolution des preuves pouvant être interprétées de manière naturelle en preuve dans des systèmes de Déduction Naturelle. En fait, ce problème comporte de nombreuses similitudes avec celui posé par L. Wos (problème n^o 24 dans [Wos88]):

> Is there a mapping between clause representation and natural-deduction representation (and corresponding inference rules and strategies) that causes reasoning programs based respectively on the two approaches or paradigms to attack a given assignment in an essentially identical fashion ?

Au chapitre I, nous présentons une transformation introduite en 1920 par T. Skolem [Sko67] et appelée *renommage*. L'utilisation systématique du renommage permet d'obtenir à partir d'une formule ϕ une formule ϕ', faiblement équivalente[1] à ϕ, dont la structure reflète la structure de ϕ [Tse68], [PG86]. La taille (mesurée en nombre de symboles) de la formule obtenue par mise sous forme clausale standard de la formule ϕ' est au plus égale au carré de la taille de ϕ : l'utilisation du renommage permet donc d'obvier au comportement exponentiel de la mise sous forme clausale standard.

[1] Deux formules sont faiblement équivalentes si et seulement si tout modèle de l'une est une expansion d'un modèle de l'autre.

Puis, après avoir rappelé quelques propriétés importantes du principe de Résolution (chapitre II), nous introduirons un calcul des séquents proche de celui de Gentzen (chapitre III). Nous montrerons que l'on peut *simuler pas à pas* une classe importante en démonstration automatique de déductions de ce calcul (chapitre IV). Nous montrerons aussi comment étendre ces résultats à d'autres classes de dérivations et nous établirons en fait qu'il est possible de simuler *polynomialement* notre calcul des séquents par la Résolution.

Cette étude nous permettra de traduire de manière naturelle les dérivations par Résolution en preuves dans notre système de Déduction Naturelle (chapitre V). A la différence de la méthode proposée dans [Pfe84], il ne sera pas nécessaire d'utiliser de structure de données auxiliaire pour obtenir cette traduction. De plus, notre méthode de traduction *préserve la structure des preuves* et permet de comparer la structure de preuves par Résolution tout en considérant une présentation de celles-ci dans un calcul en Déduction Naturelle.

Enfin nous discuterons comment certaines modifications de la transformation sous forme clausale peuvent contribuer à *l'amélioration de l'efficacité* de la Résolution (chapitre VI). Nous montrerons également qu'il est possible de *restreindre l'espace de recherche de la Résolution* afin que celui-ci corresponde plus exactement à celui du calcul des séquents. Nous montrerons en outre que certaines simplifications utilisées par des démonstrateurs utilisant de systèmes de déduction proche du calcul des séquents peuvent être reproduites dans le cadre de la Résolution.

Dans une deuxième partie, nous présenterons des algorithmes et des structures de données permettant de répondre efficacement à deux problèmes clés en démonstration automatique.

- L'un concerne la recherche et la sélection de tous les éléments d'un ensemble de termes qui filtrent, sont filtrés ou s'unifient avec un terme donné (chapitre VII).

- L'autre concerne les techniques à mettre en œuvre pour assurer une implémentation efficace de la Résolution avec sortes ordonnées. Nous nous intéresserons tout particulièrement au traitement de l'unification. En effet, si le problème de l'unification ordo-sortée a été bien étudié d'un point de vue théorique [SS87], [Kir88], [JK90], ... la mise en œuvre des solutions proposées et leur intégration aux algorithmes utilisés pour implémenter des démonstrateurs de théorèmes n'ont que peu été considérées [Cha88], [Cha89]. Ceci fera l'objet du chapitre VIII.

Les techniques d'implémentation présentées dans la deuxième partie de cette thèse seront décrites dans le cadre de la Résolution et seront illustrées par des expérimentations réalisées sur le démonstrateur par Résolution et Paramodulation que nous avons implémenté dans le cadre du projet ATINF [BdlTCC88]. Cependant, il est clair que les solutions proposées peuvent être appliquées à un ensemble plus large de démonstrateurs.

Chapitre 0

Préliminaires

0.1 Langage du premier ordre

Un *alphabet* d'un langage du premier ordre consiste en les ensembles de symboles suivants:

- un ensemble de symboles *logiques* comprenant un ensemble de *connectifs* $\mathcal{C} = \{\wedge, \vee, \neg, \rightarrow, \leftrightarrow\}$ et un ensemble de *quantificateurs* $\mathcal{Q} = \{\forall, \exists\}$.
- ensemble $\mathcal{L}$ de *symboles non logiques* comprenant:
  - un ensemble infini V de variables;
  - un ensemble F de symboles *fonctionnels* $f_0 \ldots f_n, \ldots$; chaque symbole fonctionnel a une arité fixe. Les symboles fonctionnels d'arité nulle sont aussi appelés *constantes.*
  - un ensemble P de symboles de *prédicat* $P_0 \ldots P_n, \ldots$; chaque symbole de prédicat a une arité fixe. Les symboles de prédicat d'arité nulle sont aussi appelés *propositions.*

Ces ensembles sont supposés deux à deux disjoints.

Un langage du premier ordre dont l'alphabet contient un ensemble $\mathcal{L}$ de symboles non logiques sera appelé le langage $\mathcal{L}$.

Etant donné un langage $\mathcal{L} = (V, F, P)$, l'ensemble $T(F, X)$ des *termes* construits sur l'ensemble de symboles fonctionnels F et l'ensemble de variables $X \subseteq V$ est le plus petit ensemble vérifiant:

- si v est une variable de X alors v est un terme de $T(F, X)$
- si f est un symbole fonctionnel d'arité $n \geq 0$ et si $t_1 \ldots t_n$ sont des termes de $T(F, X)$, alors $f(t_1 \ldots t_n)$ est un terme de $T(F, X)$.

$T(F, \emptyset)$ est appelé l'ensemble des termes *clos* et est aussi noté $T(F)$.

Etant donné un langage $\mathcal{L} = (V, F, P)$, l'ensemble $A(P, F, X)$ des *formules atomiques* (encore appelées *atomes*) construits sur l'ensemble de symboles de prédicat P, l'ensemble de symboles fonctionnels F et l'ensemble de variables $X \subseteq V$ est le plus petit ensemble vérifiant:

- si P est un symbole de prédicat d'arité $n \geq 0$ et si $t_1 \ldots t_n$ sont des termes de $T(F, X)$, alors $P(t_1 \ldots t_n)$ est un terme de $A(P, F, X)$.

$A(P, F, \emptyset)$ est appelé l'ensemble des atomes *clos* et est aussi noté $A(P, F)$.

Si $< t_1 \ldots t_n >$ désigne une séquence de termes et si P est un symbole de prédicat d'arité $m \leq n$, on note $P[t_1 \ldots t_n]$ un atome de symbole de prédicat P et dont les m arguments sont choisis deux à deux distincts parmi $< t_1 \ldots t_n >$ et tels que, si t_{i_j} et t_{i_k} dénotent respectivement les j-ème et k-ième arguments de cet atome, $1 \leq j < k \leq m$, alors $1 \leq i_j < i_k \leq n$.

Etant donné un langage $\mathcal{L} = (V, F, P)$, l'ensemble des *formules bien formées* de $\mathcal{L}$ (encore appelé ensemble des $\mathcal{L}$-formules) est le plus petit ensemble vérifiant:

- si A est un élément de $A(P, F, V)$, alors A est une $\mathcal{L}$-formule. L'ensemble des sous-formules immédiates de A est vide
- si ϕ_1, ϕ_2 sont des $\mathcal{L}$-formules et x est une variable de V alors
  - $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$, $\phi_1 \leftrightarrow \phi_2$ sont des $\mathcal{L}$-formules dont les sous-formules immédiates sont ϕ_1, ϕ_2.
  - $\neg\phi_1$, $\forall x \phi_1$, $\exists x \phi_1$, sont des $\mathcal{L}$-formules ayant pour unique sous-formule immédiate ϕ_1.

Si ϕ est une $\mathcal{L}$-formule, on note $SFI(\phi)$ l'ensemble des sous-formules immédiates de ϕ. L'ensemble $SF(\phi)$ des sous-formules de ϕ est l'ensemble:

$$\{\phi\} \cup [\cup_{\psi \in SFI(\phi)} SF(\psi)]$$

$SFA(\phi)$ l'ensemble des sous-formules atomiques de ϕ. On note $\psi \sqsubseteq \phi$ le fait que ψ soit une sous-formule de ϕ.

0.2 Occurrence

La notion d'occurrence utilisée ici est similaire à celle décrite dans [Gal86]. On rappelle simplement qu'une occurrence est une chaîne finie d'entiers et que l'on note Λ la chaîne vide.

On note $\psi[\phi]_O$ une formule ψ ayant ϕ comme sous-formule à l'occurrence O; on désignera ϕ par la notation $\psi\,|_O$. De même, si t et s désignent des termes, on note $t[s]_O$ un terme t ayant s pour sous-terme à l'occurrence O. On omettra le suffixe O si l'occurrence de la sous-formule ou du sous-terme est sous-entendue.

On notera $\psi[O \leftarrow \phi]$ la formule obtenue en remplaçant la sous-formule apparaissant à l'occurrence O dans ψ par ϕ. De même, on notera $\psi[\phi_1 \leftarrow \phi_2]$ le remplacement dans ϕ de toutes les occurrences de ϕ_1 dans ψ par ϕ_2.

0.3 Polarité d'une sous-formule

Définition: La *polarité* d'une sous-formule ψ apparaissant à l'occurrence O dans une formule ϕ est définie par:

- ψ est une sous-formule de polarité *positive* si et seulement si $pol(O, \phi) = 1$;
- ψ est une sous-formule de polarité *négative* si et seulement si $pol(O, \phi) = -1$;
- ψ est une sous-formule de polarité *nulle* si et seulement si $pol(O, \phi) = 0$;

où $pol(O, \phi)$ est définie inductivement par:

- $pol(\Lambda, \phi) = 1$;
- si $O = i.O'$, $1 \leq i \leq n$ et $pol(O', \phi_i) = p$ alors $pol(O, \bigwedge_{i=1}^{n} \phi_i) = p$ et $pol(O, \bigvee_{i=1}^{n} \phi_i) = p$;

- si $O = 1.O'$ et $pol(O', \phi_1) = p$ alors $pol(O, \forall x \phi_1) = p$, $pol(O, \exists x \phi_1) = p$, $pol(O, \phi_1 \to \phi_2) = -p$, $pol(O, \neg \phi_1) = -p$, $pol(O, \phi_1 \leftrightarrow \phi_2) = 0$;
- si $O = 2.O'$ et $pol(O', \phi_2) = p$ alors $pol(O, \phi_1 \to \phi_2) = p$, $pol(O, \phi_1 \leftrightarrow \phi_2) = 0$

La notation abusive $pol(\psi, \phi)$ sera utilisée si l'on sous-entend que ψ apparaît à l'occurrence O dans ϕ. $\diamond$

Les formules de polarité nulle sont donc les formules apparaissant dans la portée d'un connectif $\leftrightarrow$; on appellera formule *linéaire* une formule ne comportant pas de connectif $\leftrightarrow$.

0.4 Variables libres, variables liées

L'ensemble $FV(t)$ des *variables libres* d'un terme t est défini inductivement par:

- si t est une variable alors $FV(t) = \{t\}$
- si $t = f(t_1 \dots t_n)$ alors $FV(t) = \cup_{i=1}^{n} FV(t_i)$

L'ensemble $FV(\phi)$ des *variables libres* d'une formule ϕ est défini inductivement par:

- si ϕ est un atome $P(t_1 \dots t_n)$ alors $FV(\phi) = \cup_{i=1}^{n} FV(t_i)$
- si $\phi = \neg \phi_1$ alors $FV(\phi) = FV(\phi_1)$
- si $\phi = \phi_1 * \phi_2$, $* \in \{\wedge, \vee, \to, \leftrightarrow\}$ alors $FV(\phi) = FV(\phi_1) \cup FV(\phi_2)$
- si $\phi = Qx\phi_1$, $Q \in \{\forall, \exists\}$ alors $FV(\phi) = FV(\phi_1) \setminus \{x\}$

Suivant le contexte, on adoptera aussi la notation $VAR(t)$, $VAR(\phi)$ pour désigner respectivement $FV(t)$, $FV(\phi)$.

L'ensemble $BV(\phi)$ des *variables liées* d'une formule ϕ est définit inductivement par:

- si ϕ est un atome $P(t_1 \dots t_n)$ alors $BV(\phi) = \emptyset$
- si $\phi = \neg \phi_1$ alors $BV(\phi) = BV(\phi_1)$
- si $\phi = \phi_1 * \phi_2$, $* \in \{\wedge, \vee, \to, \leftrightarrow\}$ alors $BV(\phi) = BV(\phi_1) \cup BV(\phi_2)$
- si $\phi = Qx\phi_1$, $Q \in \{\forall, \exists\}$ alors $BV(\phi) = BV(\phi_1) \cup \{x\}$

Définition: Une formule ϕ est *rectifiée* si

- l'ensemble de variables libres de ϕ est distinct de l'ensemble des variables liées de ϕ;
- deux occurrences distinctes de quantificateurs quantifient des variables distinctes.

$\diamond$

0.5 Substitution

Une *substitution* est une application de V dans $T(F,V)$ qui est l'identité presque partout.

Etant donné une substitution σ, on notera:

- $Dom(\sigma) = \{x/x \in V \text{ et } \sigma(x) \neq x\}$;
- $Cod(\sigma) = \{\sigma(x)/x \in V \text{ et } \sigma(x) \neq x\}$;
- $VCOD(\sigma) = \cup_{x \in Dom(\sigma)} VAR(\sigma(x))$.

Toute substitution σ peut être étendue en un endomorphisme de $T(F,V)$, noté $\bar{\sigma}$, et défini inductivement par:

- si $x \in V$ alors $\bar{\sigma}(x) = \sigma(x)$
- si $s = f(s_1 \ldots s_n)$ et si $\bar{\sigma}(s_i) = t_i$ $1 \leq i \leq n$ alors $\bar{\sigma}(s) = f(t_1 \ldots t_n)$.

Par abus de notation, on notera cet endomorphisme simplement σ et on l'appellera aussi une substitution.

De même, si ϕ une $\mathcal{L}$-formule et si σ est une substitution, on définit inductivement $\sigma(\phi)$ par:

- si ϕ est un atome $P(s_1 \ldots s_n)$ si $\sigma(s_i) = t_i$ $1 \leq i \leq n$ alors alors $\sigma(\phi) = P(t_1 \ldots t_n)$
- si $\phi = \neg\phi_1$ alors $\sigma(\phi) = \neg\sigma(\phi_1)$
- si $\phi = \phi_1 * \phi_2$, $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$ alors $\sigma(\phi) = \sigma(\phi_1) * \sigma(\phi_2)$
- si $\phi = Qx\phi_1$, $Q \in \{\forall, \exists\}$ et si σ' est une substitution telle que

$$Dom(\sigma') = Dom(\sigma) \setminus \{x\} \quad \forall y \in Dom(\sigma')\ \sigma'(y) = \sigma(y)$$

alors $\sigma(\phi) = Qx\sigma'(\phi_1)$

Si $Dom(\sigma) = \{x_1 \dots x_n\}$ et $\sigma(x_i) = t_i$, $1 \leq i \leq n$, on notera aussi $\sigma(\phi)$ sous la forme $\phi(x_1/t_1, \dots, x_n/t_n)$

Etant donnée une formule ϕ, un terme t et une variable $y \in VAR(t)$, on dit que la variable y est *capturée* dans $\phi(x/t)$ lors de la substitution de x par t dans ϕ si il existe une sous-formule $Qy\phi_1$, $Q \in \{\exists, \forall\}$, telle que $x \in FV(Qy\phi_1)$.

0.6 Sémantique

On rappelle que:

Définition: Soit $\mathcal{L}$ un langage du premier ordre. Une *$\mathcal{L}$-structure* $\mathcal{M}$ est une paire (D, I) où D est un ensemble non vide appelé *domaine* et I est une fonction appelée *interprétation* qui associe à chaque symbole de fonction n-aire de $\mathcal{L}$ une fonction de $(D^n \rightarrow D)$ et qui associe à chaque symbole de prédicat n-aire de $\mathcal{L}$ une relation sur D^n. $\diamond$

Si ϕ est une formule satisfaisable dans $\mathcal{M}$ pour une *assignation* σ (c'est-à-dire une application qui associe à chacune des n variables libres $\{x_1 \dots x_n\}$ de ϕ un élément de D), on notera:

$$\mathcal{M} \models^{\sigma} \phi$$

ou bien, si σ associe $a_i \in D$ à la variable $x_i, 1 \leq i \leq n$

$$\mathcal{M} \models^{x_1 \dots x_n}_{a_1 \dots a_n} \phi$$

Définition: Etant donnés deux langages du premier ordre $\mathcal{L}$ et $\mathcal{L}'$, $\mathcal{L}'$ est une *expansion* de $\mathcal{L}$ si et seulement si $\mathcal{L}$ est un sous-ensemble de $\mathcal{L}'$.

Si $\mathcal{L}'$ est une expansion de $\mathcal{L}$, une $\mathcal{L}'$ structure $\mathcal{M}' = (D', I')$ est une *expansion* d'une $\mathcal{L}$ structure $\mathcal{M} = (D, I)$ si $D = D'$ et I est la restriction de I' au langage $\mathcal{L}$. $\diamond$

Partie I

Résolution et Déduction Naturelle

Chapitre 1

Renommages et mise sous forme clausale

Dans ce chapitre, nous étudions diverses transformations qui peuvent être utilisées pour effectuer la mise sous forme clausale d'une formule ϕ.

Les transformations utilisées par la mise sous forme clausale standard peuvent produire, dans le pire des cas, un nombre exponentiel de clauses: la taille, mesurée en nombre de symboles, de la forme clausale ainsi obtenue est alors elle aussi exponentielle. De plus, la forme clausale obtenue par cette transformation ne reflète pas la structure de la formule initiale.

Afin de remédier à ces difficultés, nous présentons une transformation appelée *renommage*. Cette transformation, introduite en 1920 par T. Skolem [Sko67], permet de nommer des sous-formules d'une formule donnée en utilisant de nouveaux symboles de prédicats. L'utilisation systématique du renommage permet d'obtenir à partir d'une formule ϕ une formule ϕ', faiblement équivalente à ϕ, dont la structure reflète la structure de ϕ. La taille (mesurée en nombre de symboles) de la formule obtenue par mise sous forme clausale standard de la formule ϕ' est au plus

égale au carré de la taille de ϕ : l'utilisation du renommage permet donc d'obvier au comportement exponentiel de la mise sous forme clausale standard.

L'utilisation du renommage dans la mise sous forme clausale est mentionnée pour la première fois dans [Tse68] dans le cadre du calcul propositionnel. Des généralisations à la logique du premier ordre de cette transformation ont été décrites par plusieurs auteurs [GNOP82] [Ede84] [PG86]. Ce dernier article présente une transformation dite "structure preserving", très proche de celle décrite dans ce chapitre, dont le caractère quadratique en taille est mentionné. Cependant, la preuve décrite dans [PG86], souvent elliptique, n'est pas satisfaisante: elle ne s'applique qu'à des formules ne contenant que des connectifs $\wedge$, $\vee$ binaires et des quantificateurs portant sur une variable. Conformément à l'usage en démonstration automatique, nous considèrerons des formules construites en utilisant des connectifs $\wedge$, $\vee$ n-aires et des quantificateurs portant sur un nombre quelconque de variables et nous montrerons que la mise sous forme clausale étudiée dans ce chapitre est une transformation quadratique en taille.

Un tel résultat peut être obtenu en utilisant les résultats de [BdlT91] où un formalisme ainsi que des théorèmes généraux permettant l'étude et la comparaison de plusieurs transformations sous forme clausale sont proposés. Notre approche, moins ambitieuse, s'attache à comprendre et à évaluer comment chacune des transformations de la mise sous forme clausale standard contribue à l'accroissement en taille de la forme clausale dans le cas particulier où ces transformations sont appliquées à une formule préalablement renommée. Cela permet de borner plus précisement la taille de la forme clausale obtenue avec la transformation considérée dans ce chapitre.

1.1 Mise sous forme clausale

Afin de pouvoir montrer par Résolution l'insatisfaisabilité d'une formule ϕ, il est nécessaire de transformer celle-ci en une formule ϕ' sous forme clausale. Rappelons que:

Définition:[[Gal86]] Une formule ϕ' est *sous forme clausale* si et seulement si:

- ϕ' ne possède pas de variable libre;
- ϕ' est une conjonction de formules sous forme prénexe;
- le préfixe de chaque conjoint de ϕ' ne comporte que des quantificateurs universels;

- deux occurrences distinctes de quantificateurs quantifient des variables distinctes: il s'agit donc de formules *rectifiées*;
- la matrice de chaque conjoint de ϕ' est une disjonction dont chaque disjoint est soit une formule atomique, soit la négation d'une formule atomique. Une telle disjonction est appelée une *clause*.

$\diamond$

Cette définition diffère de celle généralement utilisée dans la presque totalité des publications sur le sujet. Cependant, cette définition se révèlera être bien adaptée à notre étude.

Etant donnée une formule ϕ dont l'ensemble des variables libres est $\{x_1 \ldots x_n\}$, la mise sous forme clausale standard de ϕ consiste à appliquer à la formule $\forall x_1 \ldots x_n \phi$ les règles suivantes *dans l'ordre indiqué ci-après*:

1. *Mise sous forme normale négative* de la formule: il s'agit d'éliminer les connectifs $\leftrightarrow$ et $\rightarrow$ et de restreindre la portée des négations aux sous-formules atomiques. Classiquement, on utilise pour ce faire les transformations suivantes:
   - $\phi_1 \leftrightarrow \phi_2 \longrightarrow (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$
   - $\phi_1 \leftrightarrow \phi_2 \longrightarrow (\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2)$
   - $\phi_1 \rightarrow \phi_2 \longrightarrow \neg\phi_1 \vee \phi_2$
   - $\neg(\phi_1 \wedge \phi_2) \longrightarrow \neg\phi_1 \vee \neg\phi_2$
   - $\neg(\phi_1 \vee \phi_2) \longrightarrow \neg\phi_1 \wedge \neg\phi_2$
   - $\neg(\forall x \phi_1) \longrightarrow \exists x \neg\phi_1$
   - $\neg(\exists x \phi_1) \longrightarrow \forall x \neg\phi_1$
   - $\neg\neg\phi_1 \longrightarrow \phi_1$
2. *Minimisation de la portée des quantificateurs.* Cette opération n'est pas nécessaire à l'obtention de la forme clausale. Cependant, elle permet de minimiser l'arité des fonctions de Skolem ce qui, en pratique, se révèle souvent intéressant. Cette transformation peut être définie en utilisant les règles:
   - $\forall x \forall y \phi \longrightarrow \forall y \forall x \phi$
   - $\forall x(\phi_1 \wedge \phi_2) \longrightarrow (\forall x \phi_1) \wedge (\forall x \phi_2)$

- $\forall x(\phi_1 \vee \phi_2) \longrightarrow (\forall x\phi_1) \vee \phi_2$ si x n'est pas libre dans ϕ_2;
- $\forall x(\phi_1 \vee \phi_2) \longrightarrow \phi_1 \vee (\forall x\phi_2)$ si x n'est pas libre dans ϕ_1;
- $\exists x \exists y \phi \longrightarrow \exists y \exists x \phi$
- $\exists x(\phi_1 \vee \phi_2) \longrightarrow (\exists x\phi_1) \vee (\exists x\phi_2)$
- $\exists x(\phi_1 \wedge \phi_2) \longrightarrow (\exists x\phi_1) \wedge \phi_2$ si x n'est pas libre dans ϕ_2;
- $\exists x(\phi_1 \wedge \phi_2) \longrightarrow \phi_1 \wedge (\exists x\phi_2)$ si x n'est pas libre dans ϕ_1;
- $(\phi_1 \wedge (\phi_2 \wedge \phi_3)) \longrightarrow ((\phi_1 \wedge \phi_2) \wedge \phi_3)$
- $((\phi_1 \wedge \phi_2) \wedge \phi_3) \longrightarrow (\phi_1 \wedge (\phi_2 \wedge \phi_3))$
- $(\phi_1 \wedge \phi_2) \longrightarrow (\phi_2 \wedge \phi_1)$
- $(\phi_1 \vee (\phi_2 \vee \phi_3)) \longrightarrow ((\phi_1 \vee \phi_2) \vee \phi_3)$
- $((\phi_1 \vee \phi_2) \vee \phi_3) \longrightarrow (\phi_1 \vee (\phi_2 \vee \phi_3))$
- $(\phi_1 \vee \phi_2) \longrightarrow (\phi_2 \vee \phi_1)$

3. Renommage des variables liées dont le nom est partagé par deux quantificateurs;
   - $\forall x\phi_1 \longrightarrow \forall y\phi_1(x/y)$ où y est une nouvelle variable.
   - $\exists x\phi_1 \longrightarrow \exists y\phi_1(x/y)$ où y est une nouvelle variable.
4. *Skolemisation*
   - $\exists x\phi_1 \longrightarrow \phi_1(x/f_x(x_1 \ldots x_n))$ où $\{x_1 \ldots x_n\}$ est l'ensemble de variables libres de ϕ_1 et f_x est un nouveau symbole de fonction appelé *fonction de Skolem* associée à x;
5. *Mise sous forme prénexe:*
   - $(\forall x\phi_1) \wedge \phi_2 \longrightarrow \forall x(\phi_1 \wedge \phi_2)$
   - $(\forall x\phi_1) \vee \phi_2 \longrightarrow \forall x(\phi_1 \vee \phi_2)$
   - $\phi_1 \wedge (\forall x\phi_2) \longrightarrow \forall x(\phi_1 \wedge \phi_2)$
   - $\phi_1 \vee (\forall x\phi_2) \longrightarrow \forall x(\phi_1 \vee \phi_2)$

Afin d'assurer la correction de ces transformations, il est important de rappeler ici que, la formule initiale sur laquelle ces transformations sont appliquées ne contenant pas de variable libre et deux occurrences distinctes de quantificateurs quantifiant des variables distinctes, la variable x n'est pas libre dans ϕ_2

dans les deux premières règles et n'est pas libre dans ϕ_1 dans les deux dernières règles.

6. *Réduction sous forme normale conjonctive* de la matrice de la formule ainsi obtenue par application des lois de distributivité de la forme:

   - $(\phi_1 \wedge \phi_2) \vee \phi \longrightarrow (\phi_1 \vee \phi) \wedge (\phi_2 \vee \phi)$
   - $\phi \vee (\phi_1 \wedge \phi_2) \longrightarrow (\phi \vee \phi_1) \wedge (\phi \vee \phi_2)$

7. *Réduction sous forme clausale* en utilisant

   - $\forall x(\phi_1 \wedge \phi_2) \longrightarrow (\forall x\phi_1) \wedge (\forall x\phi_2)$

   puis en renommant les variables quantifiées par deux occurrences distinctes de quantificateur par la transformation:

   - $\forall x\phi_1 \longrightarrow \forall y\phi_1(x/y)$ où y est une nouvelle variable.

L'opération de skolemisation mise à part, chacune de ces transformations $\phi_1 \longrightarrow \phi_2$ correspond en fait à une formule valide $\phi_1 \leftrightarrow \phi_2$. La formule obtenue après de telles transformations est donc logiquement équivalente à la formule initiale. Par contre la skolemisation ne préserve que la satisfaisabilité [CL73] [Lov78]: en fait, la formule initiale et la formule obtenue après skolemisation sont faiblement équivalentes [BdlT89].

L'élimination des équivalences, encore appelée *linéarisation* , est obtenue en utilisant les transformations:

$$\phi_1 \leftrightarrow \phi_2 \longrightarrow (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1) \tag{1.1}$$

$$\phi_1 \leftrightarrow \phi_2 \longrightarrow (\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2) \tag{1.2}$$

Le choix de l'application de l'une ou de l'autre de ces transformations n'est pas sans conséquences, comme le remarquent [HLO+80].

Considérons la formule $F_1 = \neg(p \leftrightarrow q)$ contenant une équivalence de polarité négative. Si l'on linéarise F_1 en utilisant 1.1, on obtient:

$$\neg((p \rightarrow q) \wedge (q \rightarrow p))$$

qui, après réduction sous forme clausale, produit les clauses:

$$(p \vee q) \wedge (p \vee \neg p) \wedge (\neg q \vee q) \wedge (\neg q \vee \neg p)$$

dont deux sont des clauses tautologiques qui peuvent être éliminées. Par contre, si l'on utilise 1.2 pour linéariser F_1, on obtient:

$$\neg((p \wedge q) \vee (\neg p \wedge \neg q))$$

qui conduit à

$$(\neg p \vee \neg q) \wedge (p \vee q)$$

sans générer de clauses tautologiques.

Inversement, si l'on considère la formule $F_2 = (p \leftrightarrow q)$ contenant une équivalence de polarité positive, la forme clausale obtenue en utilisant 1.2 contient des clauses tautologiques alors que celle obtenue en utilisant 1.1 n'en contient pas.

Dans le cas de formules du premier ordre, il n'est pas toujours possible d'effectuer sur la forme clausale des simplifications a posteriori: ceci est dû à la skolemisation comme le montre l'exemple suivant. Considérons la formule F_3:

$$\neg((\exists x p(x)) \leftrightarrow (\exists y q(y)))$$

En utilisant 1.1, on obtient

$$\neg[((\exists x p(x)) \rightarrow (\exists y q(y))) \wedge ((\exists y q(y)) \rightarrow (\exists x p(x)))]$$

d'où, après introduction des constantes de Skolem a_x, b_y:

$$(p(a_x) \vee q(b_y)) \wedge \forall x_1(p(a_x) \vee \neg p(x_1)) \wedge \forall y_1(\neg q(y_1) \vee q(b_y)) \wedge \forall x_2, y_2(\neg q(y_2) \vee \neg p(x_2))$$

Cette forme clausale ne peut être simplifiée.

L'utilisation de 1.2 conduit à

$$\neg[((\exists x p(x)) \wedge (\exists y q(y))) \vee (\neg(\exists x p(x)) \wedge \neg(\exists y q(y)))$$

qui, après skolemisation, donne la forme clausale plus simple:

$$\forall x_1, y_1(\neg p(x_1) \vee \neg q(y_1)) \wedge (p(a_x) \vee q(b_y))$$

Ces remarques conduisent à procéder à l'élimination des connectifs $\leftrightarrow$ en parcourant la formule (considérée comme un arbre) de la racine vers les feuilles et en appliquant 1.1 aux formules de polarité positive et 1.2 aux formules de polarité négatives.

Malgré cette optimisation, la mise sous forme clausale standard souffre de deux inconvénients majeurs:

- Cette transformation peut produire dans le pire des cas une forme clausale dont la taille (mesurée en nombre de symboles apparaissant dans la formule) est exponentiellement plus grande que celle de la formule initiale. Le caractère exponentiel de cette transformation est dû d'une part à l'opération de linéarisation (étape 1) et d'autre part à la mise sous forme normale conjonctive de la matrice obtenue après les transformations 1 à 5. Par exemple, la mise sous forme normale conjonctive de $r_1 \leftrightarrow (r_2 \leftrightarrow (\ldots \leftrightarrow r_n)\ldots))$ produit $O(2^n)$ clauses. De même, l'application de la règle 6 sur la formule $\bigvee_{i=1}^{n}(A_i \wedge B_i)$ comportant $3n+1$ symboles produit un ensemble de 2^n clauses, chaque clause comportant n littéraux.

- Le résultat de cette transformation est une formule dont la structure ne reflète pas la structure de la formule originale.

1.2 Renommage d'une sous-formule

Il est possible de remédier à ces inconvénients en *renommant* certaines sous-formules de la formule à mettre sous forme normale conjonctive: cette opération consiste à introduire de nouveaux littéraux comme faisant réference à ces sous-formules.

Définition: Etant donné un langage du premier ordre $\mathcal{L}$, soit ϕ une $\mathcal{L}$-formule et soit ψ une sous-formule de ϕ telle que:

- ψ apparaît à l'occurrence O dans ϕ;

- $\{x_1 \ldots x_n\}$ est l'ensemble des variables libres de ψ

Le *renommage* de ψ dans ϕ désigne la formule

$$\phi[O \leftarrow NP_\psi(x_1 \ldots x_n)] \wedge Def(\psi, \phi)$$

où

- $NP_\psi \notin \mathcal{L}$ est un nouveau symbole de prédicat d'arité n, associé à l'occurrence O de ϕ et appelé *prédicat de Skolem.*

- $\phi[O \leftarrow NP_\psi(x_1 \ldots x_n)]$ est le résultat du remplacement dans ϕ à l'occurrence O de la sous-formule ψ par $NP_\psi(x_1 \ldots x_n)$.

- $Def(\psi, \phi)$, la *définition* du *littéral de Skolem* $NP_\psi(x_1 \ldots x_n)$ dans ϕ dénote l'une des formules suivantes:

- $\forall x_1 \ldots x_n (NP_\psi(x_1 \ldots x_n) \rightarrow \psi)$ si ψ est une sous-formule de polarité positive de ϕ;

- $\forall x_1 \ldots x_n (\psi \rightarrow NP_\psi(x_1 \ldots x_n))$ si ψ est une sous-formule de polarité négative de ϕ;

- $\forall x_1 \ldots x_n (NP_\psi(x_1 \ldots x_n) \leftrightarrow \psi)$ si ψ est une sous-formule de polarité nulle de ϕ.

$\diamond$

Par la suite, $\phi_1 \rightleftharpoons^p \phi_2$ dénotera la formule:

- $\phi_1 \rightarrow \phi_2$ si $p = +1$;
- $\phi_2 \rightarrow \phi_1$ si $p = -1$;
- $\phi_1 \leftrightarrow \phi_2$ si $p = 0$.

D'où la notation de la définition d'un littéral de Skolem d'une sous-formule ψ de ϕ:

$$\forall x_1 \ldots x_n (NP_\psi(x_1 \ldots x_n) \rightleftharpoons^p \psi)$$

si ψ est une sous-formule de polarité p de ϕ.

De plus, s'il n'y a pas d'ambiguïté sur l'occurrence de la sous-formule ψ renommée, nous parlerons du renommage de ψ dans ϕ au lieu du renommage de l'occurrence O de ψ dans ϕ.

Renommer une sous-formule est une transformation qui préserve la satisfaisabilité:

Théorème 1 *Soit ϕ une formule construite sur un langage du premier ordre $\mathcal{L}$ et soit $\mathcal{M} = (D, I)$ une $\mathcal{L}$-structure telle que $\mathcal{M} \models^\sigma \phi$. Soit $\mathcal{L}' = \mathcal{L} \cup \{NP_\psi\}$ où $NP_\psi \notin \mathcal{L}$ est le prédicat de Skolem associé à l'occurrence O de ψ dans ϕ. Alors il existe une $\mathcal{L}'$-structure $\mathcal{M}' = (D, I')$ telle que*

- $\forall f \in \mathcal{L}, I(f) = I'(f)$ *et*
- $\mathcal{M}' \models^\sigma \phi[O \leftarrow NP_\psi(x_1 \ldots x_n)] \wedge Def(\psi, \phi)$.

C'est-à-dire que $\mathcal{M}'$ est une expansion *de $\mathcal{M}$.*

Preuve: On définit I' par:

- $I'(NP_\psi) = \{< a_1 \dots a_n >\in D^n / \mathcal{M} \models^{x_1 \dots x_n}_{a_1 \dots a_n} \psi\}$
- $I'(f) = I(f)$, pour tout $f \in \mathcal{L}$.

On a, par construction,

$$\text{Pour tout } < a_1 \dots a_n >\in D^n, \mathcal{M}' \models^{x_1 \dots x_n}_{a_1 \dots a_n} (NP_\psi(x_1 \dots x_n) \leftrightarrow \psi) \qquad (1.3)$$

donc $\mathcal{M}' \models \forall x_1 \dots x_n (NP_\psi(x_1 \dots x_n) \leftrightarrow \psi)$ d'où $\mathcal{M}' \models Def(\psi, \phi)$. De 1.3 et $\mathcal{M} \models^\sigma \phi$, on montre facilement, par induction structurelle sur ϕ que:

$$\mathcal{M}' \models^\sigma \phi[O \leftarrow NP_\psi(x_1 \dots x_n)]$$

. D'où: $\mathcal{M}' \models^\sigma \phi[O \leftarrow NP_\psi(x_1 \dots x_n)] \wedge Def(\psi, \phi)$. C.Q.F.D.

Inversement, on montre facilement par induction sur la formule ϕ que:

Théorème 2 *Si $\phi' = (\phi[O \leftarrow NP_\psi(x_1 \dots x_n)] \wedge Def(\psi, \phi))$ est satisfaisable dans une $\mathcal{L}'$-structure $\mathcal{M}'$ pour une assignation σ définie sur les variables libres de ϕ', alors ϕ est satisfaisable dans $\mathcal{M}'$ pour σ.*

La preuve de ce théorème revient en fait à utiliser les théorèmes dits de *substitutivité de l'implication et de l'équivalence* tels qu'on les trouve par exemple dans [And86] pp54-55.

1.3 Propriétés du renommage exhaustif

Il est donc possible de renommer tout ou partie de l'ensemble des occurrences des sous-formules d'une formule ϕ avant de procéder à la mise sous forme clausale. Nous définissons ici $Rn(\phi)$ comme étant le résultat du renommage de toutes les occurrences des sous-formules de ϕ.

Définition: Le résultat du *renommage exhaustif* d'une formule ϕ dont l'ensemble des variables libres est $\{x_1 \dots x_n\}$ est la formule:

$$Rn(\phi) = \forall x_1 \dots x_n NP_\phi(x_1 \dots x_n) \wedge D_\phi$$

obtenue à partir de $< NP_\phi(x_1 \dots x_n), D_\phi >= Rn(\phi, +1)$, où $Rn(\phi', p)$ définit inductivement, pour une sous-formule ϕ' dont les variables libres sont $\{x_1 \dots x_n\}$ et qui apparaît avec une polarité p dans ϕ, le renommage exhaustif de toutes les occurrences des sous-formules de ϕ' dans ϕ:

$Rn(A,p) =$ $< NP_A(x_1 \ldots x_n), \forall x_1 \ldots x_n NP_A(x_1 \ldots x_n) \rightleftharpoons^p A >$
si A est une formule atomique.

$Rn(\neg\phi_1,p) =$ $< NP_{\neg\phi_1}(x_1 \ldots x_n),$
$(\forall x_1 \ldots x_n NP_{\neg\phi_1}(x_1 \ldots x_n) \rightleftharpoons^p (\neg NP_{\phi_1}(x_1 \ldots x_n))) \wedge D_{\phi_1} >$
avec $< NP_{\phi_1}(x_1 \ldots x_n), D_{\phi_1} >= Rn(\phi_1, -p).$

$Rn(\bigwedge_{i=1}^n \phi_i, p) =$ $< NP_{\bigwedge_{i=1}^n \phi_i}(x_1 \ldots x_n),$
$(\forall x_1 \ldots x_n NP_{\bigwedge_{i=1}^n \phi_i}(x_1 \ldots x_n) \rightleftharpoons^p (\bigwedge_{i=1}^n NP_{\phi_i}[x_1 \ldots x_n]))$
$\wedge \bigwedge_{i=1}^n D_{\phi_i} >$
avec $< NP_{\phi_i}[x_1 \ldots x_n], D_{\phi_i} >= Rn(\phi_i, p)\ 1 \leq i \leq n.$

$Rn(\bigvee_{i=1}^n \phi_i, p) =$ $< NP_{\bigvee_{i=1}^n \phi_i}(x_1 \ldots x_n),$
$(\forall x_1 \ldots x_n NP_{\bigvee_{i=1}^n \phi_i}(x_1 \ldots x_n) \rightleftharpoons^p (\bigvee_{i=1}^n NP_{\phi_i}[x_1 \ldots x_n]))$
$\wedge \bigwedge_{i=1}^n D_{\phi_i} >$
avec $< NP_{\phi_i}[x_1 \ldots x_n], D_{\phi_i} >= Rn(\phi_i, p) 1 \leq i \leq n.$

$Rn(\phi_1 \rightarrow \phi_2, p) =$ $< NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n),$
$(\forall x_1 \ldots x_n NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n) \rightleftharpoons^p$
$(NP_{\phi_1}[x_1 \ldots x_n] \rightarrow NP_{\phi_2}[x_1 \ldots x_n]))$
$\wedge D_{\phi_1} \wedge D_{\phi_2} >$ avec
$< NP_{\phi_1}[x_1 \ldots x_n], D_{\phi_1} >= Rn(\phi_1, -p)$ et
$< NP_{\phi_2}[x_1 \ldots x_n], D_{\phi_2} >= Rn(\phi_2, p).$

$Rn(\phi_1 \leftrightarrow \phi_2, p) =$ $< NP_{\phi_1 \leftrightarrow \phi_2}(x_1 \ldots x_n),$
$(\forall x_1 \ldots x_n NP_{\phi_1 \leftrightarrow \phi_2}(x_1 \ldots x_n) \rightleftharpoons^p$
$(NP_{\phi_1}[x_1 \ldots x_n] \leftrightarrow NP_{\phi_2}[x_1 \ldots x_n]))$
$\wedge D_{\phi_1} \wedge D_{\phi_2} >$ avec
$< NP_{\phi_1}[x_1 \ldots x_n], D_{\phi_1} >= Rn(\phi_1, 0)$ et
$< NP_{\phi_2}[x_1 \ldots x_n], D_{\phi_2} >= Rn(\phi_2, 0).$

$Rn(\forall x \phi_1, p) =$ $< NP_{\forall x \phi_1}(x_1 \ldots x_n),$
$(\forall x_1 \ldots x_n NP_{\forall x \phi_1}(x_1 \ldots x_n) \rightleftharpoons^p (\forall x NP_{\phi_1}[x_1 \ldots x_n, x]) \wedge D_{\phi_1} >$
avec $< NP_{\phi_1}[x_1 \ldots x_n, x], D_{\phi_1} >= Rn(\phi_1, p).$

$Rn(\exists x \phi_1, p) =$ $< NP_{\exists x \phi_1}(x_1 \ldots x_n),$
$(\forall x_1 \ldots x_n NP_{\exists x \phi_1}(x_1 \ldots x_n) \rightleftharpoons^p (\exists x NP_{\phi_1}[x_1 \ldots x_n, x])) \wedge D_{\phi_1} >$
avec $< NP_{\phi_1}[x_1 \ldots x_n, x], D_{\phi_1} >= Rn(\phi_1, p)$

$\diamond$

Historiquement, l'utilisation du renommage comme moyen de préserver la struc-

ture d'une formule avant de procéder à une mise sous forme clausale a été introduite dans [Tse68] dans le cadre du calcul propositionnel: la définition $Def(\psi, \phi)$ du littéral de Skolem comporte alors toujours une équivalence, quelle que soit la polarité de ψ dans ϕ. Une transformation similaire est utilisée dans le cadre de la logique du premier ordre dans [GNOP82]. La définition de $Def(\psi, \phi)$ en fonction de la polarité de ψ dans ϕ semble apparaître pour la première fois dans [Ede84]. De plus, la transformation décrite dans cet article évite de renommer les sous-formules atomiques de ϕ. Une étude détaillée d'une mise sous forme clausale utilisant le renommage est présentée dans [PG86]. Cette transformation, appelée "structure preserving" ne diffère de la transformation présentée ici qu'en ce que les formules atomiques et les sous-formules dont le connectif principal est une négation ne sont pas renommées. Cet article contient aussi des expérimentations qui cherchent à évaluer l'intérêt pratique de cette transformation en mesurant sur quelques exemples si l'utilisation du renommage permet de faciliter la recherche par Résolution d'une preuve de l'insatisfaisabilité d'un ensemble de clauses. Enfin, le renommage est utilisé dans [BdlT90] pour définir une mise sous forme clausale qui minimise le nombre de clauses. Une comparaison de ces deux dernières transformations est réalisée dans [BdlTC90a], [BdlTC90b].

Nous allons évaluer la taille (mesurée en nombre de symboles) de $Rn(\phi)$ ainsi que le temps nécessaire au calcul de $Rn(\phi)$. Puis nous bornerons la taille et le temps de calcul de la forme clausale associée à $Rn(\phi)$. Nos résultats sont du même ordre de grandeur que ceux de [BdlT89] et [PG86] quoique un peu plus précis.

Notons $|\phi|$ le nombre de symboles d'une formule ϕ. Nous obtenons les résultats suivants:

Théorème 3 *Soit ϕ une formule du premier ordre et soit N le nombre maximal de variables libres d'une sous-formule de ϕ. Alors $|Rn(\phi)|$ est $O(N|\phi|)$ et il existe un algorithme qui calcule $Rn(\phi)$ en $O(N(\log N)|\phi|)$ étapes.*

Preuve: Le renommage d'une sous-formule ψ de ϕ conduit à la création d'un symbole de prédicat de Skolem d'arité au plus N, donc le littéral de Skolem correspondant comporte au plus $N+1$ symboles. Ce littéral apparaît une fois dans $\phi[O \leftarrow NP_\psi(x_1 \ldots x_n)]$ et une fois dans $Def(\psi, \phi)$. Dans ce dernier cas, il est précédé d'une quantification universelle portant sur au plus N variables. De plus, on lui adjoint soit un connectif $\leftrightarrow$, soit un connectif $\rightarrow$ dans la définition de $NP_\psi(x_1 \ldots x_n)$. Enfin, un connectif $\wedge$ apparaît dans $\phi[O \leftarrow NP_\psi(x_1 \ldots x_n)] \wedge Def(\psi, \phi)$. Le renommage d'une sous-formule ψ de ϕ introduit donc au plus $3(N+1)+2$ symboles.

Soit $SF(\phi)$ le multi-ensemble des sous-formules de ϕ et notons $|SF(\phi)|$ sa cardinalité. Comme toutes les sous-formules de ϕ sont renommées, on a:

$$|Rn(\phi)| \leq (3N+5)|SF(\phi)| + |\phi|.$$

Comme $|SF(\phi)| \leq |\phi|$, $|Rn(\phi)|$ est $O(N.|\phi|)$.

De la définition inductive de $Rn(\phi, p)$ et en supposant que l'on représente la formule ϕ par un arbre étiqueté, on déduit immédiatement un algorithme récursif qui calcule $Rn(\phi)$ en un seul parcours de ϕ, des feuilles vers la racine de ϕ. La création de $Def(\psi, \phi)$ se fait en temps $O(N)$ et, en supposant que le remplacement de ψ dans ϕ par $NP_\psi(x_1 \ldots x_n)$ se fait par substitution physique sur l'arbre représentant ϕ, cette opération se fait en temps constant.

Ceci suppose que l'on dispose des variables libres de la sous-formule renommée. On peut calculer celles-ci en utilisant les structures de données décrites dans [Tar75], qui permettent d'effectuer la fusion d'ensembles disjoints en temps constant et de tester l'appartenance d'un élément à un ensemble de taille N en $O(\log N)$. Le coût de l'union de deux ensembles E_1, E_2 de cardinalité au plus N est alors au plus $N(\log N)$ si pour chaque élément x de E_1, on vérifie que x n'appartient pas à E_2 avant d'effectuer l'union de E_2 et de $\{x\}$. Le calcul des variables libres d'un atome A est alors au plus en $O(N(\log N)|A|)$ et le calcul des variables libres d'une sous-formule non-atomique s'effectue au plus en $O(N(\log N))$ en supposant données les variables libres de chacune de ses sous-formules. D'où le temps de calcul de $Rn(\phi)$ est au plus $O(N(\log N)|\phi|)$. C.Q.F.D.

Remarque 1 *Dans [PG86], le temps de calcul des variables libres n'est pas considéré. En négligeant celui-ci, on obtient alors une complexité en temps de $O(N|\phi|)$.*

Dans le cas d'une formule propositionnelle, $|Rn(\phi)|$ est en $O(|\phi|)$ et le calcul de $Rn(\phi)$ s'effectue en $O(|\phi|)$.

Etudions désormais la mise sous-forme clausale de $Rn(\phi)$ dont le résultat sera noté $CRn(\phi)$. Comme $Rn(\phi) = \forall x_1 \ldots x_n NP_\phi(x_1 \ldots x_n) \wedge D_\phi$ où D_ϕ est la conjonction des définitions issues du renommage des sous-formules de ϕ, la mise sous forme clausale de ϕ se ramène à la mise sous forme clausale de chacun des conjoints de D_ϕ.

Nous obtenons les résultats de complexité en temps et en espace suivant:

Théorème 4 *Soit ϕ une formule du premier ordre et soit N le nombre maximal de variables libres d'une sous-formule de ϕ. Alors $|CRn(\phi)|$ est $O(N|\phi|)$ et chaque*

clause de $CRn(\phi)$ comporte au plus $p+1$ littéraux, où soit $p=1$ si ϕ ne comporte pas de connectif et soit p est égal à l'arité maximale des connectifs de ϕ. De plus le temps de calcul de $CRn(\phi)$ est au plus en $O(N(\log N)|\phi|)$.

Preuve: Afin de mettre $Rn(\phi)$ sous forme clausale, nous appliquons chacune des transformations décrites précédemment à chacun des conjoints de $Rn(\phi)$ et l'on borne la taille du conjoint obtenu après transformation. On obtient alors pour chaque transformation une borne de la taille de la transformée de $Rn(\phi)$ en considérant la somme des bornes de la taille de la transformée de chaque conjoint. Pour un transformation Tr donnée dont le résultat est une conjonction de formules, nous utiliserons la notation $C \in Tr(\phi)$ pour indiquer que la formule C est un élément du multi-ensemble des conjoints de $Tr(\phi)$.

- Mise sous forme normale négative. Remarquons que le nombre d'occurrences du connectif $\leftrightarrow$ dans un conjoint de $Rn(\phi)$ est au plus de deux. (Ceci est le cas lorsqu'une formule $\phi_1 \leftrightarrow \phi_2$ de polarité nulle est renommée.) D'autre part, notons $FNN(\phi)$ le résultat de la mise sous forme normale négative de ϕ et définissons $\| \phi \| = Max(|FNN(\phi)|, |FNN(\neg\phi)|)$. Rappelons que $SFA(\phi)$ désigne le multi-ensemble des formules atomiques de ϕ et que l'on note $|SFA(\phi)|$ sa cardinalité. On montre alors facilement les inégalités suivantes:
  - $\| \phi \| \leq |\phi| + |SFA(\phi)|$ si ϕ est une formule linéaire.
  - $\| \phi_1 \leftrightarrow \phi_2 \| \leq 2(\| \phi_1 \| + \| \phi_2 \|) + 2$
  - $\| C \| \leq 2^k(|C| + |SFA(C)|)$ où C est un conjoint de $Rn(\phi)$ contenant k connectifs $\leftrightarrow$ $(O \leq k \leq 2)$.

  On en déduit alors les inégalités suivantes:

$$\begin{aligned} |FNN(Rn(\phi))| &\leq \sum_{C \in Rn(\phi)} \| C \| \\ &\leq 4\left[|Rn(\phi)| + \sum_{C \in Rn(\phi)} |SFA(C)|\right] \\ &\leq 4\,[|Rn(\phi)| + |SFA(Rn(\phi))|] \end{aligned}$$

  Or, on montre que:

$$|SFA(Rn(\phi))| \leq |SFA(\phi)| + 2|SF(\phi)| \leq 3|SF(\phi)|$$

$$|Rn(\phi)| \leq (3N+5)|SF(\phi)| + |\phi|$$

d'où l'on déduit:

$$|FNN(Rn(\phi))| \leq (12N + 32)|SF(\phi)| + 4|\phi|$$

Le nombre de connectifs $\leftrightarrow$ étant au plus de deux dans chaque conjoint C de $Rn(\phi)$, la mise sous forme normale négative de $Rn(\phi)$ se fait en temps $O(Rn(\phi))$.

- L'arité des fonctions de Skolem qui seront introduites à l'étape suivante ne peut être préalablement réduite par la minimisation de la portée des quantificateurs. Nous omettons donc cette étape désormais inutile.

- Par construction de $Rn(\phi)$, il n'existe pas de conjoint C comportant des variables dont le nom est partagé par plusieurs quantificateurs. Il n'y a donc pas de renommage de variables à effectuer.

- La skolemisation des formules

$$\forall x_1 \ldots x_n(\neg NP_{\exists x \phi_1}(x_1 \ldots x_n) \vee (\exists x NP_{\phi_1}[x_1 \ldots x_n, x]))$$

$$\forall x_1 \ldots x_n((\exists x \neg NP_{\phi_1}[x_1 \ldots x_n, x]) \vee NP_{\forall x \phi_1}(x_1 \ldots x_n))$$

  provenant de la transformation du conjoint correspondant au renommage d'une sous-formule de polarité positive ou nulle (resp. négative ou nulle) $\exists x \phi_1$ (resp. $\forall x \phi_1$) introduit au plus $(N+1)|SF(\phi)|$ symboles car:

  - il existe exactement une seule occurrence de x dans chaque conjoint qui est substituée par le terme $f_x(x_1 \ldots x_n)$ de longueur au plus $N+1$;
  - il existe au plus $|SF(\phi)|$ conjoints susceptibles d'être skolemisés.

  Si l'on note $Skolem(\phi)$ le résultat de la skolemisation d'une formule linéaire ϕ, on obtient alors:

$$\begin{aligned} |Skolem \circ FNN \circ Rn(\phi)| &\leq |FNN \circ Rn(\phi)| + (N+1)|SF(\phi)| \\ &\leq 4|\phi| + (13N + 33)|SF(\phi)| \end{aligned}$$

- Chaque conjoint est alors sous forme prénexe.

- Considérons pour finir l'application des lois de distributivité. Nous noterons $Dist(\phi)$ le résultat de l'application des lois de distributivité sur une formule ϕ. Ces lois sont appliquées sur les formules suivantes:

- $\forall x_1 \ldots x_n \neg NP_{\bigwedge_{i=1}^{p} \phi_i}(x_1 \ldots x_n) \vee (\bigwedge_{i=1}^{p} NP_{\phi_i}[x_1 \ldots x_n])$ provenant de la transformation du conjoint issu du renommage d'une sous formule $\bigwedge_{i=1}^{p} \phi_i$ de polarité positive ou nulle;
- $\forall x_1 \ldots x_n (\bigvee_{i=1}^{p} \neg NP_{\phi_i}[x_1 \ldots x_n]) \vee NP_{\bigvee_{i=1}^{p} \phi_i}(x_1 \ldots x_n)$ provenant de la transformation du conjoint issu du renommage d'une sous formule $\bigvee_{i=1}^{p} \phi_i$ de polarité négative ou nulle;
- $\forall x_1 \ldots x_n (NP_{\phi_1}[x_1 \ldots x_n] \wedge \neg NP_{\phi_2}[x_1 \ldots x_n]) \vee NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n)$ provenant de la transformation du conjoint issu du renommage d'une sous formule $\phi_1 \rightarrow \phi_2$ de polarité négative ou nulle;
- $\forall x_1 \ldots x_n \neg NP_{\phi_1 \leftrightarrow \phi_2}(x_1 \ldots x_n) \vee ((\neg NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_2}[x_1 \ldots x_n]) \wedge (\neg NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1}[x_1 \ldots x_n]))$ provenant de la transformation du conjoint issu du renommage d'une sous formule $\phi_1 \leftrightarrow \phi_2$ de polarité positive ou nulle;
- $\forall x_1 \ldots x_n ((\neg NP_{\phi_1}[x_1 \ldots x_n] \vee \neg NP_{\phi_2}[x_1 \ldots x_n]) \wedge (NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1}[x_1 \ldots x_n])) \vee NP_{\phi_1 \leftrightarrow \phi_2}(x_1 \ldots x_n)$ provenant de la transformation du conjoint issu du renommage d'une sous formule $\phi_1 \leftrightarrow \phi_2$ de polarité négative ou nulle.

Toutes ces formules sont de la forme: $(\forall x_1 \ldots x_n L \vee (\bigwedge_{i=1}^{p} \phi_i))$ où L désigne un littéral de Skolem. On a pour de tel conjoint C:

$$\begin{aligned} |Dist(C)| &= |C| + (p-1)(|L|+1) \\ &\leq |C| + (p-1)(N+2) \end{aligned}$$

Pour de telles formules C, appelons le facteur de duplication de C la valeur $d(C) = (p-1)$ et posons $d(C) = 0$ si les lois de distributivité ne peuvent être appliquées sur le conjoint C. On a alors:

$$\sum_{C \in (Skolem \circ FNN \circ Rn(\phi))} d(C) \leq |SF(\phi)|$$

D'où

$$\begin{aligned} |Dist \circ Skolem \circ FNN \circ Rn(\phi)| &\leq |Skolem \circ FNN \circ RN(\phi)| + \\ &\quad (N+2) \sum_{C \in (Skolem \circ FNN \circ Rn(\phi))} d(C) \\ &\leq |Skolem \circ FNN \circ Rn(\phi)| + (N+2)|SF(\phi)| \\ &\leq 4|\phi| + (14N+35)|SF(\phi)| \end{aligned}$$

- Il reste à transformer la formule ainsi obtenue en une conjonction de formules prénexes dont la matrice est une clause. Les seuls conjoints qui ne sont pas de cette forme sont ceux sur lesquels les lois de distributivité ont été appliquées. Rappelons que ces conjoints sont désormais de la forme $\forall x_1 \ldots x_n \bigwedge_{i=1}^{p}(L \vee \phi_i)$ où ϕ_i désigne une clause. On veut donc les transformer en: $\bigwedge_{i=1}^{p} \forall x_1 \ldots x_n(L \vee \phi_i)$. En utilisant encore le fait que $d(C) = p - 1$ et que $\sum_{C \in (Skolem o FNN o Rn(\phi))} d(C) \leq |SF(\phi)|$ et en constatant que $|\forall x_1 \ldots x_n| \leq 2N$, on montre immédiatement que cette opération rajoute au plus $2N|SF(\phi)|$ à la formule $Dist \circ Skolem \circ FNN \circ Rn(\phi)$ et donc

$$CRn(\phi) \leq |\phi| + (16N + 35)|SF(\phi)|$$

On a bien: $|CRn(\phi)|$ est en $O(N|\phi|)$; le calcul de $CRn(\phi)$ s'effectue en $O(|Rn(\phi)|)$ à partir de $Rn(\phi)$ et donc en $O(N(\log N)|\phi|)$ à partir de ϕ.

Quant au résultat sur le nombre de littéraux par clause, il s'obtient immédiatement en considérant $CRn(\phi)$. C.Q.F.D.

La mise sous forme clausale utilisant le renommage permet donc d'obtenir, dans le pire des cas, une forme clausale de taille polynômiale en temps polynômial.

Remarque 2 *[PG86] Il est possible de dériver par Résolution à partir de $CRn(\phi)$ les clauses composant la forme clausale standard de ϕ en résolvant les clauses de $CRn(\phi)$ sur leurs littéraux de Skolem.*

Par exemple, considérons la formule déjà citée: $\phi = \bigvee_{i=1}^{n}(A_i \wedge B_i)$. Soit P, (resp. P_i) le littéral de Skolem associé à ϕ (resp. à $(A_i \wedge B_i)$).

$$CRn(\phi) = P \wedge (\neg P \vee \bigvee_{i=1}^{n} P_i) \wedge$$
$$\bigwedge_{i=1}^{n}(\neg P_i \vee P_{A_i}) \wedge \bigwedge_{i=1}^{n}(\neg P_i \vee P_{B_i}) \wedge \bigwedge_{i=1}^{n}(\neg P_{A_i} \vee A_i) \wedge \bigwedge_{i=1}^{n}(\neg P_{B_i} \vee B_i)$$

En résolvant sur P, on obtient d'abord:

$$\bigvee_{i=1}^{n} P_i$$

puis, en résolvant sur P_1, P_{A_1}, P_{B_1}, on obtient

$$(A_1 \vee \bigvee_{i=2}^{n} P_i) \qquad (B_1 \vee \bigvee_{i=2}^{n} P_i)$$

puis, en résolvant sur P_2, P_{A_2}, P_{B_2}, on obtient

$$(A_1 \vee A_2 \vee \bigvee_{i=3}^{n} P_i) \qquad (A_1 \vee B_2 \vee \bigvee_{i=3}^{n} P_i)$$

$$(B_1 \vee A_2 \vee \bigvee_{i=3}^{n} P_i) \qquad (B_1 \vee B_2 \vee \bigvee_{i=3}^{n} P_i)$$

et ainsi de suite jusqu'à l'obtention de 2^n clauses.

Chapitre 2

Le principe de Résolution

Dans ce chapitre, nous donnons une présentation du principe de Résolution légèrement différente des présentations habituelles. Le formalisme introduit facilitera l'étude des traductions de preuves entre Résolution et Déduction Naturelle.

En particulier, nous nous intéresserons ultérieurement (chapitre 6) au problème de la traduction d'une réfutation par Résolution de $CRn(\neg\phi)$ en une preuve en Déduction Naturelle de la validité de ϕ. Nous montrerons qu'une telle traduction peut être réalisée très simplement pour certaines classes de dérivations : les dérivations par Résolution Sémantique Ordonnée. Afin de pouvoir extraire une preuve en Déduction Naturelle de *toute* preuve par Résolution, nous tirerons parti de l'existence de *procédures de normalisation* qui, étant donnée une réfutation par Résolution, permettent de construire une réfutation vérifiant les restrictions de la Résolution Sémantique Ordonnée. Nous décrirons en détail cette transformation en nous inspirant des résultats de [Bac86].

Bien entendu, il ne s'agit pas ici d'apporter une contribution à l'étude des stratégies de Résolution. (Nos résultats sont plus faibles que ceux décrits dans [Bac86], [Rus87].) Il s'agit de donner une description précise d'un procédé qui

sera utilisé non pas comme une technique servant à établir la complétude d'une stratégie mais comme la spécification d'un algorithme de pré-traitement utile à la présentation d'une preuve par Résolution en preuve par Déduction Naturelle.

2.1 Notations

La Résolution [Rob65] est un calcul des prédicats conçu pour la mécanisation de la logique du premier ordre et qui s'applique aux formules sous forme clausale.

On considère généralement une clause comme une disjonction de littéraux *deux à deux distincts.* Le plus souvent, l'ordre dans lequel les littéraux apparaissent dans une clause n'importe pas et l'on considère l'égalité entre clauses modulo les lois de commutativité et d'associativité de $\vee$. Ceci revient alors à considérer une clause comme un *ensemble* de littéraux. Les règles d'inférence utilisées sont alors soit la Résolution [Rob65], soit la résolution binaire et la factorisation [CL73], [Lov78].

Afin de décrire simplement les traductions entre preuves par Résolution et preuves dans un système de Déduction Naturelle, il est utile de considérer une autre présentation du principe de Résolution. Nous considèrerons désormais une clause comme une disjonction de littéraux non nécessairement deux à deux distincts; l'ordre dans lequel les littéraux apparaissent dans une clause n'a pas d'importance. Ceci revient à considérer une clause $\bigvee_{i=1}^{n} L_i$ comme un multi-ensemble de littéraux $\{L_1 \ldots L_n\}$. Nous noterons alors $C_1 \vee C_2$ la clause correspondant à la réunion des multi-ensembles de littéraux correspondant à C_1, C_2 et $C_1 \vee L$ la clause correspondant à la réunion des multi-ensembles C_1 et $\{L\}$, C_1, C_2 pouvant éventuellement désigner des multi-ensembles vides de littéraux. De même, étant données une substitution σ et une clause $C = \bigvee_{i=1}^{n} L_i$, $\sigma(C)$ désigne la clause $\bigvee_{i=1}^{n} \sigma(L_i)$.

Considérons les règles d'inférence suivantes:

- *instanciation:* $\dfrac{C}{\sigma(C)}$,
  où C est une clause et σ est une substitution;
- *contraction:* $\dfrac{(C \vee L \vee L)}{(C \vee L)}$,
  où C est une clause et L est un littéral;
- *coupure:* $\dfrac{(C_1 \vee L) \quad (C_2 \vee L^C)}{(C_1 \vee C_2)}$
  où C_1 et C_2 sont des clauses et L, L^c sont deux littéraux complémentaires.

On définit de manière usuelle pour ce nouveau calcul les notions suivantes:

Définition: Une *déduction* (ou une *dérivation*) d'une clause C par le Principe de Résolution à partir d'un ensemble S de clauses est une séquence finie de clauses $< C_1, \ldots, C_p, \ldots, C_q >$ telle que:

- $1 \leq i \leq p$, C_i est une clause de S;
- $C_q = C$;
- $p+1 \leq i \leq q$, C_i est une clause obtenue par une des règles d'instanciation, de contraction ou de coupure à partir des clauses précédant C_i dans la séquence.

Une réfutation par Résolution d'un ensemble S de clauses est une dérivation de la clause vide, notée $\Box$, par le Principe de Résolution à partir de S. On note $S \vdash \Box$ l'existence d'une telle dérivation. $\diamond$

Soit Γ (respectivement Δ) le multi-ensemble des atomes des littéraux positifs (respectivement négatifs) qui apparaissent dans une clause C. On s'autorise à noter C sous forme de *séquent* par: $\Gamma \Longrightarrow \Delta$. Avec ces notations, les règles d'inférence d'instanciation, de contraction et de coupure s'écrivent:

- instanciation: $\dfrac{\Gamma \Longrightarrow \Delta}{\sigma(\Gamma) \Longrightarrow \sigma(\Delta)}$ où σ est une substitution;
- contraction: $\dfrac{A, A, \Gamma \Longrightarrow \Delta}{A, \Gamma \Longrightarrow \Delta} \qquad \dfrac{\Gamma \Longrightarrow \Delta, A, A}{\Gamma \Longrightarrow \Delta, A}$
- coupure: $\dfrac{\Gamma_1 \Longrightarrow \Delta_1, A \quad A, \Gamma_2 \Longrightarrow \Delta_2}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}$

Il apparaît clairement que les règles d'inférence

- *factorisation:* $\dfrac{(C \vee L_1 \vee L_2)}{\sigma(C \vee L_1)}$,
  où C est une clause et L_1, L_2 sont littéraux de même polarité et d'unificateur le plus général σ;
- *résolution:* $\dfrac{(C_1 \vee L_1) \quad (C_2 \vee L_2)}{\sigma(C_1 \vee C_2)}$
  où C_1 et C_2 sont des clauses et L_1, L_2 sont deux littéraux complémentaires d'unificateur le plus général σ;

sont en fait des règles d'inférence dérivées des règles d'instanciation, de contraction et de coupure et il est aisé de traduire toute dérivation utilisant les règles de résolution et de factorisation en une dérivation utilisant les règles d'instanciation, de coupure et de contraction.

Remarque 3 *Par la suite, nous noterons C^* la clause ne contenant que des littéraux distincts, obtenue à partir d'une clause C par applications répétées de la règle de contraction.*

2.2 Stratégies de Résolution

En pratique, il est intéressant et le plus souvent indispensable de restreindre l'application des règles d'inférence du Principe de Résolution afin de limiter le nombre de clauses déductibles à partir d'un ensemble de clauses. Nous introduisons la terminologie suivante afin de pouvoir présenter deux restrictions parmi les plus connues: la Résolution Ordonnée et la Résolution Sémantique.

Soit S un ensemble de clauses et soit P l'ensemble des symboles de prédicats apparaissant dans les littéraux de S. De même, soit F l'ensemble des symboles de fonctions apparaissant dans les littéraux de S. Une *interprétation de Herbrand* des littéraux de S est un sous-ensemble de $A(P,F)$, l'ensemble des atomes clos construits sur P et sur F.

Soit $C = \Gamma \Longrightarrow \Delta$ une clause d'un ensemble S de clauses et soit I une interprétation des littéraux de S. Soit $\{x_1 \ldots x_n\}$ l'ensemble des variables libres de C.

- C est une clause *I-positive* si et seulement si pour toute substitution close σ telle que $Dom(\sigma) = \{x_1 \ldots x_n\}$, $\sigma(\Gamma) \not\subseteq I$ ou $\sigma(\Delta) \cap I \neq \emptyset$.
- C est une clause *I-négative* si et seulement si pour toute substitution close σ telle que $Dom(\sigma) = \{x_1 \ldots x_n\}$, $\sigma(\Gamma) \subseteq I$ et $\sigma(\Delta) \cap I = \emptyset$.

Une interprétation I des littéraux d'un ensemble S de clauses est une *partition* de S si et seulement si toute clause de S est soit I-positive, soit I-négative. Par exemple, $I = \emptyset$ et $I = A(P,F)$ sont des partitions de tout ensemble de clauses. Plus généralement, pour tout ensemble $P' \subseteq P$, $I = A(P',F)$ est aussi une partition de l'ensemble de clauses considéré.

Etant donnée I une partition d'un ensemble S de clauses, on appelle *coupure sémantique utilisant I* (ou I-coupure) la restriction de la règle de coupure qui impose que, parmi les deux prémisses de l'inférence, l'une soit une clause I-positive et l'autre soit une clause I-négative.

Un ordre partiel sur un ensemble E est une relation binaire, irréflexive et transitive entre les éléments de E. On le note $\succ$. Cet ordre est *bien fondé* si et seulement si

il n'existe pas de séquence infinie d'éléments de E telle que $e_1 \succ e_2 \succ \ldots$. Pour tout ensemble E_1 inclus dans E, on définit $Max(E_1) = \{x/x \in E_1 \text{ et } \forall y \in E_1, \neg(y \succ x)\}$. Les éléments de $Max(E_1)$ sont appelés les éléments *maximaux* de E_1. Un ordre partiel $\succ$ défini sur un ensemble d'atomes $A(P,F,V)$ est *stable par substitution* si et seulement si pour toute substitution σ et tout atome A_1, A_2 de $A(P,F,V)$, $A_1 \succ A_2$ implique $\sigma(A_1) \succ \sigma(A_2)$. Par la suite, nous appellerons *A-ordre* un ordre partiel bien fondé défini sur un ensemble d'atomes $A(P,F,V)$ et noté $\succ$ tel que $\succ$ est stable par substitution et $\succ$ est un ordre total sur $A(P,F)$.

Etant donné un A-ordre $\succ$, on appelle *coupure ordonnée* la restriction de la règle de coupure qui impose que dans une application de: $\frac{\Gamma_1 \Longrightarrow \Delta_1, A \quad A, \Gamma_2 \Longrightarrow \Delta_2}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}$ l'atome A soit un élément maximal de l'ensemble des atomes figurant dans $\Gamma_1, \Gamma_2, \Delta_1, \Delta_2$.

De même, étant donné un A-ordre $\succ$ et une partition I, on appelle *coupure sémantique ordonnée* la restriction de la règle de coupure sémantique qui impose que dans une application de $\frac{\Gamma_1 \Longrightarrow \Delta_1, A \quad A, \Gamma_2 \Longrightarrow \Delta_2}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}$ l'atome A soit un élément maximal de l'ensemble des atomes de Γ, Δ si l'on désigne par $\Gamma \Longrightarrow \Delta$ la clause I-négative de l'inférence.

On appellera alors *Résolution Ordonnée* la restriction de la Résolution qui, étant donné un A-ordre $\succ$, utilise comme règles d'inférence la coupure ordonnée, la contraction et la règle d'instanciation. De même, on appellera *Résolution Sémantique* la restriction de la Résolution qui, étant donnée une partition I, utilise comme règles d'inférence la coupure sémantique, la contraction et la règle d'instanciation. Enfin, on appellera *Résolution Sémantique Ordonnée* la restriction de la Résolution qui étant donnée une partition I et un A-ordre $\succ$, utilise comme règles d'inférence la coupure sémantique ordonnée, la contraction et la règle d'instanciation. En pratique, on cherche également à imposer des restrictions sur l'application de la règle de contraction [HK69], [Nol80]. Nous ne considèrerons pas ce type de restriction.

La Résolution Ordonnée, la Résolution Sémantique et la Résolution Sémantique Ordonnée sont des restrictions de la Résolution qui restent complètes pour la réfutation. Pour le montrer, on peut par exemple utiliser les techniques de preuve de complétude de raffinements de la Résolution développées dans [AB70], [HK69], [Rus87]. (Voir aussi [CL73] et [Lov78].) Une autre manière de montrer la complétude pour la réfutation de ces stratégies consiste à décrire une procédure de normalisation qui, étant donnée une réfutation par Résolution, construit une réfutation par Résolution dans laquelle toutes les applications des règles d'inférence

vérifient les restrictions de la stratégie considérée. De telles réfutations sont dites en *forme normale* (ou, plus simplement, normales).

C'est le point de vue adopté dans [Bac86] pour montrer la complétude des règles de résolution, paramodulation et factorisation orientées[1] en présence de règles de simplification et de subsomption.

2.3 Procédures de normalisation des dérivations

Par la suite, nous montrerons comment traduire certaines preuves par Résolution Sémantique Ordonnée en une preuve en Déduction Naturelle. Afin de pouvoir extraire une preuve en Déduction Naturelle de toute preuve par Résolution, il est donc nécessaire de disposer d'une procédure de normalisation des preuves par Résolution en preuves par Résolution Sémantique Ordonnée.

Dans [Bac86], la complétude de la Résolution Sémantique Ordonnée utilisant des ordres de simplification complets est obtenue en deux temps:

1. d'abord, on montre la complétude de cette stratégie pour le cas clos et ce, en appliquant une procédure de normalisation. Pour ce faire, Bachmair utilise un ordre $\succ_{\mathcal{R}}$ bien fondé sur les réfutations et donne un ensemble de règles de transformation applicables sur des preuves non normales et qui transforment toute réfutation $\mathcal{R}$ en une réfutation $\mathcal{R}'$ telle que $\mathcal{R} \succ_{\mathcal{R}} \mathcal{R}'$. La bonne fondaison l'ordre $\succ_{\mathcal{R}}$ assure la terminaison du processus de normalisation. La preuve ainsi obtenue est alors nécessairement normale.

2. puis, en utilisant un lemme de relèvement, on généralise le résultat au premier ordre.

Afin de pouvoir réutiliser les techniques décrites dans cet article, nous procéderons de manière inverse:

[1] Afin de restreindre l'application de la règle de paramodulation, les ordres sur les atomes vérifient d'autres propriétés en plus de celles mentionnées pour les A-ordres. En particulier, les ordres $\succ$ considérés sont définis sur $T(F,V) \cup A(P,F,V)$ et sont supposés totaux sur $T(F) \cup A(P,F)$. Ils vérifient en outre les propriétés suivantes pour tout terme s, t, u, v et tout atome A:

- si $t \succ s$ alors $u[t] \succ u[s]$ et $A[t] \succ A[s]$;
- $A[s] \succ (s = t)$ si $s \succ t$ et A n'est pas une équation;
- $(u[s] = v) \succ (s = t)$ si $s \succ t$ et $s \neq u[s]$;
- $(s = t) \succ (s = u)$ si $t \succ u$.

De tels ordres sont appelés *ordres de simplification complets.*

1. Etant donnée une preuve par Résolution, nous obtiendrons à partir de celle-ci une dérivation quasi-propositionnelle appelée *CC-dérivation.* Le lemme 1, qui décrit cette transformation, est en fait une sorte d'inverse des lemmes de relèvement utilisés dans les preuves de complétude des stratégies de Résolution.

2. Puis, nous appliquerons une procédure de normalisation pour obtenir une dérivation satisfaisant à certaines conditions. Nous appliquerons les techniques présentées dans [Bac86] à d'autres restrictions de la Résolution que celle décrite dans cet article. En particulier, nous montrerons comment obtenir des dérivations par Résolution Sémantique, Résolution Sémantique Ordonnée ainsi que des dérivations sans tautologie dans lesquelles la règle de contraction est appliquée dès que possible.

Définition: Une dérivation $< C_1, \ldots, C_p, \ldots, C_q >$ est une dérivation *arborescente* si et seulement si toute clause $C_i, 1 \leq i \leq q-1$ est utilisée exactement une fois comme prémisse d'une règle d'inférence. $\diamond$

Définition: Une *CC-dérivation* d'une clause C à partir d'un ensemble S de clauses est une dérivation par Résolution de C à partir de S dans laquelle les seules règles d'inférence utilisées sont la coupure et la contraction. $\diamond$

Définition: Une CC-dérivation *générée* par un ensemble S de clauses est une CC-dérivation à partir d'un ensemble S' de clauses telle que pour toute clause C' de S' il existe une substitution σ et une clause C de S telle que $C' = \sigma(C)$. $\diamond$

Lemme 1 *Toute dérivation par Résolution d'une clause C à partir d'un ensemble S de clauses peut être transformée en une CC-dérivation arborescente de C générée par S.*

Preuve: On montre par induction sur le nombre q la propriété $\mathcal{P}(q)$: "Toute dérivation $< C_1, \ldots, C_p, \ldots, C_q >$ par résolution d'une clause $C_q = C$ à partir d'un ensemble de clauses S peut être transformée en une CC-dérivation de C générée par S".

- $\mathcal{P}(0)$. Alors $C \in S$ et $< C >$ est une CC-dérivation de C générée par S.
- Supposons que, $\forall 0 < n < q, \mathcal{P}(n)$ est vérifiée et montrons $\mathcal{P}(q)$. Alors, C est obtenue par application d'une règle d'inférence et l'on considère chacune des possibilités suivantes:

- $C = (C' \vee C'')$ est obtenue par résolution: $\frac{(C' \vee L) \quad (C'' \vee L^c)}{(C' \vee C'')}$
  Posons $C_{q_1} = (C' \vee L)$ et $C_{q_2} = (C'' \vee L^c)$. Par hypothèse d'induction, on peut construire à partir de $< C_1, \ldots, C_p, \ldots, C_{q_1} >$ une CC-dérivation arborescente de $(C' \vee L)$ générée par S. Soit $< C_1^1, \ldots, C_{p_1}^1, \ldots, C_{r_1}^1 >$ cette dérivation. De même, soit $< C_1^2, \ldots, C_{p_2}^2, \ldots, C_{r_2}^2 >$ une CC-dérivation arborescente de $(C'' \vee L^c)$ générée par S.
  $< C_1^1, \ldots, C_{p_1}^1, C_1^2, \ldots, C_{p_2}^2, C_{p_1+1}^1, \ldots, C_{r_1}^1, C_{p_2+1}^2, \ldots, C_{r_2}^2, C >$ est alors une CC-dérivation arborescente de C générée par S.
- $C = (C' \vee L)$ est obtenue par contraction: $\frac{(C' \vee L \vee L)}{(C' \vee L)}$
  Posons $C_{q_1} = (C' \vee L \vee L)$. Par hypothèse d'induction, on peut construire à partir de $< C_1, \ldots, C_p, \ldots, C_{q_1} >$ une CC-dérivation arborescente de $(C' \vee L \vee L)$ générée par S. Soit $< C_1^1, \ldots, C_{p_1}^1, \ldots, C_{r_1}^1 >$ cette dérivation. Alors $< C_1^1, \ldots, C_{p_1}^1, \ldots, C_{r_1}^1, (C' \vee L) >$ est la CC-dérivation arborescente recherchée.
- $C = \sigma(C')$ est obtenue par instanciation $\frac{C'}{\sigma(C')}$
  Posons $C_{q_1} = C'$ On distingue alors les cas suivants:
  * C_{q_1} est une clause de S. Alors $< \sigma(C') >$ est une CC-dérivation arborescente de C générée par S.
  * $C_{q_1} = (C'' \vee L)$ est obtenue par contraction: $\frac{(C'' \vee L \vee L)}{(C'' \vee L)}$.
    Supposons que dans la dérivation de C, $C_{q_2} = (C'' \vee L \vee L)$. Par hypothèse d'induction appliquée à la dérivation: $< C_1, \ldots, C_p, \ldots, C_{q_2}, \sigma(C'' \vee L \vee L) >$, on peut construire une CC-dérivation de $\sigma(C'' \vee L \vee L)$ générée par S: $< C_1^1, \ldots, C_{p_1}^1, \ldots, C_{r_1}^1 >$ à partir de laquelle on construit une CC-dérivation arborescente de C générée par S: $< C_1^1, \ldots, C_{p_1}^1, \ldots, C_{r_1}^1, \sigma(C'' \vee L) >$ par application de la règle de contraction sur la clause $C_{r_1}^1 = \sigma(C'' \vee L \vee L)$.
  * C_{q_1} est obtenue par coupure. Soient $C_{q_2} = (C'' \vee L)$ et $C_{q_3} = (C''' \vee L^c)$ les clauses sur lesquelles la règle de coupure est appliquée pour obtenir C_{q_1}. En utilisant l'hypothèse d'induction sur les dérivations $< C_1, \ldots, C_p, \ldots, C_{q_2}, \sigma(C'' \vee L) >$ et $< C_1, \ldots, C_p, \ldots, C_{q_3}, \sigma(C''' \vee L^c) >$, on peut construire deux CC-dérivations générées par S de $\sigma(C'' \vee L)$ et de $\sigma(C''' \vee L^c)$ respectivement. Soient $< C_i^i, \ldots, C_{p_i}^i, \ldots, C_{r_i}^i >, i = 1, 2$ ces dérivations. Alors la dérivation $< C_1^1, \ldots, C_{p_1}^1, C_1^2, \ldots, C_{p_2}^2, C_{p_1+1}^1, \ldots, C_{r_1}^1, C_{p_2+1}^2, \ldots, C_{r_2}^2, C >$ où C

est obtenue par application de la règle de coupure sur les clauses $C^1_{r_1} = \sigma(C'' \vee L)$ et $C^2_{r_2} = \sigma(C''' \vee L^c)$ est la CC-dérivation de C recherchée.

* $C_{q_1} = \sigma'(C'')$ est obtenue par instanciation. Supposons que $C_{q_1} = C''$ dans la dérivation de C. On applique alors l'hypothèse d'induction pour construire une CC-dérivation à partir de la dérivation: $< C_1, \ldots, C_p, \ldots, C_{q_2}, (\sigma \circ \sigma')(C_{q_2}) >$ pour obtenir le résultat voulu.

C.Q.F.D.

Remarque 4 *Cette transformation peut produire une preuve dont le nombre d'inférences est exponentiellement plus grand que celui de la preuve initiale.*

Le lemme suivant, résultat essentiellement technique, nous sera d'une grande utilité:

Lemme 2 *Soit $\mathcal{D}$ une dérivation arborescente par Résolution d'une clause C à partir d'un ensemble $S \cup \{C_1\}$ de clauses et soit $C_2 \subseteq C_1$. Alors il existe une dérivation arborescente $\mathcal{D}'$ d'une clause $C' \subseteq C$ à partir de $S \cup \{C_2\}$ telle que $\mathcal{D}'$ comporte moins de clauses que $\mathcal{D}$.*

Preuve: Par induction sur la longueur N de la dérivation $\mathcal{D}$.

- $N = 1$. Si $C = C_1$ alors $\mathcal{D} =< C_1 >$ et comme $C_2 \subseteq C_1$ $\mathcal{D}' =< C_2 >$ est la dérivation recherchée. Sinon $\mathcal{D}$ satisfait aux conditions énoncées dans le lemme.

- Supposons le lemme vérifié pour toute dérivation de longueur $0 < n < N$ et considérons une dérivation de longueur N. Alors C est obtenue par application d'une règle d'inférence. Posons $\mathcal{D} = \mathcal{D}_1. < C >$ et considérons l'un des trois cas suivants:

  - instanciation: $C = \sigma(C_3)$ et $\frac{C_3}{\sigma(C_3)}$.
    Par hypothèse d'induction appliquée à la dérivation $\mathcal{D}_3$ de C_3, il existe une dérivation $\mathcal{D}'_3$ de $C'_3 \subseteq C_3$ à partir de $S \cup \{C_2\}$ telle que $\mathcal{D}'_3$ comporte moins de clauses que $\mathcal{D}_3$.
    Soient $C' = \sigma(C'_3)$, $\mathcal{D}' = \mathcal{D}'_3. < C' >$. Alors $\mathcal{D}'$ est une déduction de C' qui satisfait aux conditions énoncées dans le lemme.

- contraction: $C = (C_3 \vee L)$ et $\frac{C_3 \vee L \vee L}{C_3 \vee L}$.
  Par hypothèse d'induction appliquée à la dérivation $\mathcal{D}_3$ de $C_3 \vee L \vee L$, il existe une dérivation $\mathcal{D}'_3$ de $C'_3 \subseteq (C_3 \vee L \vee L)$ à partir de $S \cup \{C_2\}$ telle que $\mathcal{D}'_3$ comporte moins de clauses que $\mathcal{D}_3$.
  On considère alors deux cas:
  * $C'_3 \subseteq (C_3 \vee L)$.
    Alors la dérivation $\mathcal{D}'_3$ de C'_3 répond à la question.
  * $C'_3 = (C''_3 \vee L \vee L)$ et $C''_3 \subseteq C_3$.
    Soit $C' = (C''_3 \vee L)$ la clause obtenue par contraction sur C'_3. Alors $\mathcal{D}_1. < C' >$ est une dérivation de C' qui vérifie les conditions énoncées dans le lemme.
- coupure $C = (C_3 \vee C_4)$ et $\frac{C_3 \vee A \quad C_4 \vee \neg A}{C_3 \vee C_4}$.
  Par hypothèse d'induction appliquée à la dérivation $\mathcal{D}_3$ de $(C_3 \vee A)$, il existe une dérivation $\mathcal{D}'_3$ de $C'_3 \subseteq (C_3 \vee A)$ à partir de $S \cup \{C_2\}$ telle que $\mathcal{D}'_3$ comporte moins de clauses que $\mathcal{D}_3$.
  De même, il existe une dérivation $\mathcal{D}'_4$ de $C'_4 \subseteq (C_4 \vee \neg A)$ à partir de $S \cup \{C_2\}$ telle que $\mathcal{D}'_4$ comporte moins de clauses que $\mathcal{D}_4$.
  On considère alors les cas suivants:
  * $C'_3 \subseteq C_3$.
    Alors la dérivation $\mathcal{D}'_3$ de C'_3 répond à la question.
  * $C'_4 \subseteq C_4$.
    Le cas est similaire au précédent.
  * $C'_3 = (C''_3 \vee A)$ et $C''_3 \subseteq C_3$; $C'_4 = (C''_4 \vee \neg A)$ et $C''_4 \subseteq C_4$;
    Soit $C' = (C''_3 \vee C''_4)$ la clause obtenue par application de la règle de coupure sur les clauses C'_3 et C'_4. Alors la dérivation $\mathcal{D}'_3.\mathcal{D}'_4. < C' >$ vérifie les conditions énoncées dans le lemme.

C.Q.F.D.

Une application possible de ce lemme est par exemple:

Lemme 3 *Toute CC-dérivation arborescente d'une clause C peut être transformée en une CC-dérivation arborescente d'une clause $C' \subseteq C$ telle que*

- *aucune clause tautologique n'est utilisée comme prémisse d'une application de la règle de coupure;*

- *aucune clause contenant plusieurs occurrences d'un même littéral n'est utilisée comme prémisse d'une application de la règle de coupure.*

Preuve: Définissons le *poids d'une coupure* comme étant la somme du nombre d'occurrences des littéraux des prémisses de cette inférence et définissons le poids μ_D d'une dérivation comme étant la somme des poids des coupures apparaissant dans la dérivation. Alors l'ordre $\succ_{\mathcal{D}}$ défini sur les dérivations par: $\mathcal{D}_1 \succ_{\mathcal{D}} \mathcal{D}_2$ si et seulement si $\mu_{\mathcal{D}}(\mathcal{D}_1) > \mu_{\mathcal{D}}(\mathcal{D}_2)$ est un ordre bien fondé.

On montre alors immédiatement en utilisant le lemme 2 que:

> Si $\mathcal{D}_1$ est une CC-dérivation arborescente d'une clause C_1 à partir d'un ensemble S de clauses et s'il existe une CC-dérivation arborescente $\mathcal{D}_2$ d'une clause $C_2 \subseteq C_1$ à partir de S telle que $\mathcal{D}_1 \succ_{\mathcal{D}} \mathcal{D}_2$, alors pour toute dérivation $\mathcal{D}$ d'une clause C contenant la dérivation $\mathcal{D}_1$ il existe une dérivation $\mathcal{D}'$ d'une clause $C' \subseteq C$ telle que $\mathcal{D} \succ_{\mathcal{D}} \mathcal{D}'$.

Puisque $\succ_{\mathcal{D}}$ est bien fondé, il suffit de montrer que si $\mathcal{D}$ est une dérivation qui ne vérifie pas les propriétés énoncées dans le lemme, alors on peut construire une dérivation une dérivation $\mathcal{D}'$ telle que $\mathcal{D} \succ_{\mathcal{D}} \mathcal{D}'$. Une telle dérivation $\mathcal{D}$ contient nécessairement une dérivation $\mathcal{D}''$ de la forme:

$$\frac{\dfrac{\mathcal{D}_1''}{C_1 \vee L} \quad \dfrac{\mathcal{D}_2''}{C_2 \vee L^c}}{C_1 \vee C_2}$$

telle que $\dfrac{\mathcal{D}_1''}{C_1 \vee L}$ et $\dfrac{\mathcal{D}_2''}{C_2 \vee L^c}$ satisfont aux conditions du lemme mais la dernière inférence de $\mathcal{D}''$ ne les satisfait pas.

Considérons les deux règles de transformation:

$$\frac{\dfrac{\mathcal{D}_1}{C_1 \vee L} \quad \dfrac{\mathcal{D}_2}{C_2 \vee L^c}}{C_1 \vee C_2}$$

$$\longrightarrow$$

$$\frac{\dfrac{\dfrac{\mathcal{D}_1}{C_1 \vee L}}{\dots \text{plusieurs contractions} \dots} \quad \dfrac{\dfrac{\mathcal{D}_2}{C_2 \vee L^c}}{\dots \text{plusieurs contractions} \dots}}{(C_1 \vee C_2)^* - \{L, L^c\}}$$

$$\frac{\dfrac{\mathcal{D}_1}{C_1 \vee L} \quad \dfrac{\mathcal{D}_2}{C_2 \vee L \vee L^c}}{C_1 \vee C_2 \vee L}$$

$$\longrightarrow$$

$$\frac{\dfrac{\dfrac{\mathcal{D}_1}{C_1 \vee L}}{\dots \text{plusieurs contractions} \dots}}{(C_1 \vee L)^*}$$

Chacune de ces transformations transforme une CC-dérivation arborescente $\mathcal{D}''$ d'une clause C'' en une CC-dérivation arborescente $\mathcal{D}'''$ d'une clause $C''' \subseteq C''$ telle que $\mathcal{D}'' \succ_{\mathcal{D}} \mathcal{D}'''$. Par application du lemme intermédiaire mentionné dans la preuve, on peut alors construire la dérivation $\mathcal{D}'$ cherchée. C.Q.F.D.

L'existence de procédure de normalisation pour la Résolution Ordonnée, la Résolution Sémantique et la Résolution Sémantique Ordonnée se montre de manière analogue. On a:

Théorème 5 *Soit $\succ$ un A-ordre sur les atomes des littéraux d'un ensemble S de clauses et soit I une partition de S. Toute CC-réfutation arborescente à partir de S peut être transformée en une CC-réfutation arborescente à partir de S telle que:*

- *aucune clause tautologique n'est utilisée comme prémisse d'une application de la règle de coupure;*
- *aucune clause contenant plusieurs occurrences d'un même littéral n'est utilisée comme prémisse d'une application de la règle de coupure;*
- *toutes les applications de la règle de coupure sont des applications de la règle de coupure ordonnée (resp. coupure sémantique, coupure sémantique ordonnée).*

Preuve: On peut supposer que l'on considère des dérivations vérifiant les deux premiers points énoncés dans le lemme puisque l'on dispose du lemme précédent. Nous traiterons le problème de l'existence d'une procédure de normalisation de telles dérivations en dérivations par Résolution Ordonnée et en dérivations par Résolution Sémantique en même temps, les preuves étant analogues. Nous nous inspirons ici de [Bac86].

Nous définissons d'abord un ordre bien fondé sur les dérivations.

Considérons d'abord le cas de la coupure ordonnée. A chaque atome A apparaissant dans la réfutation, on associe un symbole fonctionnel f_A d'arité 2 et l'on définit un ordre $\succ_F$ sur ces symboles fonctionnels par:

$$f_{A_1} \succ_F f_{A_2} \text{ si et seulement si } A_1 \succ A_2$$

Soit $\dagger$ une constante que l'on rajoute à l'ensemble de symboles fonctionnels ainsi défini. A chaque CC-dérivation $\mathcal{D}$ à partir de S, on associe un terme construit comme suit:

- $\mathcal{D} =< C >$ et $C \in S$. Alors le terme associé à $\mathcal{D}$ est $\dagger$.

- $\mathcal{D} = \mathcal{D}_1. < \sigma(C) >$ où $\mathcal{D}_1$ est une dérivation de C. Soit t_1 le terme associé à $\mathcal{D}_1$. Alors t_1 est aussi le terme associé à $\mathcal{D}$. (Remarque: par définition d'une CC-dérivation $t_1 = \dagger$).

- $\mathcal{D} = \mathcal{D}_1. < (C \vee L) >$ où $(C \vee L)$ est une clause obtenue par contraction à partir de $(C \vee L \vee L)$ et $\mathcal{D}_1$ est une dérivation de $(C \vee L \vee L)$. Soit t_1 le terme associé à $\mathcal{D}_1$. Alors t_1 est aussi le terme associé à $\mathcal{D}$.

- $\mathcal{D}$ est une dérivation d'une clause $C_1 \vee C_2$ obtenue par application de la règle de coupure sur les clauses $(C_1 \vee A)$ et $(C_2 \vee \neg A)$. Soient t_1 et t_2 les termes associés aux dérivations respectives de ces clauses. Alors $f_A(t_1, t_2)$ est le terme associé à la dérivation $\mathcal{D}$.

On définit alors l'ordre $\succ_{\mathcal{D}}$ sur les dérivations par:

$$\mathcal{D}_1 \succ_{\mathcal{D}} \mathcal{D}_2 \text{ si et seulement si } t_1 \succ_F^{RPO} t_2$$

où t_1, t_2 sont les termes associés aux dérivations $\mathcal{D}_1, \mathcal{D}_2$ et $\succ_F^{RPO}$ désigne l'ordre RPO[2] défini sur l'ensemble des termes construits sur l'alphabet F ordonné par la relation $\succ_F$. $\succ_{\mathcal{D}}$ est un ordre bien fondé.

Considérons maintenant la cas de la Résolution Sémantique. Etant donnée une application de la règle de coupure de la forme $\frac{C_1 \quad C_2}{C_3}$ dans laquelle les clauses C_1, C_2 sont des clauses I-positives, nous désignerons par abus de notation $(C_1 \cup C_2) \cap I$ l'ensemble des littéraux L apparaissant dans les prémisses de cette inférence tels que l'atome du littéral L est A_L et soit $L = \neg A_L$ et $A_L \notin I$, soit $L = A_L$ et $A_L \in I$.

[2]Etant donné un ensemble F, un quasi-ordre $\succeq_F$ sur F est une relation binaire réflexive et transitive définie sur les éléments de F. On note alors $\succ_F$ l'ordre partiel défini sur F par

$$f_1 \succ_F f_2 \text{ si et seulement si } f_1 \succeq_F f_2 \text{ et non } f_2 \succeq_F f_1$$

pour tout élément f_1, f_2 de F et l'on note $\approx_F$ la relation d'équivalence définie sur F par

$$f_1 \approx_F f_2 \text{ si et seulement si } f_1 \succeq_F f_2 \text{ et } f_2 \succeq_F f_1$$

pour tout terme f_1, f_2 de F. De même, on note $\succeq_F^*$ (resp. $\succ_F^*$) l'extension multi-ensemble induite par $\succeq_F$ (resp. $\succ_F$). Soit $\succeq_F$ un quasi-ordre sur un ensemble de symboles fonctionnels F. Le quasi-ordre RPO $\succeq_F^{RPO}$ sur l'ensemble $T(F)$ des termes construits sur l'alphabet F est défini par

$$\begin{array}{ll} & s = f(s_1 \ldots s_m) \succeq_F^{RPO} g(t_1 \ldots t_n) = t \\ \text{si} & s_i \succeq_F^{RPO} t \text{ pour un } i = 1, \ldots, m \\ \text{ou} & f \succ_F g \text{ et } s \succ_F^{RPO} t_j \text{ pour tout } j = 1, \ldots, n \\ \text{ou} & f \approx_F g \text{ et } \{s_1 \ldots s_m\} \succ_F^{RPO*} \{t_1 \ldots t_n\} \end{array}$$

Voir [Der87].

Nous noterons $|(C_1 \cup C_2) \cap I|$ sa cardinalité. Nous définissons le poids d'une telle application de la règle de coupure comme étant $|(C_1 \cup C_2) \cap I|$ et nous définissons le poids d'une I-coupure comme étant nulle. (Remarque: il n'est pas possible d'avoir une application de la règle de coupure dans laquelle les deux prémisses de l'inférence soient I-négatives.) Définissons alors le poids μ_D d'une dérivation comme étant la somme des poids des coupures apparaissant dans la dérivation. Alors l'ordre $\succ_{\mathcal{D}}$ défini sur les dérivations par: $\mathcal{D}_1 \succ_{\mathcal{D}} \mathcal{D}_2$ si et seulement si $\mu_{\mathcal{D}}(\mathcal{D}_1) > \mu_{\mathcal{D}}(\mathcal{D}_2)$ est un ordre bien fondé.

On montre alors immédiatement en utilisant le lemme 2 que:

> Si $\mathcal{D}_1$ est une CC-dérivation arborescente d'une clause C_1 à partir d'un ensemble S de clauses et s'il existe une CC-dérivation arborescente $\mathcal{D}_2$ d'une clause $C_2 \subseteq C_1$ à partir de S telle que $\mathcal{D}_1 \succ_{\mathcal{D}} \mathcal{D}_2$, alors pour toute réfutation $\mathcal{D}$ contenant la dérivation $\mathcal{D}_1$ il existe une dérivation $\mathcal{D}'$ telle que $\mathcal{D} \succ_{\mathcal{D}} \mathcal{D}'$.

pour l'un et l'autre des ordres $\succ_{\mathcal{D}}$ définis précédemment.

Puisque dans les deux cas $\succ_{\mathcal{D}}$ est bien fondé, il suffit de montrer que si $\mathcal{D}$ est une réfutation non normale, alors on peut construire une dérivation $\mathcal{D}'$ telle que $\mathcal{D} \succ_{\mathcal{D}} \mathcal{D}'$.

On remarque que toute réfutation $\mathcal{D}$ est telle que la dernière inférence de toute réfutation: $\dfrac{A \quad \neg A}{\square}$ est une application de la coupure ordonnée (resp. de la coupure sémantique). Donc une réfutation $\mathcal{D}$ qui n'est pas en forme normale contient forcément une dérivation $\mathcal{D}''$ de la forme:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{C_1} \quad \dfrac{\mathcal{D}_2}{C_2}}{C}\ \textit{coupure}}{\ldots \text{plusieurs contractions} \ldots} \quad \dfrac{\mathcal{D}_3}{C_3}}{C'}\ \textit{coupure}}{\ldots \text{plusieurs contractions} \ldots}$$
$$C'^*$$

telle que le première application de la règle de coupure n'est pas en forme normale et la seconde application de la règle de coupure est en forme normale.

Nous montrons comment transformer une telle dérivation $\mathcal{D}''$ en une dérivation $\mathcal{D}'''$ telle que $\mathcal{D}'' \succ_{\mathcal{D}} \mathcal{D}'''$ et donc, par application du lemme précité, comment transformer une dérivation non normale $\mathcal{D}$ contenant $\mathcal{D}''$ en une dérivation $\mathcal{D}'$ telle que $\mathcal{D} \succ_{\mathcal{D}} \mathcal{D}'$.

On considère les règles de transformations:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{C_1 \vee L_A \vee L_B} \quad \dfrac{\mathcal{D}_2}{C_2 \vee L_B^c}}{C_1 \vee C_2 \vee L_A}}{\ldots \text{plusieurs contractions} \ldots} \quad \dfrac{\mathcal{D}_3}{C_3 \vee L_A^c}}{((C_1 \vee C_2)^* - \{L_A\}) \vee L_A}}{((C_1 \vee C_2)^* - \{L_A\}) \vee C_3}}{\ldots \text{plusieurs contractions} \ldots}}{(((C_1 \vee C_2)^* - \{L_A\}) \vee C_3)^*}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{C_1 \vee L_A \vee L_B} \quad \dfrac{\mathcal{D}_3}{C_3 \vee L_A^c}}{C_1 \vee C_3 \vee L_B}}{\ldots \text{plusieurs contractions} \ldots} \quad \dfrac{\mathcal{D}_2}{C_2 \vee L_B^c}}{((C_1 \vee C_3)^* - \{L_B\}) \vee L_B}}{((C_1 \vee C_3)^* - \{L_B\}) \vee C_2}}{\ldots \text{plusieurs contractions} \ldots}}{(((C_1 \vee C_3)^* - \{L_B\}) \vee C_2)^*}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{C_1 \vee L_A \vee L_B} \quad \dfrac{\mathcal{D}_2}{C_2 \vee L_A \vee L_B^c}}{C_1 \vee C_2 \vee L_A \vee L_A}}{\ldots \text{plusieurs contractions} \ldots} \quad \dfrac{\mathcal{D}_3}{C_3 \vee L_A^c}}{((C_1 \vee C_2)^* - \{L_A\}) \vee L_A}}{((C_1 \vee C_2)^* - \{L_A\}) \vee C_3}}{\ldots \text{plusieurs contractions} \ldots}}{(((C_1 \vee C_2)^* - \{L_A\}) \vee C_3)^*}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{C_1 \vee L_A \vee L_B} \quad \dfrac{\mathcal{D}_3}{C_3 \vee L_A^c}}{C_1 \vee C_3 \vee L_B}}{\ldots \text{plusieurs contractions} \ldots} \quad \dfrac{\dfrac{\dfrac{\mathcal{D}_2}{C_2 \vee L_A \vee L_B^c} \quad \dfrac{\mathcal{D}_3}{C_3 \vee L_A^c}}{C_2 \vee C_3 \vee L_B^c}}{\ldots \text{plusieurs contractions} \ldots}}{((C_1 \vee C_3)^* - \{L_B\}) \vee L_B \qquad ((C_2 \vee C_3)^* - \{L_B^c\}) \vee L_B^c}}{((C_1 \vee C_3)^* - \{L_B\}) \vee ((C_2 \vee C_3)^* - \{L_B^c\})}}{\ldots \text{plusieurs contractions} \ldots}}{(((C_1 \vee C_3)^* - \{L_B\}) \vee ((C_2 \vee C_3)^* - \{L_B^c\}))^*}$$

avec les hypothèses suivantes:

- L_A, L_A^c, L_B, L_B^c ne figurent pas dans C_1 (car les deux premières conditions du lemme sont supposées vérifiées). De même, L_B, L_B^c ne figurent pas dans C_2, et L_A, L_A^c ne figurent pas dans C_3;
- L_A ne figure pas dans C_2;
- $A \succ B$ et A est un élément maximal de l'ensemble des atomes figurant dans C_1, C_2, C_3 si l'on considère le problème de l'existence d'une procédure de normalisation pour la Résolution Ordonnée;

- $C_3 \vee L_A^c$ est une clause I-négative et $C_1 \vee L_A \vee L_B$ et $C_2 \vee L_B^c$ sont des clauses I-positives si l'on considère le problème de l'existence d'une procédure de normalisation pour la Résolution Sémantique.

En utilisant ces hypothèses, on montre que:

- $(C_1 \vee C_2)^* - \{L_A\} = (C_1 \vee C_2)^*$ car L_A ne figure ni dans C_1, ni dans C_2;
- $((C_1 \vee C_2)^* \vee C_3)^* = (C_1 \vee C_2 \vee C_3)^*$
- $(((C_1 \vee C_2)^* - \{L_A\}) \vee C_3)^* = (C_1 \vee C_2 \vee C_3)^*$

et l'on obtient alors:

$$(((C_1 \vee C_3)^* - \{L_B\}) \vee C_2)^* \subseteq (((C_1 \vee C_2)^* - \{L_A\}) \vee C_3)^*$$

$$(((C_1 \vee C_3)^* - \{L_B\}) \vee ((C_2 \vee C_3)^* - \{L_B^c\}))^* \subseteq (((C_1 \vee C_2)^* - \{L_A\}) \vee C_3)^*$$

Chacune de ces transformations transforme une CC-dérivation arborescente $\mathcal{D}''$ d'une clause C'' en une CC-dérivation arborescente $\mathcal{D}'''$ d'une clause $C''' \subseteq C''$. Il nous reste à montrer que $\mathcal{D}'' \succ_{\mathcal{D}} \mathcal{D}'''$.

Considérons d'abord le cas de la Résolution Ordonnée. La première règle de transformation transforme une déduction à laquelle on associe un terme $f_A(f_B(t_1, t_2), t_3)$ en une dérivation à laquelle on associe un terme $f_B(f_A(t_1, t_3), t_2)$ où t_i dénotent les termes associés aux dérivations $\frac{\mathcal{D}_i}{C_i}$ $(i = 1, 2, 3)$. Or on a:

$$f_A(f_B(t_1, t_2), t_3) \succ_F^{RPO} f_B(f_A(t_1, t_3), t_2)$$

car

- $f_A \succ f_B$ et
- $f_A(f_B(t_1, t_2), t_3) \succ_F^{RPO} f_A(t_1, t_3)$ car
  - $\{f_B(t_1, t_2), t_3\} \succ_F^{RPO*} \{t_1, t_3\}$
- $f_A(f_B(t_1, t_2), t_3) \succ_F^{RPO} t_2$

De même, on montre pour la deuxième règle que:

$$f_A(f_B(t_1, t_2), t_3) \succ_F^{RPO} f_B(f_A(t_1, t_3), f_A(t_2, t_3))$$

Considérons maintenant le cas de la Résolution Sémantique. La première règle de transformation transforme une déduction $\mathcal{D}''$ telle que

$$\mu_{\mathcal{D}}(\mathcal{D}'') = \sum_{i=1}^{3} \mu_{\mathcal{D}}(\mathcal{D}_i) + |((C_1 \vee L_A \vee L_B) \cup (C_2 \vee L_B^c)) \cap I|$$

en une déduction $\mathcal{D}'''$ telle que

$$\mu_{\mathcal{D}}(\mathcal{D}''') = \sum_{i=1}^{3} \mu_{\mathcal{D}}(\mathcal{D}_i) + |((((C_1 \vee C_3)^* - \{L_B\}) \vee L_B) \cup (C_2 \vee L_B^c)) \cap I|$$

On rappelle que:

- L_A, L_A^c, L_B, L_B^c ne figurent pas dans C_1 et L_A, L_B, L_B^c ne figure pas dans C_2 par hypothèse.
- L_A^c ne figure pas dans C_2 car sinon la résolvante $C_1 \vee C_2 \vee L_A$ serait une tautologie;
- $C_3 \vee L_A^c$ étant une clause négative, $((C_3 \vee L_A^c) \cap I) = \emptyset$. Donc les littéraux de C_3 n'interviennent pas dans le calcul de $\mu_{\mathcal{D}}(\mathcal{D}''')$ et l'occurrence de L_A dans $C_1 \vee L_A \vee L_B$ intervient dans $\mu_{\mathcal{D}}(\mathcal{D}'')$. (Car soit $L_A^c = A$ et $A \notin I$; soit $L_A^c = \neg A$ et $A \in I$).

Posons $k = |(C_1 \cup C_2) \cap I|$. On a alors:

$$\mu_{\mathcal{D}}(\mathcal{D}'') = \sum_{i=1}^{3} \mu_{\mathcal{D}}(\mathcal{D}_i) + k + 2$$

$$\mu_{\mathcal{D}}(\mathcal{D}''') = \sum_{i=1}^{3} \mu_{\mathcal{D}}(\mathcal{D}_i) + k + 1$$

D'où $\mu_{\mathcal{D}}(\mathcal{D}''') < \mu_{\mathcal{D}}(\mathcal{D}'')$. On procède de manière similaire pour la deuxième règle de transformation.

Enfin, constatons que si la procédure de normalisation de dérivations par Résolution en dérivations par Résolution Sémantique est appliquée à une dérivation vérifiant le fait que chaque application de la règle de coupure est une application de la règle de coupure ordonnée, on obtient alors une dérivation par Résolution Sémantique Ordonnée. C.Q.F.D.

Chapitre 3

Un calcul en Déduction Naturelle

3.1 Introduction

Dans [Gen69], G. Gentzen a présenté un calcul des prédicats du premier ordre appelé calcul en *Déduction Naturelle*. Ce calcul comprend un ensemble de règles d'inférence censées refléter les étapes élémentaires de raisonnement couramment employées en Mathématiques. Ce calcul permet d'établir la validité de formules soit du point de vue de la logique intuitionniste, soit du point de vue de la logique classique suivant que l'on rejette ou que l'on accepte le schéma d'axiomes $A \vee \neg A$ exprimant la loi du tiers exclu. On appelle NK la version classique de ce calcul et NJ la version intuitionniste.

Gentzen cherche alors à montrer que toute dérivation dans ce calcul peut être transformée en une dérivation sous forme normale qui soit "sans détour", c'est-à-dire en une dérivation qui ne contient pas d'autres formules que des instances de formules apparaissant dans le résultat final. Cette dernière propriété est appelée *propriété de la sous-formule* et l'existence d'un procédé systématique permettant de traduire toute preuve en une preuve vérifiant la propriété de la sous-formule est appelée

Hauptsatz ou théorème de *normalisation.* Gentzen annonce qu'il peut démontrer le Hauptsatz pour le calcul NJ mais pas pour le calcul NK (voir [Gen69] p 69 et aussi [Pra65] p 35). Gentzen établit alors son Hauptsatz pour un calcul auxiliaire appelé *calcul des séquents* et dont on dispose d'une version classique (LK) et d'une version intuitionniste (LJ). D. Prawitz établira le théorème de normalisation pour le calcul NJ et pour une variante du calcul NK [Pra65].

La Déduction Naturelle et le calcul des séquents sont donc des calculs distincts. Cependant, il est d'usage en démonstration automatique d'appeler Déduction Naturelle des calculs tels le calcul des séquents ou la méthode des tableaux [Smu68] par opposition à des calculs qui, comme la Résolution, utilisent peu de règles d'inférence applicables à des formules en forme normale.

Cet abus de langage se trouve justifié par le fait qu'il existe des correspondances profondes entre Déduction Naturelle, calcul des séquents et méthode des tableaux. Dans [Pra65] pp 88-93, le calcul des séquents LJ (resp. LK) est présenté comme un méta-calcul pour la relation de déduction dans le système de Déduction Naturelle NJ (resp. NK). Prawitz montre qu'en fait, une preuve dans le calcul des séquents peut être interprétée comme une suite d'instructions permettant de construire une preuve dans le calcul en Déduction Naturelle correspondant[1]. De même, une correspondance étroite entre calcul des séquents et méthode des tableaux peut être établie à partir des résultats de [Smu68].

Dans [Fel90], il est montré qu'une logique intuitionniste avec quantification du deuxième ordre sur les fonctions a un pouvoir d'expression suffisant pour spécifier simultanément un calcul des séquents et un calcul de Déduction Naturelle proche des calculs LJ et NJ. La spécification décrite dans cet article est exécutable dans le langage de programmation logique λprolog [NM88] et fournit une implémentation de la traduction d'une preuve dans un calcul des séquents en une preuve en Déduction Naturelle.

Dans ce chapitre, nous présentons le calcul des séquents LK ainsi que quelques unes des propriétés de ce calcul qui seront utilisées dans cette thèse. Nous présentons également un calcul appelé LK' dérivé de celui de Gentzen que nous utiliserons principalement par la suite en y faisant référence comme une calcul en Déduction Naturelle, selon l'usage en démonstration automatique.

[1] Cependant, à des preuves distinctes dans le calcul des séquents peut correspondre une unique preuve dans le calcul en Déduction Naturelle correspondant.

3.2 Le calcul des séquents

Nous présentons ici le calcul des séquents.

Définition: Un *séquent* S est une paire (Γ, Δ) de séquences de formules. Γ est appelé l'*antécédent* de S et Δ est appelé le *conséquent* de S. Si Γ et Δ désignent des séquences non vides de formules, on note:

- $\Gamma \Longrightarrow \Delta$ le séquent (Γ, Δ)
- $\Longrightarrow \Delta$ le séquent $(<>, \Delta)$
- $\Gamma \Longrightarrow$ le séquent $(\Gamma, <>)$

$\diamond$

Si Γ_1, Γ_2 sont des séquences de formules et si $\phi_1 \ldots \phi_n$ sont des formules, on notera $\Gamma_1, \phi_1 \ldots \phi_n, \Gamma_2$ la séquence de formules obtenue par concaténation des séquences $\Gamma_1, < \phi_1 \ldots \phi_n >, \Gamma_2$.

Définition: Le calcul des séquents comporte les axiomes et règles d'inférence suivants:

1. Axiome:

$$\frac{}{\phi_1 \Longrightarrow \phi_1}$$

2. Règles d'inférence *structurelles* (indépendantes des connectifs et des quantificateurs):

   - affaiblissement:

$$\frac{\Gamma \Longrightarrow \Delta}{\phi_1, \Gamma \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta, \phi_1}$$

   - contraction:

$$\frac{\phi_1, \phi_1, \Gamma \Longrightarrow \Delta}{\phi_1, \Gamma \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow \Delta, \phi_1, \phi_1}{\Gamma \Longrightarrow \Delta, \phi_1}$$

   - échange:

$$\frac{\Gamma_1, \phi_1, \phi_2, \Gamma_2 \Longrightarrow \Delta}{\Gamma_1, \phi_2, \phi_1, \Gamma_2 \Longrightarrow \Delta} \qquad \frac{\Gamma \Longrightarrow \Delta_1, \phi_1, \phi_2, \Delta_2}{\Gamma \Longrightarrow \Delta_1, \phi_2, \phi_1, \Delta_2}$$

3. Règles d'inférence *logiques* (concernant les connectifs et les quantificateurs):

   - introduction de $\neg$:

$$\neg - IA : \frac{\Gamma \Longrightarrow \Delta, \phi_1}{\neg\phi_1, \Gamma \Longrightarrow \Delta}$$

$$\neg - IC : \frac{\phi_1, \Gamma \Longrightarrow \Delta}{\Gamma \Longrightarrow \Delta, \neg\phi_1}$$

- introduction de $\wedge$:

$$\wedge - IA : \frac{\phi_1, \Gamma \Longrightarrow \Delta}{\phi_1 \wedge \phi_2, \Gamma \Longrightarrow \Delta} \quad \frac{\phi_2, \Gamma \Longrightarrow \Delta}{\phi_1 \wedge \phi_2, \Gamma \Longrightarrow \Delta}$$

$$\wedge - IC : \frac{\Gamma \Longrightarrow \Delta, \phi_1 \quad \Gamma \Longrightarrow \Delta, \phi_2}{\Gamma \Longrightarrow \Delta, \phi_1 \wedge \phi_2}$$

- introduction de $\vee$:

$$\vee - IA : \frac{\phi_1, \Gamma \Longrightarrow \Delta \quad \phi_2, \Gamma \Longrightarrow \Delta}{\phi_1 \vee \phi_2, \Gamma \Longrightarrow \Delta}$$

$$\vee - IC : \frac{\Gamma \Longrightarrow \Delta, \phi_1}{\Gamma \Longrightarrow \Delta, \phi_1 \vee \phi_2} \quad \frac{\Gamma \Longrightarrow \Delta, \phi_2}{\Gamma \Longrightarrow \Delta, \phi_1 \vee \phi_2}$$

- introduction de $\rightarrow$:

$$\rightarrow - IA : \frac{\Gamma_1 \Longrightarrow \Delta_1, \phi_1 \quad \phi_2, \Gamma_2 \Longrightarrow \Delta_2}{\phi_1 \rightarrow \phi_2, \Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}$$

$$\rightarrow - IC : \frac{\phi_1, \Gamma \Longrightarrow \Delta, \phi_2}{\Gamma \Longrightarrow \Delta, \phi_1 \rightarrow \phi_2}$$

- introduction de $\forall$:

$$\forall - IA : \frac{\phi_1(x/t), \Gamma \Longrightarrow \Delta}{\forall x \phi_1, \Gamma \Longrightarrow \Delta}$$

où t est un terme dont les variables libres ne sont pas capturées lors de la substitution de x par t dans ϕ_1

$$\forall - IC : \frac{\Gamma \Longrightarrow \Delta, \phi_1(x/u)}{\Gamma \Longrightarrow \Delta, \forall x \phi_1}$$

où u est une variable qui n'est pas libre dans $\Gamma, \Delta, \forall x \phi_1$

- introduction de $\exists$:

$$\exists - IA : \frac{\phi_1(x/u), \Gamma \Longrightarrow \Delta}{\exists x \phi_1, \Gamma \Longrightarrow \Delta}$$

où u est une variable qui n'est pas libre dans $\Gamma, \Delta, \exists x \phi_1$

$$\exists - IC : \frac{\Gamma \Longrightarrow \Delta, \phi_1(x/t)}{\Gamma \Longrightarrow \Delta, \exists x \phi_1}$$

où t est un terme dont les variables libres ne sont pas capturées lors de la substitution de x par t dans ϕ_1

4. Une règle dite de *coupure* ou *Cut*:

$$Cut : \frac{\Gamma_1 \Longrightarrow \Delta_1, \phi_1 \quad \phi_1, \Gamma_2 \Longrightarrow \Delta_2}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}$$

$\diamond$

A cause du rôle particulier de la règle de coupure, on notera $LK + cut$ le calcul présenté ci-dessus et LK ce calcul privé de la règle de coupure. Le calcul LJ (resp. $LJ + Cut$) utilise les mêmes schémas de règles d'inférence avec la restriction que tout séquent comporte dans sa partie conséquent *au plus une* formule.

Les règles dites structurelles permettent de manipuler les formules dans un séquent: en particulier, l'existence des règles d'échange permet de considérer un séquent comme une paire de multi-ensembles de formules.

Les règles dites logiques permettent l'introduction des connectifs et des quantificateurs dans les parties antécédents et conséquents des conclusions des règles d'inférence. Les formules $\neg\phi_1, \phi_1 \wedge \phi_2, \phi_1 \vee \phi_2, \phi_1 \rightarrow \phi_2, \forall x\phi_1, \exists x\phi_1$ apparaissant dans la conclusion des règles logiques sont appelées *formules principales* tandis que les formules $\phi_1, \phi_2, \phi_1(x/t), \phi_1(x/u)$ apparaissant dans les prémisses de ces mêmes règles sont appelées *formules secondaires* de l'inférence. La variable u apparaissant dans la prémisses des règles $\forall - IC, \exists - IA$ est appelée *eigenvariable* ("variable propre") de l'inférence. Notons enfin que les règles $\forall - IA, \exists - IC$ diffèrent de celles de [Gen69]: en effet, Gentzen ne considérait dans son calcul que la logique du premier ordre *sans* symbole fonctionnel. L'extension à la logique du premier ordre *avec* symboles fonctionnels que nous considérons ici est celle que l'on trouve, par exemple, dans [Gal86].

Définition: Un séquent S est dérivable par le calcul des séquents si et seulement si il existe une séquence finie de séquents $< S_1, \ldots, S_p >$ telle que:

- $S_p = S$
- $1 \leq i \leq p$, S_i est un séquent obtenu par une des règles d'inférence décrites ci-dessus.

La suite de séquents $< S_1, \ldots, S_p >$ est appelée une *déduction* ou une *dérivation* de S. Cette dérivation est dite *arborescente* si chaque séquent S_i est utilisé une seule fois comme prémisse d'une règle d'inférence. Une formule ϕ est dérivable par le calcul des séquents si le séquent $\Longrightarrow \phi$ est dérivable par ce calcul. $\diamond$

Que l'on se place dans le cadre de la logique classique ou dans celui de la logique intuitionniste, un séquent $\phi_1 \ldots \phi_m \Longrightarrow \psi_1 \ldots \psi_n$ sera dit *valide* si et seulement si la formule $(\bigwedge_{i=1}^{m} \phi_i) \rightarrow (\bigvee_{i=1}^{n} \psi_i)$ est valide. La correction de ce calcul se montre par la technique habituelle:

- tout axiome du calcul des séquents est valide;
- les règles d'inférence structurelles et logiques permettent de déduire des séquents valides si les séquents de leurs prémisses sont valides.

On en déduit donc que le calcul des séquents permet de déduire des formules valides. On peut trouver une preuve de la complétude de ce calcul dans [Gen69] ou [Gal86].

3.3 Instances de dérivations dans le calcul des séquents

Contrairement à la règle d'instanciation de la Résolution, on ne dispose pas dans le calcul LK d'une règle d'inférence permettant de déduire à partir d'un séquent S une "instance" de S. Nous donnons ici des conditions suffisantes pour construire à partir d'une déduction $\mathcal{D}$ d'un séquent S une déduction $\mathcal{D}'$ d'une "instance" de S. Lorsque nous traiterons ultérieurement de la traduction des preuves par Résolution en preuve en Déduction Naturelle, les résultats obtenus dans ce chapitre nous permettrons d'obtenir la preuve en déduction naturelle correspondant à la déduction par Résolution d'une clause de la forme $\sigma(C)$ à partir de la preuve en Déduction Naturelle obtenue par transformation de la déduction par Résolution de la clause C. D'autre part, ces résultats permettent également de justifier certaines hypothèses simplificatrices que nous ferons sur les déductions dans le calcul des séquents.

Définition: Une variable est *libre (resp. liée) dans un séquent* $\Gamma \Longrightarrow \Delta$ si et seulement si elle est libre (resp. liée) dans une formule apparaissant dans Γ, Δ.
Une variable est *libre (resp. liée) dans une dérivation* $\mathcal{D}$ si et seulement si elle est libre (resp. liée) dans un séquent de $\mathcal{D}$. $\diamond$

Etant donné un séquent $S = \phi_1 \dots \phi_m \Longrightarrow \psi_1 \dots \psi_n$ de LK, $S(x/t)$ désigne le séquent $\phi_1(x/t) \dots \phi_m(x/t) \Longrightarrow \psi_1(x/t) \dots \psi_n(x/t)$. Etant donné une dérivation $\mathcal{D} =< S_1 \dots S_n >$ dans LK, $\mathcal{D}(x/t)$ désigne la séquence de séquents $< S_1(x/t) \dots S_n(x/t) >$.

Bien entendu, si $\mathcal{D}$ une dérivation dans LK, $\mathcal{D}(x/t)$ peut ne pas être une dérivation dans LK. Cependant, nous allons montrer deux lemmes qui sont similaires mais un peu plus généraux que ceux utilisés dans la preuve du Hauptsatz pour le calcul des séquents (cf lemme 3.10 [Gen69] pp 90,91 ainsi que le lemme 6.4.3 dans [Gal86] p274). Nous redonnons la preuve de ces lemmes car

- d'une part, les lemmes que nous présentons utilisent une hypothèse de moins que ceux décrits dans [Gal86]: dans le calcul d'une instance $\mathcal{D}(x/t)$ d'une déduction $\mathcal{D}$, nous n'imposerons pas que la variable x ne soit jamais quantifiée dans une formule d'un séquent de $\mathcal{D}$. Ne pas utiliser une telle hypothèse permet de justifier simplement les hypothèses simplificatrices que nous ferons sur les déductions dans le calcul des séquents.

- d'autre part, l'utilisation de ces lemmes pour construire des preuves en Déduction Naturelle à partir de déductions par Résolution nécessite de détailler comment s'opère la construction de la dérivation $\mathcal{D}(x/t)$, au moins pour les cas les plus délicats.

Lemme 4 *Soit $\mathcal{D}$ est une dérivation dans LK d'un séquent $\Gamma \Longrightarrow \Delta$, et soit z une variable qui n'apparaît ni libre ni liée dans $\mathcal{D}$. Alors $\mathcal{D}(y/z)$ est une dérivation du séquent $\Gamma(y/z) \Longrightarrow \Delta(y/z)$.*

Preuve: Preuve par induction sur la longueur N de la dérivation $\mathcal{D}$.

- N=1. Alors $\mathcal{D}$ est une séquence constituée d'un seul séquent qui est nécessairement un axiome de la forme $\phi_1 \Longrightarrow \phi_1$. Le séquent $S' = \phi_1(y/z) \Longrightarrow \phi_1(y/z)$ est aussi un axiome et l'on a $\mathcal{D}(y/z) =< S' >$.

- Supposons la propriété établie pour des dérivations de longueur $n < N$ et montrons cette propriété pour les dérivations $\mathcal{D}$ de longueur N. Nous ne considérerons que les dérivations $\mathcal{D}$ dont la dernière règle d'inférence introduit un quantificateur $\forall, \exists$, les autres cas étant triviaux.

  - Le dernier séquent de $\mathcal{D}$ est obtenu par la règle $\exists - IC$: $\dfrac{\Gamma \Longrightarrow \Delta, \phi_1(x/t)}{\Gamma \Longrightarrow \Delta, \exists x \phi_1}$. Par hypothèse d'induction appliquée à la dérivation de $\Gamma \Longrightarrow \Delta, \phi_1(x/t)$, supposons que l'on dispose d'une dérivation du séquent $S = \Gamma(y/z) \Longrightarrow \Delta(y/z), \phi_1(x/t)(y/z)$. On considère alors les cas suivants:

    1. $y \neq x$. Alors $\phi_1(x/t)(y/z) = \phi_1(y/z)(x/t(y/z))$.
    Puisque les variables de t ne sont pas capturées lors du remplacement de x par t dans ϕ_1 et que z n'apparaît pas dans $\mathcal{D}$, les variables de $t(y/z)$ ne peuvent être capturées lors du remplacement de x par $t(y/z)$ dans $\phi_1(y/z)$. Donc on peut appliquer la règle $\exists - IC$ à S et l'on déduit: $\Gamma(y/z) \Longrightarrow \Delta(y/z), \exists x(\phi_1(y/z))$. D'autre part, puisque

$y \neq x$, $\exists x(\phi_1(y/z)) = (\exists x\phi_1)(y/z)$; d'où $(\Gamma \Longrightarrow \Delta, \exists x\phi_1)(y/z) = (\Gamma(y/z) \Longrightarrow \Delta(y/z), \exists x\phi_1(y/z))$.

2. $y = x$. Alors $\phi_1(x/t)(x/z) = \phi_1(x/t(x/z))$.
   Puisque les variables de t ne sont pas capturées lors du remplacement de x par t dans ϕ_1 et que z n'apparaît pas dans $\mathcal{D}$, les variables de $t(x/z)$ ne sont pas capturées lors du remplacement de x par $t(x/z)$ dans ϕ_1. Donc on peut appliquer la règle $\exists - IC$ à S et l'on déduit: $\Gamma(x/z) \Longrightarrow \Delta(x/z), \exists x\phi_1$. D'autre part, $\exists x\phi_1 = (\exists x\phi_1)(x/z)$; d'où $(\Gamma \Longrightarrow \Delta, \exists x\phi_1)(x/z) = (\Gamma(x/z) \Longrightarrow \Delta(x/z), \exists x\phi_1)$.

- Si le dernier séquent de $\mathcal{D}$ est obtenu par la règle $\forall - IA$: $\dfrac{\phi_1(x/t), \Gamma \Longrightarrow \Delta}{\forall x\phi_1, \Gamma \Longrightarrow \Delta}$, le traitement est similaire au précédent.

- Le dernier séquent de $\mathcal{D}$ est obtenu par la règle $\forall - IC$: $\dfrac{\Gamma \Longrightarrow \Delta, \phi_1(x/u)}{\Gamma \Longrightarrow \Delta, \forall x\phi_1}$ et u n'est pas libre dans $\Gamma, \Delta, \forall x\phi_1$. On remarque que comme z n'apparaît pas dans $\mathcal{D}$, $z \neq u$ et $z \neq x$ et donc u n'est pas libre dans $\Gamma(y/z), \Delta(y/z), \forall x(\phi_1(y/z))$.
  Par hypothèse d'induction appliquée à la dérivation de $\Gamma \Longrightarrow \Delta, \phi_1(x/u)$, supposons que l'on dispose d'une dérivation de S=$\Gamma(y/z) \Longrightarrow \Delta(y/z), \phi_1(x/u)(y/z)$. On considère alors les cas suivants:

  1. $y \neq x, y \neq u$. On a alors $\phi_1(x/u)(y/z) = \phi_1(y/z)(x/u)$ et puisque u n'est pas libre dans $\Gamma(y/z), \Delta(y/z), \forall x(\phi_1(y/z))$ on peut appliquer la règle $\forall - IC$ à S et l'on déduit: $\Gamma(y/z) \Longrightarrow \Delta(y/z), \forall x(\phi_1(y/z))$. D'autre part, puisque $y \neq x$, $\forall x(\phi_1(y/z)) = (\forall x\phi_1)(y/z)$; d'où $(\Gamma \Longrightarrow \Delta, \forall x\phi_1)(y/z) = (\Gamma(y/z) \Longrightarrow \Delta(y/z), \forall x(\phi_1(y/z)))$.
  2. $y = x, y \neq u$. On a alors $\phi_1(x/u)(x/z) = \phi_1(x/u)$ et puisque u n'est pas libre dans $\Gamma(x/z), \Delta(x/z), \forall x\phi_1$ on peut appliquer la règle $\forall - IC$ à S et l'on déduit: $\Gamma(x/z) \Longrightarrow \Delta(x/z), \forall x\phi_1$. D'autre part, $(\forall x\phi_1) = (\forall x\phi_1)(x/z)$; d'où $(\Gamma \Longrightarrow \Delta, \forall x\phi_1)(x/z) = (\Gamma(x/z) \Longrightarrow \Delta(x/z), \forall x\phi_1)$.
  3. $y = u$. On a alors $\phi_1(x/u)(u/z) = \phi_1(x/z)$. Comme u n'est pas libre dans Γ, Δ, on a $\Gamma(u/z) = \Gamma$ et $\Delta(u/z) = \Delta$ et comme z n'apparaît pas dans $\mathcal{D}$, z n'est pas libre dans $\Gamma, \Delta, \forall x\phi_1$. Donc on peut appliquer la règle $\forall - IC$ à S et l'on déduit: $\Gamma \Longrightarrow \Delta, \forall x\phi_1$. D'autre part, ou bien $u \neq x$ et $(\phi_1(u/z)) = \phi_1$ car u est non libre dans $\forall x\phi_1$ et

donc dans ϕ_1; on a alors $(\Gamma \Longrightarrow \Delta, \forall x\phi_1)(u/z) = (\Gamma \Longrightarrow \Delta, \forall x\phi_1)$; ou bien $u = x$ et $\forall x\phi_1 = (\forall x\phi_1)(x/z)$ et $(\Gamma \Longrightarrow \Delta, \forall x\phi_1)(x/z) = (\Gamma \Longrightarrow \Delta, \forall x\phi_1)$.

- Si le dernier séquent de $\mathcal{D}$ est obtenu par la règle $\exists - IA$: $\frac{\phi_1(x/u), \Gamma \Longrightarrow \Delta}{\exists x\phi_1, \Gamma \Longrightarrow \Delta}$, le traitement est similaire au précédent.

- Le cas des autres règles d'inférence est trivial.

C.Q.F.D.

Lemme 5 *Soit $\mathcal{D}$ une dérivation du séquent $\Gamma \Longrightarrow \Delta$. Soit y une variable qui n'est pas une variable propre d'une application d'une règle d'inférence $\forall - IC, \exists - IA$ dans $\mathcal{D}$. Soit s un terme dont les variables libres sont distinctes des variables quantifiées dans les formules des séquents de $\mathcal{D}$ et sont distinctes des variables propres apparaissant dans $\mathcal{D}$. Alors $\mathcal{D}(y/s)$ est une dérivation dans LK du séquent $\Gamma(y/s) \Longrightarrow \Delta(y/s)$.*

Preuve: Ici encore on raisonne par induction sur la longueur de la dérivation $\mathcal{D}$. Seul le cas des règles d'introduction des quantificateurs est délicat.

Supposons que le dernier séquent de $\mathcal{D}$ soit obtenu par la règle $\exists - IC$: $\frac{\Gamma \Longrightarrow \Delta, \phi_1(x/t)}{\Gamma \Longrightarrow \Delta, \exists x\phi_1}$ et que, par hypothèse d'induction appliquée à la dérivation de $\Gamma \Longrightarrow \Delta, \phi_1(x/t)$, on dispose d'une dérivation de $S = \Gamma(y/s) \Longrightarrow \Delta(y/s), \phi_1(x/t)(y/s)$. On distingue alors deux cas

- $x \neq y$.
  On a $\phi_1(x/t)(y/s) = \phi_1(y/s)(x/t(y/s))$. De plus, les variables libres de s sont distinctes des variables quantifiées dans $\mathcal{D}$. Donc les variables de $t(y/s)$ ne sont pas capturées lors de la substitution de x par $t(y/s)$ dans $\phi_1(y/s)$. On peut alors appliquer la règle $\exists - IC$ à S et l'on obtient le séquent $\Gamma(y/s) \Longrightarrow \Delta(y/s), \exists x(\phi_1(y/s))$. Or $(\exists x\phi_1)(y/s) = \exists x(\phi_1(y/s))$ et l'on a bien $\Gamma(y/s) \Longrightarrow \Delta(y/s), \exists x(\phi_1(y/s)) = (\Gamma \Longrightarrow \Delta, \exists x\phi_1)(y/s)$.

- $x = y$.
  On a $\phi_1(x/t)(x/s) = \phi_1(x/t(x/s))$. De plus, les variables libres de s sont distinctes des variables quantifiées dans $\mathcal{D}$. Donc les variables de $t(y/s)$ ne sont pas capturées lors de la substitution de x par $t(y/s)$ dans ϕ_1. On peut alors appliquer la règle $\exists - IC$ à S et l'on obtient le

séquent $\Gamma(x/s) \Longrightarrow \Delta(x/s), \exists x\phi_1$. Or $(\exists x\phi_1)(x/s) = \exists x\phi_1$ et l'on a bien $(\Gamma(x/s) \Longrightarrow \Delta(x/s), \exists x\phi_1) = (\Gamma \Longrightarrow \Delta, \exists x\phi_1)(x/s)$.

Le traitement du cas de $\forall - IA$ est similaire.

Considérons désormais que le dernier séquent de $\mathcal{D}$ est obtenu par $\forall - IC$: $\dfrac{\Gamma \Longrightarrow \Delta, \phi_1(x/u)}{\Gamma \Longrightarrow \Delta, \forall x\phi_1}$ et que, par hypothèse d'induction appliquée à la dérivation de $\Gamma \Longrightarrow \Delta, \phi_1(x/u)$, on dispose d'une dérivation de $S = \Gamma(y/s) \Longrightarrow \Delta(y/s), \phi_1(x/u)(y/s)$. Ici encore, on considère deux cas:

- $x \neq y$
  On a de plus $y \neq u$ et $u \notin VAR(s)$ par hypothèse (u est une variable propre). De même, $x \notin VAR(s)$ car les variables libres de s ne sont pas quantifiées dans $\mathcal{D}$. Donc, $\phi_1(x/u)(y/s) = \phi_1(y/s)(x/u)$. Puisque $u \notin VAR(s)$ et u n'est pas libre dans $\Gamma, \Delta, \forall x\phi_1$ (restriction sur la variable u dans $\forall - IC$), u n'est pas libre dans $\Gamma(y/s), \Delta(y/s), \forall x(\phi_1(y/s))$. On peut alors appliquer la règle $\forall - IC$ à S et l'on obtient le séquent $\Gamma(y/s) \Longrightarrow \Delta(y/s), \forall x\phi_1(y/s)$. Or $\forall x\phi_1(y/s) = (\forall x\phi_1)(y/s)$ et donc $(\Gamma(y/s) \Longrightarrow \Delta(y/s), \forall x\phi_1(y/s)) = (\Gamma \Longrightarrow \Delta, \forall x\phi_1)(y/s)$.

- $x = y$
  On rappelle que $y \neq u$ et $u \notin VAR(s)$ par hypothèse (u est une variable propre) et que $x \notin VAR(s)$ car les variables libres de s ne sont pas quantifiées dans $\mathcal{D}$. Donc, $\phi_1(x/u)(x/s) = \phi_1(x/u)$. Puisque $u \notin VAR(s)$ et u n'est pas libre dans $\Gamma, \Delta, \forall x\phi_1$ (restriction sur la variable u dans $\forall - IC$), u n'est pas libre dans $\Gamma(x/s), \Delta(x/s), \forall x\phi_1$. On peut alors appliquer la règle $\forall - IC$ à S et l'on obtient le séquent $\Gamma(x/s) \Longrightarrow \Delta(x/s), \forall x\phi_1$. Or $\forall x\phi_1 = (\forall x\phi_1)(x/s)$ et donc $\Gamma(x/s) \Longrightarrow \Delta(x/s), \forall x\phi_1 = (\Gamma \Longrightarrow \Delta, \forall x\phi_1)(x/s)$.

Le traitement du cas de $\exists - IA$ est similaire. C.Q.F.D.

Corollaire 1 *Toute déduction $\mathcal{D}$ d'un séquent $\Gamma \Longrightarrow \Delta$ peut être transformée en une déduction $\mathcal{D}'$ d'un séquent $\Gamma' \Longrightarrow \Delta'$ telle qu'aucune variable libre de $\mathcal{D}'$ n'apparaît liée dans $\mathcal{D}'$. Cette transformation s'effectue simplement par renommage des variables libres de $\mathcal{D}$.*

Par analogie avec la notion de formule rectifiée ([Gal86] p 171), nous dirons

Définition: Une formule ϕ est une formule *quasi-rectifiée* si et seulement si, pour toute variable x apparaissant liée dans ϕ

- x n'apparaît pas libre dans ϕ;
- il n'existe pas deux sous-formules distinctes de ϕ de la forme $Q_i x \phi_i$, $Q_i \in \{\forall, \exists\}$ $i = 1, 2$ telles que $Q_1 x \phi_1$ soit une sous-formule de $Q_2 x \phi_2$.

$\diamond$

Lemme 6 *Il existe un algorithme qui, étant donnée une formule ϕ, produit une formule ϕ' quasi-rectifiée et équivalente à ϕ.*

Preuve: On utilise les propriétés suivantes ([Gal86] p 171)

- Si ϕ est une formule telle que la variable y n'apparaît pas dans ϕ, alors $Qx\phi$ et $Qy\phi(x/y)$, $Q \in \{\forall, \exists\}$ sont deux formules équivalentes.
- Si ϕ_1 et ϕ_2 sont deux formules équivalentes, alors pour toute formule ϕ telle que ϕ_1 est une sous-formule de ϕ, ϕ et $\phi[\phi_1 \leftarrow \phi_2]$ sont deux formules équivalentes.

Soit $\{Q_1 x_1 \phi_1 \dots Q_n x_n \phi_n\}$ l'ensemble des sous-formules de ϕ de la forme $Qx\psi$ telles qu'il existe dans ψ une sous-formule de la forme $Q'x\psi'$. Soit $\{z_1 \dots z_n\}$ un ensemble de variables n'apparaissant ni libre ni liée dans ϕ. Alors la formule

$$(\dots(\phi[Q_1 x_1 \phi_1 \leftarrow Q_1 z_1 \phi_1(x_1/z_1)])\dots)[Q_n x_n \phi_n \leftarrow Q_n z_n \phi_n(x_n/z_n)]$$

est une formule quasi-rectifiée équivalente à ϕ. C.Q.F.D.

Remarquons que si $\mathcal{D}$ est une déduction dans le calcul des séquents d'une formule ϕ, alors toute formule quasi-rectifiée équivalente à ϕ obtenue comme décrit ci-dessus admet dans le calcul des séquents une preuve comportant le même nombre d'inférence et le même nombre de symboles que $\mathcal{D}$. En effet, soit $\{Q_1 x_1 \phi_1 \dots Q_n x_n \phi_n\}$ l'ensemble des sous-formules de $\mathcal{D}$ de la forme $Qx\psi$ telle qu'il existe dans ψ une sous-formule de la forme $Q'x\psi'$. Soit $\{z_1 \dots z_n\}$ un ensemble de variables n'apparaissant ni libre ni liée dans $\mathcal{D}$. Alors la formule

$$(\dots(\mathcal{D}[Q_1 x_1 \phi_1 \leftarrow Q_1 z_1 \phi_1(x_1/z_1)])\dots)[Q_n x_n \phi_n \leftarrow Q_n z_n \phi_n(x_n/z_n)]$$

est une déduction d'une formule quasi-rectifiée équivalente à ϕ.

Remarque 5 *Ceci n'est pas vrai si l'on considère l'algorithme qui permet d'obtenir des formules* rectifiées.

En effet, soit la suite de formule définie par:

- $\phi_0 = \exists x(p(x) \rightarrow p(x))$
- $\phi_{n+1} = (\phi_n \wedge \phi_n)\ 0 \leq n$

On montre facilement que pour tout n, ϕ_n admet une dérivation *non arborescente* comportant un nombre linéaire d'inférences. Cependant, toute preuve d'une formule rectifiée obtenue à partir de ϕ_n comme décrit dans ([Gal86] p 171) comportera $O(2^n)$ inférences.

Par la suite nous ne considérerons que *des dérivations $\mathcal{D}$ dans LK de formules valides quasi-rectifiées telles qu'aucune variable libre de $\mathcal{D}$ n'apparaît liée dans $\mathcal{D}$.* Ceci n'entraîne aucune perte de généralité, tant sur la classe de formules valides dérivables que sur la taille des déductions de ces formules.

3.4 Règles d'inférence dérivées

Considérons les deux règles d'inférence:

$$\wedge - IC' : \frac{\Gamma_1 \Longrightarrow \Delta_1, \phi_1 \quad \Gamma_2 \Longrightarrow \Delta_2, \phi_2}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \phi_1 \wedge \phi_2}$$

$$\vee - IA' : \frac{\phi_1, \Gamma_1 \Longrightarrow \Delta_1 \quad \phi_2, \Gamma_2 \Longrightarrow \Delta_2}{\phi_1 \vee \phi_2, \Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}$$

Appelons LK' le calcul des séquents comportant les mêmes règles d'inférence que LK à l'exception des règles $\wedge - IC$, $\vee - IA$ qui sont respectivement remplacées par les règles $\wedge - IC'$, $\vee - IA'$. Alors tout séquent dérivable dans LK est dérivable dans LK' et réciproquement.

En effet, une application de la règle $\wedge - IC'$ dans LK' peut être simulée dans LK par la déduction:

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \Longrightarrow \Delta_1, \phi_1}{\text{Affaiblissements et échanges..}}}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \phi_1} \quad \dfrac{\dfrac{\Gamma_2 \Longrightarrow \Delta_2, \phi_2}{\text{Affaiblissements et échanges..}}}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \phi_2}}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \phi_1 \wedge \phi_2}$$

Inversement, toute application de la règle $\wedge - IC$ dans LK peut être simulée dans LK' par la déduction:

$$\dfrac{\dfrac{\dfrac{\Gamma \Longrightarrow \Delta, \phi_1 \quad \Gamma \Longrightarrow \Delta, \phi_2}{\Gamma, \Gamma \Longrightarrow \Delta, \Delta, \phi_1 \wedge \phi_2}}{\text{Contractions et échanges..}}}{\Gamma \Longrightarrow \Delta, \phi_1 \wedge \phi_2}$$

* $SI(\phi_1, S_1, \mathcal{D}) =< O.1, -p, \sigma >$;
* $SI(\phi_2, S_2, \mathcal{D}) =< O.2, p, \sigma >$;
* $\forall \psi \in \Gamma, \Delta, SI(\psi, S_1, \mathcal{D}) = SI(\psi, S, \mathcal{D})$.

– S est obtenu à partir de S_1 par la règle: $\forall - IA$ (resp. $\exists - IC$) et $SI(\forall x \phi_1, S, \mathcal{D}) =< O, p, \sigma >$ (resp. $SI(\exists x \phi_1, S, \mathcal{D}) =< O, p, \sigma >$).

* $SI(\phi_1(x/t), S_1, \mathcal{D}) =< O.1, p, \{x \mapsto t\} \circ \sigma >$;
* $\forall \psi \in \Gamma, \Delta, SI(\psi, S_1, \mathcal{D}) = SI(\psi, S, \mathcal{D})$.

Avec les hypothèses faites sur les dérivations, on a

* $x \notin Dom(\sigma)$ car on ne considère que des formules quasi-rectifiées;
* $(Dom(\sigma) \cup \{x\}) \cap (Var(t) \cup [\cup_{y \in Dom(\sigma)} Var(\sigma(y))]) = \emptyset$ car variables liées et variables libres sont distinctes.

– S est obtenu à partir de S_1 par une des règles: $\exists - IA$ (resp. $\forall - IC$) et $SI(\exists x \phi_1, S, \mathcal{D}) =< O, p, \sigma >$ (resp. $SI(\forall x \phi_1, S, \mathcal{D}) =< O, p, \sigma >$).

* $SI(\phi_1(x/u), S_1, \mathcal{D}) =< O.1, p, \{x \mapsto u\} \circ \sigma >$;
* $\forall \psi \in \Gamma, \Delta, SI(\psi, S_1, \mathcal{D}) = SI(\psi, S, \mathcal{D})$.

Avec les hypothèses faites sur les dérivations, on a

* $x \notin Dom(\sigma)$ car car on ne considère que des formules quasi-rectifiées;
* $(Dom(\sigma) \cup \{x\}) \cap (\{u\} \cup [\cup_{y \in Dom(\sigma)} Var(\sigma(y))]) = \emptyset$ car variables liées et variables libres sont distinctes;

Chapitre 4

Simulation polynomiale du calcul LK' par la Résolution

4.1 Résultats

Dans ce chapitre, nous établissons une correspondance entre la dérivation d'un séquent dans le calcul LK' et la dérivation par Résolution d'une clause correspondant à ce séquent à partir d'un ensemble de clauses obtenues en utilisant le renommage exhaustif. Cette correspondance permet d'établir une *simulation polynomiale* du calcul LK' par la Résolution

Si l'on ne considère que des dérivations arborescentes dont les axiomes sont de la forme $A \Longrightarrow A$ où A est une formule atomique, alors la taille de la déduction dans LK' et la taille de la preuve par Résolution correspondante sont du même ordre de grandeur, que l'on mesure la taille d'une preuve en nombre de symboles apparaissant dans la preuve ou en nombre d'inférences. La simulation de LK' par la Résolution est alors *une simulation linéaire.*

En général, les systèmes de démonstration automatique utilisant les méthodes de

déduction naturelle cherchent à construire de telles dérivations. En effet, considérer des dérivations plus générales - ce qui d'un point de vue théorique permet un gain d'efficacité- s'avère souvent peu satisfaisant en pratique.

Par exemple, considérer des axiomes de la forme $\phi \Longrightarrow \phi$ où ϕ est une formule non nécessairement atomique impose de reconnaître des formules identiques, ce qui est aussi coûteux que de dériver le séquent $\phi \Longrightarrow \phi$ à partir d'axiomes comportant des formules atomiques. Nous montrons en fait que la transformation d'une dérivation $\mathcal{D}$ d'une formule ψ en une dérivation $\mathcal{D}'$ de ψ dont les axiomes sont de la forme $A \Longrightarrow A$ où A est une formule atomique conduit à une augmentation polynomiale du nombre d'inférences de $\mathcal{D}'$: si n est le nombre de séquents de $\mathcal{D}$, alors le nombre d'inférences de $\mathcal{D}'$ est en $O(n|SF(\psi)|)$ et la taille mesurée en nombre de symboles de $\mathcal{D}'$ est en $O(|\mathcal{D}||SF(\psi)|)$. D'où le caractère généralement polynomial de la simulation de LK' par la Résolution.

D'autre part, considérer des dérivations non arborescentes est, sauf cas particulier, très coûteux car cela nécessite de maintenir une bibliothèque des séquents déjà dérivés. Bien que nous ne disposons pas d'expérimentations à ce sujet, nous justifions ce point de vue par les résultats suivants relatifs à la Résolution ou à la Model Elimination [Lov78]:

- le test de θ-subsomption nécessite (dans le cas clausal) une recherche qui s'apparente à la recherche d'un résultat déjà démontré dans une bibliothèque de lemmes. Or, cette recherche est connue pour être très coûteuse.

- certaines procédure de preuve comme la "Model Elimination" permettent de réutiliser certaines dérivations sous forme de lemmes. Cependant, les expérimentations rapportées dans [FLSD74] montre qu'en fait l'utilisation de ces lemmes est souvent pénalisante car on ne dispose pas encore de moyen pour évaluer la pertinence de tel ou tel résultat intermédiaire.

Malgré ces objections, nous décrivons au paragraphe 3 un nouveau renommage (inspiré de [PG86]) et nous montrons que cette nouvelle transformation permet de simuler de manière linéaire les dérivations *non arborescente* dont les axiomes sont de la forme $A \Longrightarrow A$ où A est une formule atomique. Nous mentionerons également les difficultés qu'il y a à refuter une forme clausale obtenue avec cette transformation du point de vue de la Résolution.

4.2 Un cas particulier intéressant en pratique

Etant donné un séquent $S = \Gamma \Longrightarrow \Delta$ d'une dérivation $\mathcal{D}$ dans LK' de $\Longrightarrow \phi$, on note $C_{\Gamma \Longrightarrow \Delta}$ la clause:

$$(\bigvee_{\gamma \in \Gamma} \sigma_\gamma(\neg NP_\gamma(x_1 \ldots x_{n_\gamma}))) \vee (\bigvee_{\delta \in \Delta} \sigma_\delta(NP_\delta(x_1 \ldots x_{n_\delta})))$$

où $\forall \psi \in (\Gamma \cup \Delta), SI(\psi, S, \mathcal{D}) =< O, p, \sigma_\psi >$ et $NP_\psi(x_1 \ldots x_{n_\psi})$ est le prédicat de Skolem introduit lors du renommage de $\neg\phi$ à l'occurrence $1.O$.

Théorème 6 *Soit $\mathcal{D}$ une dérivation arborescente dans le calcul LK' d'une formule ϕ telle que tous les axiomes de $\mathcal{D}$ soient de la forme $A \Longrightarrow A$ où A est une formule atomique. Alors il existe une réfutation par Résolution de l'ensemble de clauses $CRn(\neg\phi)$ comportant au plus $O(n)$ applications de règles d'inférence dont au plus*

- *$2n + 2$ applications de la règle de coupure*
- *$2n$ applications de la règle d'instanciation*
- *n applications de la règle de contraction*

si n est le nombre de séquents apparaissant dans la dérivation $\mathcal{D}$.

Preuve: Pour tout séquent $\Gamma \Longrightarrow \Delta$ de $\mathcal{D}$, on montre comment construire une dérivation par Résolution d'une clause C, telle que $C \subseteq C_{\Gamma \Longrightarrow \Delta}$ [1].

- $S = A \Longrightarrow A$ où A est une formule atomique.

  Soient $< O_1, -1, \sigma_1 >$, $< O_2, 1, \sigma_2 >$ les valeurs respectives de SI pour l'occurrence de A dans la partie antécédent de S et l'occurrence de A dans la partie conséquent de S. Soient $L_1 = \phi\,|_{O_1}$ et $L_2 = \phi\,|_{O_2}$. Par définition de SI, $A = \sigma_1(L_1) = \sigma_2(L_2)$.

  Soit n_i $(i = 1, 2)$ le nombre de variables libres de L_i. Comme O_1 (resp. O_2) désigne une sous-formule de polarité négative (resp. positive) de ϕ, soit $C_1 = (\neg NP_1(x_1 \ldots x_{n_1}) \vee L_1)$ (resp. $C_2 = (NP_2(x_1 \ldots x_{n_2}) \vee \neg L_2)$) la clause correspondant au renommage de l'occurrence $1.O_1$ (resp. $1.O_2$) dans $\neg\phi$.

  On construit alors à partir de $\{C_1, C_2\}$ une dérivation par Résolution de $C_{A \Longrightarrow A} = \sigma_1(\neg NP_1(x_1 \ldots x_{n_1})) \vee \sigma_2(NP_2(x_1 \ldots x_{n_2}))$

[1] On rappelle que l'inclusion s'entend ici au sens de l'inclusion entre les multi-ensembles de littéraux figurant dans chacune des deux clauses

1. $(\neg NP_1(x_1 \ldots x_{n_1}) \vee L_1)$
2. $(NP_2(x_1 \ldots x_{n_2}) \vee \neg L_2)$
3. $\sigma_1(\neg NP_1(x_1 \ldots x_{n_1}) \vee L_1)$ (instanciation)
4. $\sigma_2(NP_2(x_1 \ldots x_{n_2}) \vee \neg L_2)$ (instanciation)
5. $\sigma_1(\neg NP_1(x_1 \ldots x_{n_1})) \vee \sigma_2(NP_2(x_1 \ldots x_{n_2}))$ (coupure)

- $S = \Gamma \Longrightarrow \Delta, \psi_1 \vee \psi_2$ et S est obtenu par application de la règle $\vee - IC$.

  Les cas $\frac{\Gamma \Longrightarrow \Delta, \psi_1}{\Gamma \Longrightarrow \Delta, \psi_1 \vee \psi_2}$ et $\frac{\Gamma \Longrightarrow \Delta, \psi_2}{\Gamma \Longrightarrow \Delta, \psi_1 \vee \psi_2}$ étant symétrique, considérons simplement le premier. Supposons que $SI(\psi_1 \vee \psi_2, S, \mathcal{D}) =< O, 1, \sigma >$. On a donc: $SI(\psi_1, S_1, \mathcal{D}) =< O.1, 1, \sigma >$ où $S_1 = \Gamma \Longrightarrow \Delta, \psi_1$. $\psi_1 \vee \psi_2$ est une instance d'une sous-formule $\phi_1 \vee \phi_2$ apparaissant positivement dans ϕ et négativement dans $\neg\phi$. Le renommage de $\phi_1 \vee \phi_2$ dans $\neg\phi$ introduit donc la définition :

  $$(\forall x_1 \ldots x_n(\bigvee_{i=1}^{2} NP_{\phi_i}[x_1 \ldots x_n]) \rightarrow NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n))$$

  et $(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n)) \in CRn(\neg\phi)$.

  Par hypothèse d'induction, $CRn(\neg\phi) \vdash C$ et $C \subseteq C_{\Gamma \Longrightarrow \Delta, \psi_1}$.

  - Si $C \subseteq C_{\Gamma \Longrightarrow \Delta}$ alors $C \subseteq C_{\Gamma \Longrightarrow \Delta, \psi_1 \vee \psi_2}$.
  - Sinon $\sigma(NP_{\phi_1}[x_1 \ldots x_n]) \in C$. Posons $C = (\sigma(NP_{\phi_1}[x_1 \ldots x_n]) \vee C')$. On construit à partir de:

    $$\{(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n)), C\}$$

    une dérivation d'une clause incluse dans $C_{\Gamma \Longrightarrow \Delta, \phi_1 \vee \phi_2}$ comme suit:

    1. $(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n))$
    2. $(\sigma(NP_{\phi_1}[x_1 \ldots x_n]) \vee C')$
    3. $\sigma(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n))$ (instanciation).
    4. $(\sigma(NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n)) \vee C')$ (coupure).

  D'où l'existence d'une dérivation à partir de $CRn(\neg\phi)$ d'une clause $C \subseteq C_{\Gamma \Longrightarrow \Delta, \psi_1 \vee \psi_2}$.

- $S = \Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \psi_1 \wedge \psi_2$ et S est obtenu par application de la règle $\wedge - IC'$.

Supposons que $SI(\psi_1 \wedge \psi_2, S, \mathcal{D}) =< O, 1, \sigma >$. On a donc:

$$SI(\psi_1, S_1, \mathcal{D}) =< O.1, 1, \sigma > \qquad SI(\psi_2, S_2, \mathcal{D}) =< O.2, 1, \sigma >$$

en posant $S_i = \Gamma_i \Longrightarrow \Delta_i, \psi_i$ $(i = 1, 2)$.

$\psi_1 \wedge \psi_2$ est une instance d'une sous-formule $\phi_1 \wedge \phi_2$ apparaissant positivement dans ϕ et négativement dans $\neg\phi$. Le renommage de $\phi_1 \wedge \phi_2$ dans $\neg\phi$ introduit donc la définition:

$$(\forall x_1 \ldots x_n(\bigwedge_{i=1}^{2} NP_{\phi_i}[x_1 \ldots x_n]) \rightarrow NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n))$$

et $(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee \neg NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n)) \in CRn(\neg\phi)$.

Par hypothèse d'induction, $CRn(\neg\phi) \vdash C_i$ et $C_i \subseteq C_{\Gamma_i \Longrightarrow \Delta_i, \psi_i}$ $(i = 1, 2)$.

- Si $C_1 \subseteq C_{\Gamma_1 \Longrightarrow \Delta_1}$ alors $C_1 \subseteq C_{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \psi_1 \wedge \psi_2}$.
- Si $C_2 \subseteq C_{\Gamma_2 \Longrightarrow \Delta_2}$ alors $C_2 \subseteq C_{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \psi_1 \wedge \psi_2}$.
- Sinon $\sigma(NP_{\phi_i}[x_1 \ldots x_n]) \subseteq C_i$ $(i = 1, 2)$.
  Posons $C_i = (\sigma(NP_{\phi_i}[x_1 \ldots x_n]) \vee C_i')$ et soit $C' = (C_1' \cup C_2')$.
  On construit à partir de:

  $$\{(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee \neg NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n)), C_1, C_2\}$$

  une dérivation d'une clause incluse dans $C_{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \psi_1 \wedge \psi_2}$ comme suit:
  1. $(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee \neg NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n))$
  2. $(\sigma(NP_{\phi_1}[x_1 \ldots x_n]) \vee C_1')$
  3. $(\sigma(NP_{\phi_2}[x_1 \ldots x_n]) \vee C_2')$
  4. $\sigma(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee \neg NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n))$ (instanciation).
  5. $(\sigma(\neg NP_{\phi_2}[x_1 \ldots x_n]) \vee \sigma(NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n)) \vee C_1')$ (coupure).
  6. $(\sigma(NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n)) \vee C_1' \vee C_2')$ (coupure).

D'où l'existence d'une dérivation à partir de $CRn(\neg\phi)$ d'une clause $C \subseteq C_{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \psi_1 \wedge \psi_2}$.

- $S = \Gamma \Longrightarrow \Delta, \neg\psi_1$ et S est obtenu par application de la règle $\neg - IC$.

  Supposons que $SI(\neg\psi_1, S, \mathcal{D}) =< O, 1, \sigma >$. On a donc:

  $$SI(\psi_1, S', \mathcal{D}) =< O.1, 1, \sigma >$$

en posant $S' = \psi_1, \Gamma \Longrightarrow \Delta$.

$\neg\psi_1$ est une instance d'une sous-formule $\neg\phi_1$ apparaissant positivement dans ϕ et négativement dans $\neg\phi$. Le renommage de $\neg\phi_1$ dans $\neg\phi$ introduit donc la définition :

$$(\forall x_1 \ldots x_n(\neg NP_{\phi_1}(x_1 \ldots x_n)) \rightarrow NP_{\neg\phi_1}(x_1 \ldots x_n))$$

et $(NP_{\phi_1}(x_1 \ldots x_n) \vee NP_{\neg\phi_1}(x_1 \ldots x_n)) \in CRn(\neg\phi)$.

Par hypothèse d'induction, $CRn(\neg\phi) \vdash C$ et $C \subseteq C_{\psi_1,\Gamma\Longrightarrow\Delta}$.

- Si $C \subseteq C_{\Gamma\Longrightarrow\Delta}$ alors $C \subseteq C_{\Gamma\Longrightarrow\Delta,\neg\psi_1}$.
- Sinon $\neg\sigma(NP_{\phi_1}(x_1 \ldots x_n)) \in C$.
  Posons $C = (\neg\sigma(NP_{\phi_1}(x_1 \ldots x_n)) \vee C')$. On construit à partir de:

  $$\{(NP_{\phi_1}(x_1 \ldots x_n) \vee NP_{\neg\phi_1}(x_1 \ldots x_n)), C\}$$

  une dérivation d'une clause incluse dans $C_{\Gamma\Longrightarrow\Delta,\neg\psi_1}$ comme suit:
  1. $(NP_{\phi_1}(x_1 \ldots x_n) \vee NP_{\neg\phi_1}(x_1 \ldots x_n))$
  2. $(\neg\sigma(NP_{\phi_1}(x_1 \ldots x_n)) \vee C')$
  3. $\sigma(NP_{\phi_1}(x_1 \ldots x_n) \vee NP_{\neg\phi_1}(x_1 \ldots x_n))$ (instanciation)
  4. $(\sigma(NP_{\neg\phi_1}(x_1 \ldots x_n)) \vee C')$ (coupure).

D'où l'existence d'une dérivation à partir de $CRn(\neg\phi)$ d'une clause $C \subseteq C_{\Gamma\Longrightarrow\Delta,\neg\psi_1}$.

- $S = \Gamma \Longrightarrow \Delta, \exists x\psi_1$ et S est obtenu par application de la règle $\exists - IC$.

  Supposons que $SI(\exists x\psi_1, S, \mathcal{D}) =< O, 1, \sigma >$. On a donc:

  $$SI(\psi_1(x/t), S', \mathcal{D}) =< O.1, 1, \sigma' >$$

  en posant $S' = \Gamma \Longrightarrow \Delta, \psi_1(x/t)$ et $\sigma' = \{x \mapsto t\} \circ \sigma$.

  On rappelle que $x \notin Var(t)$ et $x \notin Dom(\sigma)$ et donc σ' est une substitution telle que $\sigma'(x) = t$. De plus, $\forall y \in Dom(\sigma), x \notin Var(\sigma(y))$ et donc $\forall y \in Dom(\sigma), \sigma'(y) = \sigma(y)$ et $\sigma'(NP_{\exists x\phi_1}(x_1 \ldots x_n)) = \sigma(NP_{\exists x\phi_1}(x_1 \ldots x_n))$.

  $\exists x\psi_1$ est une instance d'une sous-formule $\exists x\phi_1$ apparaissant positivement dans ϕ et négativement dans $\neg\phi$. Le renommage de $\exists x\phi_1$ dans $\neg\phi$ introduit donc la définition :

  $$(\forall x_1 \ldots x_n(\exists x NP_{\phi_1}[x_1 \ldots x_n, x]) \rightarrow NP_{\exists x\phi_1}(x_1 \ldots x_n))$$

et $(\neg NP_{\phi_1}[x_1 \ldots x_n, x] \vee NP_{\exists x \phi_1}(x_1 \ldots x_n)) \in CRn(\neg\phi)$.

Par hypothèse d'induction, $CRn(\neg\phi) \vdash C$ et $C \subseteq C_{\Gamma \Longrightarrow \Delta, \psi_1(x/t)}$.

- Si $C \subseteq C_{\Gamma \Longrightarrow \Delta}$ alors $C \subseteq C_{\Gamma \Longrightarrow \Delta, \psi_1(x/t)}$.
- Sinon $\sigma'(NP_{\phi_1}[x_1 \ldots x_n, x]) \in C$.
  Posons $C = (\sigma'(NP_{\phi_1}[x_1 \ldots x_n, x]) \vee C')$. On construit à partir de:

$$\{(\neg NP_{\phi_1}[x_1 \ldots x_n, x] \vee NP_{\exists x \phi_1}(x_1 \ldots x_n)), C\}$$

  une dérivation d'une clause incluse dans $C_{\Gamma \Longrightarrow \Delta, \exists x \psi_1}$ comme suit:

  1. $(\neg NP_{\phi_1}[x_1 \ldots x_n, x] \vee NP_{\exists x \phi_1}(x_1 \ldots x_n))$
  2. $(\sigma'(NP_{\phi_1}[x_1 \ldots x_n, x]) \vee C')$
  3. $\sigma'(\neg NP_{\phi_1}[x_1 \ldots x_n, x] \vee NP_{\exists x \phi_1}(x_1 \ldots x_n))$ (instanciation)
  4. $(\sigma(NP_{\exists x \phi_1}(x_1 \ldots x_n)) \vee C')$ (coupure).

D'où l'existence d'une dérivation à partir de $CRn(\neg\phi)$ d'une clause $C \subseteq C_{\Gamma \Longrightarrow \Delta, \exists x \psi_1}$.

- $S = \Gamma \Longrightarrow \Delta, \forall x \psi_1$ et S est obtenu par application de la règle $\forall - IC$.

Supposons que $SI(\forall x \psi_1, S, \mathcal{D}) =< O, 1, \sigma >$. On a donc:

$$SI(\psi_1(x/u), S', \mathcal{D}) =< O.1, 1, \sigma' >$$

en posant $S' = \Gamma \Longrightarrow \Delta, \psi_1(x/u)$ et $\sigma' = \{x \mapsto u\} \circ \sigma$.

On rappelle que $x \notin Dom(\sigma)$ et donc σ' est une substitution telle que $\sigma'(x) = u$. De plus, $\forall y \in Dom(\sigma), x \notin Var(\sigma(y))$. Donc $\forall y \in Dom(\sigma), \sigma'(y) = \sigma(y)$.

$\forall x \psi_1$ est une instance d'une sous-formule $\forall x \phi_1$ apparaissant positivement dans ϕ et négativement dans $\neg\phi$. Le renommage de $\forall x \phi_1$ dans $\neg\phi$ introduit donc la définition :

$$(\forall x_1 \ldots x_n (\forall x NP_{\phi_1}[x_1 \ldots x_n, x]) \rightarrow NP_{\forall x \phi_1}(x_1 \ldots x_n))$$

et $(\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}(x_1 \ldots x_n)) \in CRn(\neg\phi)$ où f_x est un symbole de fonction introduit par skolemisation.

Par hypothèse d'induction, $CRn(\neg\phi) \vdash C$ et $C \subseteq C_{\Gamma \Longrightarrow \Delta, \psi_1(x/u)}$.

- Si $C \subseteq C_{\Gamma \Longrightarrow \Delta}$ alors $C \subseteq C_{\Gamma \Longrightarrow \Delta, \psi_1(x/u)}$.

- Sinon $\sigma'(NP_{\phi_1}[x_1 \ldots x_n, x]) \in C$.

  Posons $C = (\sigma'(NP_{\phi_1}[x_1 \ldots x_n, x]) \vee C')$. Remarquons que $Var(\Gamma) \cup Var(\Delta) \cup Var(\forall x \psi_1) = \cup_{i=1}^{n} Var(\sigma(x_i))$. Comme u n'est pas libre dans $\Gamma, \Delta, \forall x \psi_1$, on a $u \notin Var(\sigma(f_x(x_1 \ldots x_n)))$ et u n'est pas libre dans C'.

  On construit à partir de:

  $$\{(\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}(x_1 \ldots x_n)), C\}$$

  une dérivation d'une clause incluse dans $C_{\Gamma \Longrightarrow \Delta, \forall x \psi_1}$ comme suit:

  1. $(\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}(x_1 \ldots x_n))$
  2. $(\sigma'(NP_{\phi_1}[x_1 \ldots x_n, x]) \vee C')$
  3. $\sigma(\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}(x_1 \ldots x_n))$ (instanciation).
  4. $\{u \mapsto \sigma(f_x(x_1 \ldots x_n))\}(\sigma'(NP_{\phi_1}[x_1 \ldots x_n, x]) \vee C')$ (instanciation).
  5. $(\sigma(NP_{\forall x \phi_1}(x_1 \ldots x_n)) \vee C'$ (coupure).

  D'où l'existence d'une dérivation à partir de $CRn(\neg \phi)$ d'une clause $C \subseteq C_{\Gamma \Longrightarrow \Delta, \forall x \psi_1}$.

On traite de manière similaire les cas correspondant à des séquents obtenus par les règles d'inférence $\rightarrow -IA$, $\wedge - IA$, $\vee - IA'$, $\neg - IA$, $\rightarrow -IC$ ainsi que les règles de contraction, d'affaiblissement ou d'échange.

On obtient ainsi une dérivation par Résolution de $NP_\phi(x_1 \ldots x_n)$ à partir de $CRn(\neg \phi)$ et plus précisément à partir de $CRn(\phi, -1)$, l'ensemble de clauses obtenu par mise sous forme clausale de D_ϕ avec $Rn(\phi, -1) =< NP_\phi(x_1 \ldots x_n), D_\phi >$. Or $CRn(\neg \phi)$ contient les clauses $NP_{\neg \phi}(x_1 \ldots x_n), \neg NP_{\neg \phi}(x_1 \ldots x_n) \vee \neg NP_\phi(x_1 \ldots x_n)$. Comme énoncé dans le théorème, la déduction de $NP_\phi(x_1 \ldots x_n)$ peut être étendue en une réfutation de $CRn(\phi)$ par deux applications de la règle de coupure. En examinant chacun des cas décrits ci-dessus, on vérifie facilement que cette déduction comporte bien $O(n)$ applications de règles d'inférence dont au plus

- $2n + 2$ applications de la règle de coupure
- $2n$ applications de la règle d'instanciation
- n applications de la règle de contraction

si n est le nombre de séquents apparaissant dans la dérivation $\mathcal{D}$. C.Q.F.D.

Ce théorème donne une borne sur le nombre d'inférences utilisées dans la traduction de $\mathcal{D}$ en fonction du nombre d'inférences utilisées dans $\mathcal{D}$. Mais la preuve

présentée ici permet aussi d'estimer la taille (mesurée en nombre total de symboles) de la traduction d'une dérivation $\mathcal{D}$ en fonction de la taille de $\mathcal{D}$.

En effet, pour toute formule ψ_1 apparaissant dans un séquent $\Gamma \Longrightarrow \Delta$ de $\mathcal{D}$ tel que $SI(\psi_1, S, \phi) =< O, p, \sigma >$, $\phi \mid_O = \phi_1$ et $\sigma(\phi_1) = \psi_1$, on montre que:

- $|NP_{\phi_1}(x_1 \ldots x_n)| \leq |\phi_1|$
- $|NP_{\phi_1}(x_1 \ldots x_n)| \leq |\sigma(NP_{\phi_1}(x_1 \ldots x_n))| \leq |\psi_1|$

En tenant compte du fait que les littéraux de $C_{\Gamma \Longrightarrow \Delta}$ correspondant aux formules de Γ sont précédés d'une négation, on a

$$\begin{aligned} |C_{\Gamma \Longrightarrow \Delta}| &\leq |\Gamma \Longrightarrow \Delta| + |\Gamma| \\ &\leq 2|\Gamma \Longrightarrow \Delta| \end{aligned}$$

En appliquant ces inégalités aux déductions figurant dans la preuve du théorème précédent, on montre

Corollaire 2 *Soit $\mathcal{D}$ une dérivation arborescente dans LK' d'une formule ϕ telle que tout axiome de $\mathcal{D}$ soit de la forme $A \Longrightarrow A$ où A est une formule atomique. Alors la taille de la déduction par Résolution de $NP_\phi(x_1 \ldots x_n)$ obtenue par la traduction ci-dessus est au plus de $10|\mathcal{D}|$.*

4.3 Cas général

4.3.1 Extension d'une dérivation en une dérivation dont les axiomes contiennent des formules atomiques

Considérons maintenant le cas d'une dérivation arborescente comportant des axiomes de la forme $\phi \Longrightarrow \phi$ où ϕ n'est pas nécessairement une formule atomique. On utilise alors le lemme suivant:

Lemme 8 *Soit ϕ une formule du premier ordre et soit c la cardinalité de $SF(\phi)$. Alors il existe une dérivation arborescente $\mathcal{D}$ dans LK' du séquent $\phi \Longrightarrow \phi$ telle que*

- *tout axiome de $\mathcal{D}$ est de la forme $A \Longrightarrow A$ où A est une formule atomique;*
- *tout séquent de $\mathcal{D}$ comporte au plus 3 formules*
- *$\mathcal{D}$ comporte au plus $4c$ séquents et $|\mathcal{D}|$ est au plus de $10c|\phi|$.*

Preuve: La preuve se fait par induction sur la structure de la formule ϕ. Dans le cas d'une formule atomique, le lemme est trivialement vérifié. Sinon, on vérifie que, dans chacun des cas ci-dessous, une telle déduction $\mathcal{D}$ de ϕ peut être obtenue à partir des déductions arborescentes des séquents $\phi_i \Longrightarrow \phi_i$ $i = 1, 2$, où ϕ_i désignent les sous-formules directes de ϕ.

- $\phi = \phi_1 \vee \phi_2$

$$\dfrac{\dfrac{\dfrac{\phi_1 \Longrightarrow \phi_1}{\phi_1 \Longrightarrow \phi_1 \vee \phi_2} \quad \dfrac{\phi_2 \Longrightarrow \phi_2}{\phi_2 \Longrightarrow \phi_1 \vee \phi_2}}{\phi_1 \vee \phi_2 \Longrightarrow \phi_1 \vee \phi_2, \phi_1 \vee \phi_2}}{\phi_1 \vee \phi_2 \Longrightarrow \phi_1 \vee \phi_2}$$

- $\phi = \phi_1 \wedge \phi_2$

$$\dfrac{\dfrac{\dfrac{\phi_1 \Longrightarrow \phi_1}{\phi_1 \wedge \phi_2 \Longrightarrow \phi_1} \quad \dfrac{\phi_2 \Longrightarrow \phi_2}{\phi_1 \wedge \phi_2 \Longrightarrow \phi_2}}{\phi_1 \wedge \phi_2, \phi_1 \wedge \phi_2 \Longrightarrow \phi_1 \wedge \phi_2}}{\phi_1 \wedge \phi_2 \Longrightarrow \phi_1 \wedge \phi_2}$$

- $\phi = \phi_1 \rightarrow \phi_2$

$$\dfrac{\dfrac{\dfrac{\phi_1 \Longrightarrow \phi_1 \quad \phi_2 \Longrightarrow \phi_2}{\phi_1 \rightarrow \phi_2, \phi_1 \Longrightarrow \phi_2}}{\phi_1, \phi_1 \rightarrow \phi_2 \Longrightarrow \phi_2}}{\phi_1 \rightarrow \phi_2 \Longrightarrow \phi_1 \rightarrow \phi_2}$$

- $\phi = \neg\phi_1$

$$\dfrac{\dfrac{\dfrac{\phi_1 \Longrightarrow \phi_1}{\neg\phi_1, \phi_1 \Longrightarrow}}{\phi_1, \neg\phi_1 \Longrightarrow}}{\neg\phi_1 \Longrightarrow \neg\phi_1}$$

- $\phi = \forall x \phi_1$

$$\dfrac{\dfrac{\phi_1(x) \Longrightarrow \phi_1(x)}{\forall x \phi_1 \Longrightarrow \phi_1(x)}}{\forall x \phi_1 \Longrightarrow \forall x \phi_1}$$

- $\phi = \exists x \phi_1$

$$\dfrac{\dfrac{\phi_1(x) \Longrightarrow \phi_1(x)}{\phi_1(x) \Longrightarrow \exists x \phi_1}}{\exists x \phi_1 \Longrightarrow \exists x \phi_1}$$

C.Q.F.D.

Nous pouvons maintenant affirmer que:

Théorème 7 *Soit $\mathcal{D}$ une dérivation arborescente dans LK' d'une formule ϕ. Alors on peut construire à partir de $CRn(\phi, -1)$ une déduction $\mathcal{R}$ par Résolution de $NP_\phi(x_1 \ldots x_n)$ telle que:*

- *$\mathcal{R}$ comporte $O(n|SF(\phi)|)$ inférences si $\mathcal{D}$ comporte n séquents.*
- *$|R|$ est $O(|\mathcal{D}||SF(\phi)|)$.*

Cette déduction peut être étendue en une réfutation de $CRn(\phi)$ en 2 applications de la règle de coupure.

Preuve: Considérons une telle déduction $\mathcal{D}$. Cette déduction comporte $p \leq n$ séquents de la forme $\psi_i \Longrightarrow \psi_i$ où ψ_i, $1 \leq i \leq p$ est une formule non atomique. De telles formules ψ_i sont des instances de sous-formules de ϕ (cf. propriété de la sous-formule) et donc $|SF(\psi_i)| \leq |SF(\phi)|$ $1 \leq i \leq p$.

De plus,

$$\sum_{i=1}^{p} |\psi_i| \ \leq \ \frac{1}{2} \sum_{i=1}^{p} |\psi_i \Longrightarrow \psi_i| \ \leq \ \frac{1}{2} |\mathcal{D}|$$

En utilisant les résultats du lemme précédent, la déduction $\mathcal{D}$ peut être étendue en une déduction arborescente $\mathcal{D}'$ dont les axiomes sont de la forme $A \Longrightarrow A$ où A est une formule atomique. La déduction $\mathcal{D}'$ ainsi obtenue comporte au plus $n + 4n|SF(\phi)|$ séquents et $|\mathcal{D}'| \leq |\mathcal{D}| + 5n|\mathcal{D}||SF(\phi)|$. On peut alors appliquer le théorème et le corollaire précédents à cette déduction arborescente pour obtenir le résultat voulu. C.Q.F.D.

4.3.2 Dérivations non arborescentes

Nous considérons maintenant le cas d'une dérivation non arborescente de LK'. Nous ne traiterons pas ce cas en détail; nous nous contenterons de discuter quelles sont les hypothèses qui permettent de généraliser les résultats des sections précédentes à ce type de dérivation.

Bien entendu, il est toujours possible de transformer une dérivation $\mathcal{D}$ dans LK' d'une formule ϕ en une dérivation arborescente $\mathcal{D}'$ dans LK' de ϕ et d'appliquer la transformation décrite au paragraphe précédent à la dérivation $\mathcal{D}'$. Cependant, il existe une infinité de formules ϕ admettant une dérivation dans LK' comportant n inférences pour lesquelles toute dérivation arborescente de ϕ comporte au moins $2^n - 1$ inférences. La famille de formules définies par

- $\phi_1 = (A \rightarrow A)$
- $\phi_{n+1} = (\phi_n \wedge \phi_n)$

est un exemple de telles formules. Cette transformation des preuves de LK' en preuves par Résolution peut donc produire une preuve dont le nombre d'inférences (et donc la taille) est exponentiellement plus grand que celui de la preuve initiale. Nous allons montrer que l'on peut obtenir des résultats similaires à ceux de la section précédente en *modifiant la définition d'un renommage.*

Rappelons que dans une dérivation non arborescente, un même séquent peut être utilisé plusieurs fois comme prémisse d'une règle d'inférence. On remarque alors

que la notion de schéma d'instanciation telle qu'elle a été définie jusqu'à présent ne s'applique plus ici car, étant donné une dérivation $\mathcal{D}$ de $\Longrightarrow \phi$, une formule ψ apparaissant dans un séquent $\Gamma \Longrightarrow \Delta$ de $\mathcal{D}$ peut être une instance de plusieurs occurrences distinctes de sous-formules de ϕ. Par exemple, considérons la déduction constituée des séquents:

1. $p(a,b) \Longrightarrow p(a,b)$ (axiome)
2. $\Longrightarrow p(a,b) \rightarrow p(a,b)$ ($\rightarrow -IC$ avec 1)
3. $\Longrightarrow \exists x(p(a,x) \rightarrow p(a,x))$ ($\exists - IC$ avec 2)
4. $\Longrightarrow \exists y(p(y,b) \rightarrow p(y,b))$ ($\exists - IC$ avec 2)
5. $\Longrightarrow (\exists x(p(a,x) \rightarrow p(a,x))) \wedge (\exists y(p(y,b) \rightarrow p(y,b)))$ ($et - IC$ avec 3,4)

La formule $p(a,b) \rightarrow p(a,b)$ correspond à la fois à $(p(a,x) \rightarrow p(a,x))(x/b)$ et à $(p(y,b) \rightarrow p(y,b))(y/a)$.

La déduction arborescente correspondant à cette déduction est constituée des séquents:

1. $p(a,b) \Longrightarrow p(a,b)$ (axiome)
2. $\Longrightarrow p(a,b) \rightarrow p(a,b)$ ($\rightarrow -IC$ avec 1)
3. $\Longrightarrow \exists x(p(a,x) \rightarrow p(a,x))$ ($\exists - IC$ avec 2)
4. $p(a,b) \Longrightarrow p(a,b)$ (axiome)
5. $\Longrightarrow p(a,b) \rightarrow p(a,b)$ ($\rightarrow -IC$ avec 4)
6. $\Longrightarrow \exists y(p(y,b) \rightarrow p(y,b))$ ($\exists - IC$ avec 5)
7. $\Longrightarrow (\exists x(p(a,x) \rightarrow p(a,x))) \wedge (\exists y(p(y,b) \rightarrow p(y,b)))$ ($\wedge - IC$ avec 3,6)

Dans cette séquence, les séquents numérotés 2 et 5 sont identiques. Le premier correspond à une instance de la sous-formule $p(a,x) \rightarrow p(a,x)$; le second correspond à une instance de la sous-formule $p(y,b) \rightarrow p(y,b)$. La technique de traduction développée jusqu'à présent permet d'associer à chaque séquent $\Gamma \Longrightarrow \Delta$ d'une dérivation dans LK', une clause $C \subseteq C_{\Gamma \Longrightarrow \Delta}$. Soit C^2 (resp. C^5) la clause ainsi associée au séquent numéro 2 (resp. 5). Ces clauses sont des clauses unitaires. Cependant, il est important de constater ici que, le renommage étant une notion

définie relativement à une *occurrence* d'une sous-formule, les clauses C^2, C^5 sont des clauses dont les littéraux sont *distincts* car les séquents 2 et 5 font référence à des sous-formules distinctes de $(\exists x(p(a,x) \rightarrow p(a,x))) \wedge (\exists y(p(y,b) \rightarrow p(y,b)))$.

Afin de pouvoir réutiliser les résultats de la section précédente, il est nécessaire de pouvoir *renommer de "manière identique" deux occurrences distinctes de sous-formules de même polarité ayant une instance commune.*

Dans le cas du calcul propositionnel, ceci revient à appliquer la technique du renommage à une formule ϕ représentée non pas comme un arbre mais comme un *graphe orienté acyclique* (encore appelé DAG).

Dans le cas du calcul des prédicats du premier ordre, renommer de "manière identique" deux occurrences distinctes de sous-formules de même polarité ayant une instance commune nécessite en fait une *généralisation.* Dans l'exemple précédent, on remarque que les deux formules $(p(a,x) \rightarrow p(a,x)), (p(y,b) \rightarrow p(y,b))$ sont des instances de la formule plus générale $p(x_1,y_1) \rightarrow p(x_2,y_2)$. Nous allons tirer parti du fait que, si l'on introduit un nouveau symbole de prédicat NP défini par:

$$\forall x_1, y_1, x_2, y_2 NP(x_1,y_1,x_2,y_2) \leftrightarrow (p(x_1,y_1) \rightarrow p(x_2,y_2))$$

on obtient alors que

$$\begin{array}{c}(\exists x\ p(a,x) \rightarrow p(a,x)) \wedge (\exists y\ p(y,b) \rightarrow p(y,b)) \\ \leftrightarrow \\ (\exists x NP(a,x,a,x)) \wedge (\exists y NP(y,b,y,b))\end{array}$$

est une formule valide pour définir un nouveau renommage.

Nous introduisons la terminologie suivante:

Définition: Une formule ϕ_1 est une *généralisation* d'une formule ϕ_2 si il existe une substitution σ telle que $\phi_2 = \sigma(\phi_1)$. De plus, ϕ_1 est appelée généralisation *linéaire* si:

- toutes les sous-formules atomiques de ϕ_1 sont de la forme $p(y_1 \ldots y_n)$ où y_i $i = 1, \ldots, n$ sont des variables n'apparaissant pas liées dans ϕ_1;
- chaque variable libre de ϕ_1 apparaît exactement une seule fois dans ϕ_1.

$\diamond$

Il est aisé de calculer la généralisation linéaire d'une formule. Soit *gen* la fonction définie sur les formules du premier ordre par:

- $gen(p(t_1 \ldots t_n)) =< p(y_1 \ldots y_n), < t_1 \ldots t_n >, < y_1 \ldots y_n >>$
  où y_i $i = 1, \ldots, n$ sont des nouvelles variables deux à deux distinctes et $p(t_1 \ldots t_n)$ désigne une formule atomique dont le symbole de prédicat p est d'arité n.

- $gen(*_{i=1}^{p}\phi_i) =< *_{i=1}^{p}\phi_i, S_1^T \ldots S_p^T, S_1^V \ldots S_p^V >$
  si $gen(\phi_i) =< \phi_i', S_i^T, S_i^V >$ $i = 1, \ldots, p$ avec la convention que $*$ désigne soit $\vee$ ou $\wedge$, soit $\rightarrow$ ou $\leftrightarrow$ et $n = 2$, soit $\neg$ et $n = 1$.

- $gen(Qx\phi_1) =< Qx\phi_1', S_1^T, S_1^V >$ si $gen(\phi_1) =< \phi_1', S_1^T, S_1^V >$ avec la convention que Q désigne soit $\forall$ soit $\exists$.

Soit ϕ une formule telle que $gen(\phi) =< \phi', S^T, S^V >$. Alors ϕ' est une généralisation linéaire de ϕ et S^T et S^V désignent des séquences de même longueur dont les éléments sont des termes. En outre, tous les éléments de S^V sont des variables deux à deux distinctes. Posons $S^T =< t_1 \ldots t_n >, S^V =< y_1 \ldots y_n >$ et définissons σ comme étant la substitution telle que $Dom(\sigma) = \{y_1 \ldots y_n\}$ et $\sigma(y_i) = t_i$, $i = 1, \ldots, n$. On a:

$$\phi = \sigma(\phi')$$

Soit $\succeq$ le quasi-ordre défini sur les formules par:

$$\phi_1 \succeq \phi_2 \text{ ssi } \phi_1 \text{ est une généralisation de } \phi_2.$$

Soit $\approx$ la relation d'équivalence définie par:

$$\phi_1 \approx \phi_2 \text{ ssi } \phi_1 \succeq \phi_2 \text{ et } \phi_2 \succeq \phi_1$$

On établit facilement que deux généralisation linéaires d'une même formule sont identiques à un renommage de leurs variables libres près:

Lemme 9 *Soient ϕ_1, ϕ_2 deux généralisations linéaires d'une formule ϕ. Alors*

$$\phi_1 \approx \phi_2$$

La propriété suivante est très importante: elle établit que deux formules ayant des instances communes ont les mêmes généralisations linéaires:

Lemme 10 *Soient ϕ_1, ϕ_2 deux formules du premier ordre et soient deux substitutions σ_1, σ_2 telles que $\sigma_1(\phi_1) = \sigma_2(\phi_2)$. Soient ψ_1, ψ_2 des généralisations linéaires de respectivement ϕ_1, ϕ_2. Alors*

$$\psi_1 \approx \psi_2$$

Chacun de ces lemmes peut être établi facilement par une induction sur la structure des formules considérées. Désormais, étant donnée une formule ϕ, nous noterons $\bar{\phi}$ une généralisation linéaire de ϕ. Deux formules ϕ_1, ϕ_2 sont dites *structurellement équivalentes* si $\bar{\phi_1} \approx \bar{\phi_2}$.

On définit alors le renommage d'une formule ψ dans une formule ϕ comme suit:

Définition: Soient ϕ, ψ_1, ψ_2 des formules du premier ordre. On désigne alors par $\phi[\psi_1 \downarrow^p \psi_2]$ la formule obtenue en remplaçant dans ϕ toute sous-formule de ϕ de polarité p ayant la forme $\sigma(\psi_1)$ par $\sigma(\psi_2)$. Plus précisément, soit $\{O_1 \ldots O_n\}$ l'ensemble des occurrences des sous-formules de ϕ de polarité p telle qu'il existe une substitution σ_i pour laquelle $\phi \mid_{O_i} = \sigma_i(\psi_1)$. Alors

$$\phi[\psi_1 \downarrow^p \psi_2] = \phi[O_1 \leftarrow \sigma_1(\psi_2), \ldots, O_n \leftarrow \sigma_n(\psi_2)]$$

$\diamond$

Définition: Soit $\bar{\psi}$ une généralisation linéaire d'une formule ψ telle que $\{x_1 \ldots x_n\}$ l'ensemble des variables libres de $\bar{\psi}$ ne contienne pas de variable apparaissant dans une formule ϕ. Le *renommage des sous-formules structurellement équivalentes à ψ dans ϕ* est la formule

$$\phi[\bar{\psi} \downarrow^p NP_{\bar{\psi}}(x_1 \ldots x_n)] \wedge \forall x_1 \ldots x_n NP_{\bar{\psi}}(x_1 \ldots x_n) \rightleftharpoons^p \bar{\psi}$$

où $NP_{\bar{\psi}}$ est un nouveau symbole de prédicat d'arité n appelé *prédicat de Skolem.* On notera cette formule $Rn_{\approx}(\phi, \bar{\psi})$. $\diamond$

On vérifie que, avec cette nouvelle définition, le renommage demeure une transformation qui préserve la satisfaisabilité. On définit maintenant le renommage exhaustif des sous-formules d'une formule ϕ.

Définition: Soit $GLSF(\phi)$ l'ensemble des généralisations linéaires des sous-formules de ϕ et soit $GLSF_{\approx}(\phi)$ l'ensemble quotient de $GLSF(\phi)$ par la relation $\approx$. (On identifie deux généralisations linéaires identiques à un renommage de leurs variables libres près.) Soit $< \psi_1 \ldots \psi_n >$ un énumération des éléments de $GLSF_{\approx}(\phi)$ tel que:

$$\forall 1 \leq i < j \leq n, |\psi_i| > |\psi_j|$$

Alors le renommage exhaustif des sous-formules de ϕ est défini par:

$$Rn_{\approx}((\ldots(Rn_{\approx}(\phi, \psi_1))\ldots), \psi_n)$$

On le note $Rn_{\approx}(\phi)$. $\diamond$

Notons $CRn_{\approx}(\phi)$ le résultat de la mise sous forme clausale de $Rn_{\approx}(\phi)$. Par analogie avec ce qui a été fait précédemment, on montre les résultats de complexité suivants:

$$|Rn_{\approx}(\phi)| \leq (3N+5)|SF(\phi)| + |\phi|$$

$$|CRn_{\approx}(\phi)| \leq (16N+35)|SF(\phi)| + |\phi|$$

où N désigne le nombre maximal de variables libres apparaissant dans une généralisation linéaire d'une sous-formule de ϕ. Ce nombre est inférieur à la somme des arités des symboles de prédicats des sous-formules atomiques de ϕ. Donc $N \leq |\phi|$. En fait, $|Rn_{\approx}(\phi)|, |CRn_{\approx}(\phi)|$ sont en $O(|\phi|^2)$.

En reprenant ce qui a été fait au paragraphe précédent pour les déductions arborescentes, on montre que:

Théorème 8 *Etant donné une dérivation $\mathcal{D}$ dans LK' d'une formule ϕ, il existe une réfutation $\mathcal{R}$ par Résolution de $CRn_{\approx}(\phi)$ comportant $O(n|SF(\phi)|)$ inférences et telle que $|\mathcal{R}|$ est $O(|\mathcal{D}||SF\phi|)$ si n est le nombre de séquents de $\mathcal{D}$*

Remarque 6 *Même si en théorie l'utilisation du renommage $CRn_{\approx}$ permet de trouver des preuves exponentiellement plus courtes que celles obtenues avec le renommage CRn, l'utilisation en pratique de ce renommage peut être discutable. En effet, le calcul des généralisations linéaires conduit à créer des formules avec de nombreuses variables libres. Ceci a pour conséquence une augmentation du nombre de littéraux a priori unifiables, ce qui augmente l'espace de recherche d'une réfutation.*

4.4 Conclusion

L'existence d'une simulation polynomiale du calcul LK' par la Résolution nous a été suggérée par les résultats de [Tse68]. Dans cet article, Tseitin indique de manière succincte et informelle comment obtenir une preuve par Résolution à partir d'une preuve dans un calcul des séquents dans le cadre du calcul propositionnel.

Une autre preuve formelle de l'existence d'une simulation polynomiale d'un calcul des séquents par la Résolution est celle trouvée indépendamment de la nôtre par E. Eder [Ede90]. Dans cet article, Eder compare le principe de Résolution utilisant la résolution binaire et la factorisation avec le calcul des séquents LK de Gentzen. Eder utilise un renommage différent du nôtre que nous pouvons décrire par:

Définition: Etant donné un langage du premier ordre $\mathcal{L}$, soit ϕ une $\mathcal{L}$-formule et soit ψ une sous-formule de ϕ telle que:

- ψ apparaît à l'occurrence O dans ϕ;
- $\{x_1 \ldots x_n\}$ sont les variables libres de ψ

Le *renommage* de ψ dans ϕ est la formule

$$\phi[O \leftarrow NP_\psi(x_1 \ldots x_n)] \wedge Def(\psi, \phi)$$

où

- $NP_\psi \notin \mathcal{L}$ est un nouveau symbole de prédicat d'arité n, associé à l'occurrence O de ϕ et appelé *prédicat de Skolem.*
- $Def(\psi, \phi)$, la *définition* du *littéral de Skolem* $NP_\psi(x_1 \ldots x_n)$ dans ϕ dénote la formule suivante:

$$\forall x_1 \ldots x_n (NP_\psi(x_1 \ldots x_n) \leftrightarrow \psi)$$

$\diamond$

Lors de la traduction des dérivations arborescentes, nous avons utilisé une forme restreinte de ce renommage (avec une implication au lieu d'une équivalence dans la définition du littéral de Skolem). Pour de telles déductions, les résultats obtenus par Eder et les nôtres sont du même ordre de grandeur. Pour ce qui est des déductions non arborescentes, nous avons eu recours à un autre renommage (utilisant la notion de généralisation linéaire) alors qu'Eder établit un résultat de simulation polynomiale avec la même transformation que celle mentionnée plus haut: la taille de la réfutation par Résolution ainsi obtenue est alors en $O(|\mathcal{D}|^3)$ et comporte $O(|\mathcal{D}|^2)$ clauses.

Remarque 7 *Le renommage qu'utilise E.Eder conduit à une forme clausale qui, en pratique, est souvent difficile à réfuter par un démonstrateur de théorèmes utilisant la Résolution. En effet, la présence d'une équivalence dans la définition d'un littéral de Skolem augmente à la fois le nombre de clauses obtenues et le nombre de littéraux potentiellement unifiables. Il en résulte une augmentation sensible de l'espace de recherche d'une réfutation. Il y a donc intérêt à utiliser le renommage exhaustif que nous avons présenté au chapitre 1.*

Les résultats obtenus dans ce chapitre sont importants car ils montrent que l'on peut simuler pas à pas le calcul LK' par la Résolution à condition d'utiliser le renommage exhaustif. Il est donc a priori possible de reproduire dans le cadre de

la Résolution les méthodes de raisonnement des calculs en Déduction Naturelle. De plus, l'étude de la taille des dérivations par Résolution obtenues par traduction des preuves de LK' montre qu'en fait, cette simulation s'opère sans perte d'efficacité.

Pour être d'un réel intérêt pratique, ces résultats demeurent néanmoins insuffisants: nous ne disposons que d'un résultat d'existence d'une dérivation par Résolution à partir de $CRn(\neg\phi)$ permettant de simuler une dérivation donnée de $\Longrightarrow \phi$ dans le calcul LK'. Il nous reste encore

- à déterminer s'il existe des stratégies suffisamment restrictives de Résolution permettant de trouver des preuves par Résolution correspondant à la traduction de preuves en Déduction Naturelle. Nous apporterons une réponse à ce problème au chapitre suivant en étudiant comment traduire les réfutations par Résolution de $CRn(\neg\phi)$ en des preuves de la validité de ϕ dans le calcul des séquents.

- à juger de l'efficacité réelle de cette technique par rapport à l'utilisation de la forme clausale standard ou à l'utilisation de procédures de preuves mettant directement en œuvre des calculs en Déduction Naturelle. Nous apporterons une réponse partielle à ces questions au chapitre 6 en considérant divers résultats permettant d'améliorer l'efficacité de la Résolution.

Chapitre 5

Traduction de preuves par Résolution en preuves par Déduction Naturelle

Etant données une formule valide ϕ et une réfutation par Résolution de $CRn(\neg\phi)$, nous nous intéressons désormais au problème de construire une dérivation de ϕ dans le calcul LK'.

Dans le chapitre précédent, nous avons vu comment transcrire une dérivation de ϕ dans LK' en une dérivation de $NP_\phi(x_1 \ldots x_n)$ à partir de $CRn(\neg\phi)$. Cette traduction repose sur le fait que, à tout séquent $\Gamma \Longrightarrow \Delta$ de la dérivation de ϕ dans LK', on peut associer une dérivation par Résolution d'une clause $C \subseteq C_{\Gamma \Longrightarrow \Delta}$. Aussi est-il naturel d'envisager la traduction inverse en interprétant chacune des clauses de la dérivation de NP_ϕ comme une clause de la forme $C_{\Gamma \Longrightarrow \Delta}$ et d'associer à cette clause une dérivation d'un séquent $\Gamma \Longrightarrow \Delta$ dans le système de déduction naturelle LK'.

Dans un premier temps, nous donnerons une traduction des preuves par

Résolution dans le calcul $LK' + Cut$. Puis, nous améliorerons cette traduction en montrant que de nombreuses applications de la règle de coupure dans la dérivation ainsi obtenue peuvent en fait être éliminées et remplacées par des règles d'introduction de connectifs ou de quantificateurs sans qu'il en résulte une augmentation de la taille de la preuve dans LK. cette nouvelle traduction permet d'obtenir des preuves plus naturelles et plus lisibles. La traduction de dérivations obtenues à partir de clauses contenant des fonctions de Skolem ne pourra être assurée que pour certaines classes de dérivations: les dérivations par Résolution Sémantique Ordonnée. Cependant, l'existence de théorèmes de normalisation pour la Résolution nous permet d'assurer que de toute réfutation par Résolution de $CRn(\neg\phi)$ on peut extraire une preuve sans coupure de la validité de ϕ dans le calcul des séquents. Puisque une telle preuve s'obtient par l'intermédiaire des théorèmes de normalisation, la structure de la preuve en Déduction Naturelle ne reflètera pas forcément celle de la preuve par Résolution. Par contre, si l'on considère la traduction de dérivations obtenues à partir de clauses *ne contenant pas* de fonction de Skolem, la structure des preuves en Déduction Naturelle sera le reflet de la structure des preuves par Résolution. Ceci constitue donc une nette amélioration par rapport à l'approche proposée dans [Pfe84].

5.1 Préliminaires

On supposera sans perte de généralité que ϕ est une formule fermée: en effet, on montre facilement que si $Var(\phi) = \{x_1 \ldots x_n\}$, alors ϕ est valide si et seulement si $\forall x_1 \ldots x_n \phi$ est valide. De plus on supposera que l'on possède une dérivation par Résolution de NP_ϕ à partir de $CRn(\phi, -1)$. Une telle dérivation peut toujours être obtenue comme le montre le raisonnement suivant.

Puisque ϕ est une formule valide, la formule $\neg\phi$ est insatisfaisable et il existe une réfutation $\mathcal{R}$ de l'ensemble de clauses $CRn(\neg\phi)$. Rappelons que $CRn(\neg\phi) = CRn(\phi, -1) \cup \{NP_{\neg\phi}, \neg NP_{\neg\phi} \vee \neg NP_\phi\}$. Supposons que $NP_{\neg\phi}$ ne soit pas utilisé comme prémisse d'une règle d'inférence dans $\mathcal{R}$: $\mathcal{R}$ est alors une réfutation de l'ensemble de clauses: $CRn(\phi, -1) \cup \{\neg NP_{\neg\phi} \vee \neg NP_\phi\}$. Cependant, aucune clause de $CRn(\phi, -1)$ ne contient une occurrence positive de $NP_{\neg\phi}$ et donc le littéral $\neg NP_{\neg\phi}$ de la clause $\neg NP_{\neg\phi} \vee \neg NP_\phi$ est pur et cette clause n'est pas utilisée comme prémisse d'une règle d'inférence. $\mathcal{R}$ est donc une réfutation de $CRn(\phi, -1)$. Par un raisonnement analogue au précédent, on montre que NP_ϕ est un littéral pur

dans $CRn(\phi, -1)$, et en itérant l'argument précédent sur les sous-formules de ϕ, on montre que $\mathcal{R}$ est une réfutation de l'ensemble vide de clause, ce qui conduit à une contradiction car cet ensemble est toujours satisfaisable. Donc nécessairement, les clauses $NP_{\neg\phi}, \neg NP_{\neg\phi} \vee \neg NP_{\phi}$ sont utilisées dans $\mathcal{R}$. Selon une technique de preuve bien connue en démonstration automatique et que nous ne formaliserons pas ici ([CL73], [Lov78]), soit $\mathcal{R}'$ la dérivation obtenue à partir de $\mathcal{R}$ en "effaçant" les clauses $NP_{\neg\phi}, \neg NP_{\neg\phi} \vee \neg NP_{\phi}$: $\mathcal{R}'$ est une dérivation de NP_{ϕ}.

Enfin, nous utiliserons sans le montrer le lemme suivant:

Lemme 11 *Soit V un ensemble fini de variables. Etant donnée une dérivation $\mathcal{R}$ d'une clause C, on peut toujours, en utilisant des substitutions de renommage, transformer $\mathcal{R}$ en une déduction $\mathcal{R}'$ telle que*

- *$\mathcal{R}$ et $\mathcal{R}'$ sont des séquences de clauses de même taille (disons n);*
- *la i-ème clause de $\mathcal{R}'$ est une variante de la i-ème clause de $\mathcal{R}$;*
- *toute application de la règle d'instanciation $\frac{C}{\sigma(C)}$ est telle que*

$$\forall y \in Dom(\sigma), Var(\sigma(y)) \cap V = \emptyset$$

et

$$\forall y \in Dom(\sigma), Var(\sigma(y)) \cap Dom(\sigma) = \emptyset.$$

Nous considérerons par la suite que des dérivations par Résolution à partir de $CRn(\phi)$ telle que si V_{ϕ} désigne l'ensemble des variables quantifiées dans ϕ, alors toute application de la règle d'instanciation $\frac{C}{\sigma(C)}$ est telle que

$$\forall y \in Dom(\sigma), Var(\sigma(y)) \cap V_{\phi} = \emptyset$$

$$\forall y \in Dom(\sigma), Var(\sigma(y)) \cap Dom(\sigma) = \emptyset.$$

Enfin, étant donnés une dérivation $\mathcal{D}$ par Résolution de NP_{ϕ}, un terme t et une variable v telle que v n'apparaît dans aucune clause de $\mathcal{D}$, notons $\mathcal{D}(t/v)$ la séquence de clauses obtenue en remplaçant toute occurrence de t dans une clause de $\mathcal{D}$ par la variable v. Alors, on montre que $\mathcal{D}(t/v)$ est aussi une dérivation par Résolution de NP_{ϕ}. Appelons *terme de Skolem* tout terme non variable dont le symbole de tête est un symbole de fonction de Skolem et soit $< t_1 \ldots t_p >$ une énumération des termes de Skolem apparaissant dans $\mathcal{D}$. On suppose que cette énumération vérifie que t_i n'est pas un sous-terme de t_j pour tout i, j tels que $1 \leq i < j \leq p$. La

dérivation $(\ldots(\mathcal{D}(t_1/v_1))\ldots)(t_p/v_p)$ où les variables v_i sont deux à deux distinctes et distinctes des variables apparaissant dans la déduction $\mathcal{D}$ et dans ϕ est appelée la *dérivation dé-Skolemisée* associée à $\mathcal{D}$.

5.2 Traduction des preuves par Résolution dans le calcul $LK' + Cut$

Cette approche n'est possible que si aucune clause de $CRn(\phi, -1)$ ne contient de fonction de Skolem, ou de manière équivalente, si ϕ ne contient pas de sous-formule $\forall x \phi_1$ de polarité positive ni de sous-formule $\exists x \phi_1$ de polarité négative. En effet, dans ce cas, on montre qu'il est possible d'associer à chacune des clauses de l'ensemble initial de clauses $CRn(\phi, -1)$ une dérivation d'un séquent dans LK'. Puis, les règles clausales de coupure et de contraction du principe de Résolution sont interprétées comme les règles de coupure et de contraction du calcul des séquents $LK' + cut$ tandis que la règle d'instanciation sera prise en compte en utilisant les lemmes qui permettent de construire à partir d'une dérivation $\mathcal{D}$ d'un séquent $\Gamma \Longrightarrow \Delta$ une dérivation du séquent $\Gamma(x/t) \Longrightarrow \Delta(x/t)$. On obtient ainsi une dérivation dans le calcul $LK' + cut$ de la formule ϕ. Par application du théorème d'élimination des coupures [Gen69], on peut transformer cette dérivation en une dérivation sans coupure de ϕ.

Définition: Soit ϕ une formule du premier ordre et soit A_ϕ la réunion de l'ensemble des sous-formules atomiques de ϕ et l'ensemble des atomes de Skolem $NP_{\phi_1}(x_1 \ldots x_n)$ associés au renommage d'une occurrence d'une sous-formule ϕ_1 de ϕ. On définit alors l'application $t^{\phi}_{A \to F} : A_\phi \to SF(\phi)$ par:

- $t^{\phi}_{A \to F}(A) = A$ si A est une sous-formule atomique de ϕ;
- $t^{\phi}_{A \to F}(A) = \phi_1$ si $A = NP_{\phi_1}(x_1 \ldots x_n)$ est un atome de Skolem associé au renommage d'une occurrence d'une sous-formule ϕ_1 de ϕ.

On définit alors les applications S^+, S^- qui associent à une séquence de littéraux dont les atomes sont des instances d'éléments de A_ϕ une séquence d'instances de

sous-formules de ϕ par:

$$\begin{array}{lcl} S^+(<>) & = & <> \\ S^+(<\sigma(A)>.S) & = & <\sigma(t^{\phi}_{A\to F}(A))>.S^+(S) \\ S^+(<\neg\sigma(A)>.S) & = & S^+(S) \\ S^-(<>) & = & <> \\ S^-(<\neg\sigma(A)>.S) & = & <\sigma(t^{\phi}_{A\to F}(A))>.S^-(S) \\ S^-(<\sigma(A)>.S) & = & S^-(S) \end{array}$$

Soit C_ϕ l'ensemble des clauses dont chaque atome est une instance d'un atome de A_ϕ. Soit S_ϕ l'ensemble de séquents dont chaque formule est une instance d'une sous-formule de ϕ. On définit alors l'application $t^{\phi}_{C\to S} : C_\phi \to S_\phi$ par:

$$t^{\phi}_{C\to S}(L_1 \vee \ldots \vee L_n) = S^-(<L_1 \ldots L_n>) \Longrightarrow S^+(<L_1 \ldots L_n>).$$

$\diamond$

On peut alors énoncer les lemmes suivants:

Lemme 12 *Soit ϕ une formule ne comportant pas de sous-formule $\exists x\phi_1$ de polarité négative ni de sous-formule $\forall x\phi_1$ de polarité positive. Soit C une clause de $CRn(\phi,-1)$. Alors $t^{\phi}_{C\to S}(C)$ est bien défini et $t^{\phi}_{C\to S}(C)$ est dérivable dans LK'.*

Preuve: Etant donnée une telle formule ϕ, on montre que chacune des clauses de $CRn(\phi,-1)$ est de la forme de celles décrites dans le tableau de la figure 5.1. Pour chacune de ces clauses C, on vérifie trivialement que $t^{\phi}_{C\to S}(C)$ est bien défini et que $t^{\phi}_{C\to S}(C)$ est dérivable dans LK'. C.Q.F.D.

Lemme 13 *Soit $C = (C'_1 \vee C'_2)$ une clause obtenue par coupure à partir des clauses $C_1 = (C'_1 \vee \sigma(A))$, $C_2 = (C'_2 \vee \neg\sigma(A))$ telles que $A \in A_\phi$ et $t^{\phi}_{C\to S}(C_i)$ soit bien défini et dérivable dans LK' ($i = 1,2$). Alors $t^{\phi}_{C\to S}(C)$ est bien défini et $t^{\phi}_{C\to S}(C)$ est dérivable dans $LK' + cut$ et donc dans LK'.*

Preuve: Puisque $t^{\phi}_{C\to S}(C_i) i = 1,2$ est bien défini, chaque littéral de C_i a pour atome une instance d'un élément de A_ϕ. Donc chaque atome d'un littéral de C est aussi une instance d'un élément de A_ϕ et $t^{\phi}_{C\to S}(C)$ est bien défini. Soit $\mathcal{D}_i$ la dérivation dans LK' de $t^{\phi}_{C\to S}(C_i)$. On construit alors la dérivation suivante de $t^{\phi}_{C\to S}(C)$ dans

Figure 5.1: LK'-Dérivation de $t^{\phi}_{C\to S}(C)$

Clause C de $CRn(\phi,-1)$	LK'-Dérivation de $t^{\phi}_{C\to S}(C)$
$\neg NP_A(x_1\dots x_n)\vee A$	$A\Longrightarrow A$
$\neg NP_{\phi_1\wedge\phi_2}(x_1\dots x_n)\vee NP_{\phi_1}[x_1\dots x_n]$	$\dfrac{\phi_1\Longrightarrow\phi_1}{\phi_1\wedge\phi_2\Longrightarrow\phi_1}$
$\neg NP_{\phi_1\wedge\phi_2}(x_1\dots x_n)\vee NP_{\phi_2}[x_1\dots x_n]$	$\dfrac{\phi_2\Longrightarrow\phi_2}{\phi_1\wedge\phi_2\Longrightarrow\phi_2}$
$\neg NP_{\phi_1\vee\phi_2}(x_1\dots x_n)\vee$ $NP_{\phi_1}[x_1\dots x_n]\vee NP_{\phi_2}[x_1\dots x_n]$	$\dfrac{\phi_1\Longrightarrow\phi_1\quad\phi_2\Longrightarrow\phi_2}{\phi_1\vee\phi_2\Longrightarrow\phi_1,\phi_2}$
$\neg NP_{\neg\phi_1}(x_1\dots x_n)\vee\neg NP_{\phi_1}(x_1\dots x_n)$	$\dfrac{\phi_1\Longrightarrow\phi_1}{\neg\phi_1,\phi_1\Longrightarrow}$
$\neg NP_{\phi_1\to\phi_2}(x_1\dots x_n)\vee$ $\neg NP_{\phi_1}[x_1\dots x_n]\vee NP_{\phi_2}[x_1\dots x_n]$	$\dfrac{\phi_1\Longrightarrow\phi_1\quad\phi_2\Longrightarrow\phi_2}{\phi_1\to\phi_2,\phi_1\Longrightarrow\phi_2}$
$\neg NP_{\forall x\phi_1}(x_1\dots x_n)\vee NP_{\phi_1}[x_1\dots x_n,x]$	$\dfrac{\phi_1(x)\Longrightarrow\phi_1(x)}{\forall x\phi_1\Longrightarrow\phi_1(x)}$
$\neg A\vee NP_A(x_1\dots x_n)$	$A\Longrightarrow A$
$\neg NP_{\phi_1}[x_1\dots x_n]\vee NP_{\phi_1\vee\phi_2}(x_1\dots x_n)$	$\dfrac{\phi_1\Longrightarrow\phi_1}{\phi_1\Longrightarrow\phi_1\vee\phi_2}$
$\neg NP_{\phi_2}[x_1\dots x_n]\vee NP_{\phi_1\vee\phi_2}(x_1\dots x_n)$	$\dfrac{\phi_2\Longrightarrow\phi_2}{\phi_2\Longrightarrow\phi_1\vee\phi_2}$
$\neg NP_{\phi_1}[x_1\dots x_n]\vee\neg NP_{\phi_2}[x_1\dots x_n]$ $\vee NP_{\phi_1\wedge\phi_2}(x_1\dots x_n)$	$\dfrac{\phi_1\Longrightarrow\phi_1\quad\phi_2\Longrightarrow\phi_2}{\phi_1,\phi_2\Longrightarrow\phi_1\wedge\phi_2}$
$NP_{\phi_1}(x_1\dots x_n)\vee NP_{\neg\phi_1}(x_1\dots x_n)$	$\dfrac{\phi_1\Longrightarrow\phi_1}{\Longrightarrow\phi_1,\neg\phi_1}$
$NP_{\phi_1}[x_1\dots x_n]\vee NP_{\phi_1\to\phi_2}(x_1\dots x_n)$	$\dfrac{\dfrac{\phi_1\Longrightarrow\phi_1}{\phi_1\Longrightarrow\phi_1,\phi_2}}{\Longrightarrow\phi_1,\phi_1\to\phi_2}$
$\neg NP_{\phi_2}[x_1\dots x_n]\vee NP_{\phi_1\to\phi_2}(x_1\dots x_n)$	$\dfrac{\dfrac{\phi_2\Longrightarrow\phi_2}{\phi_1,\phi_2\Longrightarrow\phi_2}}{\phi_2\Longrightarrow\phi_1\to\phi_2}$
$\neg NP_{\phi_1}[x_1\dots x_n,x]\vee NP_{\exists x\phi_1}(x_1\dots x_n)$	$\dfrac{\phi_1(x)\Longrightarrow\phi_1(x)}{\phi_1(x)\Longrightarrow\exists x\phi_1}$

LK' + cut:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{t^{\phi}_{C\to S}(C_1)}}{\dots \text{plusieurs échanges} \dots}}{\Gamma_1 \Longrightarrow \Delta_1, \sigma(t^{\phi}_{A\to F}(A))} \quad \dfrac{\dfrac{\dfrac{\mathcal{D}_2}{t^{\phi}_{C\to S}(C_2)}}{\dots \text{plusieurs échanges} \dots}}{\sigma(t^{\phi}_{A\to F}(A)), \Gamma_2 \Longrightarrow \Delta_2}}{\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}}{\dots \text{plusieurs échanges} \dots}}{t^{\phi}_{C\to S}(C)}$$

En utilisant le théorème d'élimination des coupures de [Gen69], on peut transformer cette dérivation en une dérivation dans LK' de $t^{\phi}_{C\to S}(C)$. C.Q.F.D.

Lemme 14 *Soit $C = (C' \vee L)$ une clause obtenue par contraction à partir de la clause $C_1 = (C' \vee L \vee L)$ telle que L est un littéral dont l'atome est $\sigma(A)$, avec $A \in A_\phi$ et $t^{\phi}_{C\to S}(C_1)$ soit bien défini et dérivable dans LK'. Alors $t^{\phi}_{C\to S}(C)$ est bien défini et $t^{\phi}_{C\to S}(C)$ est dérivable dans LK'.*

Preuve: En utilisant un argument similaire à celui utilisé dans le lemme précédent, on montre que $t^{\phi}_{C\to S}(C)$ est bien défini. Montrons maintenant que, sous les hypothèses du lemme, ce séquent est dérivable dans LK'. Soit $\mathcal{D}$ la dérivation dans LK' du séquent $t^{\phi}_{C\to S}(C_1)$ et supposons que $L = \sigma(A), A \in A_\phi$, le cas $L = \neg\sigma(A), A \in A_\phi$ se déduisant immédiatement par symétrie de celui-ci. On dérive alors le séquent $t^{\phi}_{C\to S}(C)$ comme suit:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}}{t^{\phi}_{C\to S}(C_1)}}{\dots \text{plusieurs échanges} \dots}}{\Gamma_1 \Longrightarrow \Delta_1, \sigma(t^{\phi}_{A\to F}(A)), \sigma(t^{\phi}_{A\to F}(A))}}{\Gamma_1 \Longrightarrow \Delta_1, \sigma(t^{\phi}_{A\to F}(A))}}{\dots \text{plusieurs échanges} \dots}}{t^{\phi}_{C\to S}(C)}$$

C.Q.F.D.

Lemme 15 *Soit $C = \sigma(C')$ une clause obtenue par instanciation à partir de la clause C' telle que C' soit dérivable dans LK' et $t^{\phi}_{C\to S}(C')$ soit bien défini. Alors $t^{\phi}_{C\to S}(C)$ est bien défini et $t^{\phi}_{C\to S}(C)$ est dérivable dans LK'.*

Preuve: On vérifie simplement que l'on peut appliquer le lemme 5 d'instanciation d'une dérivation $\mathcal{D}$ d'un séquent $\Gamma \Longrightarrow \Delta$ en une dérivation $\mathcal{D}(y/s)$ d'un séquent $\Gamma(y/s) \Longrightarrow \Delta(y/s)$.

On rappelle que l'on a supposer pour toute substitution σ apparaissant dans une application de la règle d'instanciation que:

$$\forall y \in Dom(\sigma), Var(\sigma(y)) \cap V_\phi = \emptyset$$

où V_ϕ désigne l'ensemble des variables quantifiées dans ϕ et

$$\forall y \in Dom(\sigma), Var(\sigma(y)) \cap Dom(\sigma) = \emptyset.$$

Donc pour tout $y \in Dom(\sigma)$, $\sigma(y)$ est un terme dont les variables libres sont distinctes des variables quantifiées dans les formules des séquents de la dérivation $\mathcal{D}$ de $t^{\phi}_{C \to S}(C')$. De plus, comme ϕ ne contient pas de sous-formule $\forall x \phi_1$ de polarité positive ni de sous-formule $\exists x \phi_1$ de polarité négative, $t^{\phi}_{C \to S}(C')$ ne peut contenir de variable propre. On peut appliquer le lemme 5.

Posons $Dom(\sigma) = \{y_1 \ldots y_n\}$. La dérivation dans LK' de $t^{\phi}_{C \to S}(C)$ s'obtient en considérant la dérivation

$$(\ldots(\mathcal{D}(y_1/\sigma(y_1)))\ldots)(y_n/\sigma(y_n)).$$

C.Q.F.D.

La preuve ainsi obtenue n'est pas satisfaisante et ceci pour deux raisons:

- la preuve comporte essentiellement des applications des règles de coupure et de contraction: elle est aussi peu naturelle et aussi peu lisible qu'une preuve par Résolution. De plus, dans bien des cas, l'utilisation de la règle de coupure semble introduire une complication inutile: elle peut être souvent éliminée et remplacée par une règle d'introduction d'un connectif ou d'un quantificateur.

- la traduction ne permet pas de prendre en compte des dérivations dans LK de théorèmes comportant des occurrences positives (resp. négatives) de sous-formules de la forme $\forall x \phi_1$ (resp. $\exists x \phi_1$). En effet, le renommage de telles sous-formules introduit les clauses auxquelles il n'est pas possible d'associer une dérivation dans LK comme nous l'avons fait pour les autres clauses de $Rn(\phi)$. En particulier, les dérivations

$$\frac{\phi_1(x/v_{f_x(x_1 \ldots x_n)}) \Longrightarrow \phi_1(x/v_{f_x(x_1 \ldots x_n)})}{\exists x \phi_1 \Longrightarrow \phi_1(x/v_{f_x(x_1 \ldots x_n)})}$$

$$\frac{\phi_1(x/v_{f_x(x_1 \ldots x_n)}) \Longrightarrow \phi_1(x/v_{f_x(x_1 \ldots x_n)})}{\phi_1(x/v_{f_x(x_1 \ldots x_n)}) \Longrightarrow \forall x \phi_1}$$

sont incorrectes car elles ne respectent pas les conditions imposées sur les variables propres dans l'applications des règles $\forall - IC$, $\exists - IA$.

5.3 Utilisation de règles d'introduction de connectif et de quantificateur

Nous montrons qu'il est possible d'améliorer considérablement la lisibilité des preuves dans $LK + cut$ obtenues à partir de la traduction décrite au paragraphe précédent en remplaçant certaines applications de la règle de coupure par une application d'une règle appropriée d'introduction d'un connectif ou d'un quantificateur.

Lemme 16 *Etant donnée une formule valide ϕ, soit F l'ensemble des symboles fonctionnels apparaissant dans $CRn(\neg\phi)$. Soit P^+ l'ensemble des symboles de prédicats comprenant*

- *les symboles de prédicat des sous-formules atomiques de ϕ*
- *les symboles de prédicat des littéraux de Skolem correspondant au renommage d'une occurrence* positive *d'une sous-formule de $\neg\phi$.*

Soit I_ϕ l'ensemble des atomes $A(P^+, F)$. Alors I_ϕ est une partition de $CRn(\neg\phi)$.

La preuve de ce lemme est triviale en considérant les clauses de $CRn(\neg\phi)$ qui sont décrites ci-après (voir figure 5.2). On remarque que les seules clauses négatives de $CRn(\neg\phi)$ sont celles de la forme $\neg A \vee NP_A(x_1 \ldots x_n)$ issues du renommage d'une sous-formule atomique A de polarité négative dans $\neg\phi$.

Nous allons montrer que toute application de la règle de coupure sur le littéral positif d'une clause C de $CRn(\phi, -1)$ peut être traduite dans LK par une application de la règle d'introduction du connectif (resp. du quantificateur) principal de la formule dont le renommage a produit la clause C.

Pour établir ce résultat, nous supposerons que l'on dispose d'une déduction par Résolution de l'une des deux formes ci-dessous

$$\frac{\dfrac{\mathcal{D}_1}{C_{\Gamma\Longrightarrow\Delta}} \quad \dfrac{C}{\sigma(C)}}{C_{\Gamma''\Longrightarrow\Delta''}} \qquad \frac{\dfrac{\mathcal{D}_1}{C_{\Gamma\Longrightarrow\Delta}} \quad \dfrac{\mathcal{D}_2}{C_{\Gamma'\Longrightarrow\Delta'}} \quad \dfrac{C}{\sigma(C)}}{C_{\Gamma''\Longrightarrow\Delta''}}$$

où $C_{\Gamma\Longrightarrow\Delta}$,$C_{\Gamma'\Longrightarrow\Delta'}$, $C_{\Gamma''\Longrightarrow\Delta''}$ sont des clauses et C est une clause I_ϕ-positive de $CRn(\phi, -1)$. On suppose alors, par hypothèse d'induction, que l'on dispose d'une dérivation dans LK' des séquents $\Gamma \Longrightarrow \Delta$, $\Gamma' \Longrightarrow \Delta'$ et l'on montre comment sans coupure obtenir une dérivation dans LK' de $\Gamma'' \Longrightarrow \Delta''$. Pour toute substitution σ telle que toute variable de $Dom(\sigma)$ est distincte des variables quantifiées dans ϕ et pour toutes sous-formules ϕ_1, ϕ_2 de ϕ, on notera ψ_1 (resp. ψ_2) la formule $\sigma(\phi_1)$ (resp $\sigma(\phi_2)$).

Figure 5.2: Partition de $CRn(\neg\phi)$ selon I_ϕ.

Clause C de $CRn(\neg\phi)$	Littéraux évalués à vrai dans I_ϕ
Renommage d'une sous-formule positive	
$NP_{\neg\phi}$	$NP_{\neg\phi}$
$\neg NP_A(x_1 \dots x_n) \vee A$	A
$\neg NP_{\phi_1 \wedge \phi_2}(x_1 \dots x_n) \vee NP_{\phi_1}[x_1 \dots x_n]$	$NP_{\phi_1}[x_1 \dots x_n]$
$\neg NP_{\phi_1 \wedge \phi_2}(x_1 \dots x_n) \vee NP_{\phi_2}[x_1 \dots x_n]$	$NP_{\phi_2}[x_1 \dots x_n]$
$\neg NP_{\phi_1 \vee \phi_2}(x_1 \dots x_n) \vee$ $NP_{\phi_1}[x_1 \dots x_n] \vee NP_{\phi_2}[x_1 \dots x_n]$	$NP_{\phi_i}[x_1 \dots x_n] (i = 1,2)$
$\neg NP_{\neg\phi_1}(x_1 \dots x_n) \vee \neg NP_{\phi_1}(x_1 \dots x_n)$	$\neg NP_{\phi_1}(x_1 \dots x_n)$
$\neg NP_{\phi_1 \rightarrow \phi_2}(x_1 \dots x_n) \vee$ $\neg NP_{\phi_1}[x_1 \dots x_n] \vee NP_{\phi_2}[x_1 \dots x_n]$	$\neg NP_{\phi_1}[x_1 \dots x_n], NP_{\phi_2}[x_1 \dots x_n]$
$\neg NP_{\forall x \phi_1}(x_1 \dots x_n) \vee NP_{\phi_1}[x_1 \dots x_n, x]$	$NP_{\phi_1}[x_1 \dots x_n, x]$
$\neg NP_{\exists x \phi_1}(x_1 \dots x_n) \vee NP_{\phi_1}[x_1 \dots x_n, f_x(x_1 \dots x_n)]$	$NP_{\phi_1}[x_1 \dots x_n, f_x(x_1 \dots x_n)]$
Renommage d'une sous-formule négative	
$\neg A \vee NP_A(x_1 \dots x_n)$	-
$\neg NP_{\phi_1}[x_1 \dots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \dots x_n)$	$\neg NP_{\phi_1}[x_1 \dots x_n]$
$\neg NP_{\phi_2}[x_1 \dots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \dots x_n)$	$\neg NP_{\phi_2}[x_1 \dots x_n]$
$\neg NP_{\phi_1}[x_1 \dots x_n] \vee \neg NP_{\phi_2}[x_1 \dots x_n]$ $\vee NP_{\phi_1 \wedge \phi_2}(x_1 \dots x_n)$	$\neg NP_{\phi_i}[x_1 \dots x_n] (i = 1,2)$
$NP_{\phi_1}(x_1 \dots x_n) \vee NP_{\neg\phi_1}(x_1 \dots x_n)$	$NP_{\phi_1}(x_1 \dots x_n)$
$NP_{\phi_1}[x_1 \dots x_n] \vee NP_{\phi_1 \rightarrow \phi_2}(x_1 \dots x_n)$	$NP_{\phi_1}[x_1 \dots x_n]$
$\neg NP_{\phi_2}[x_1 \dots x_n] \vee NP_{\phi_1 \rightarrow \phi_2}(x_1 \dots x_n)$	$\neg NP_{\phi_2}[x_1 \dots x_n]$
$\neg NP_{\phi_1}[x_1 \dots x_n, f_x(x_1 \dots x_n)] \vee NP_{\forall x \phi_1}(x_1 \dots x_n)$	$\neg NP_{\phi_1}[x_1 \dots x_n, f_x(x_1 \dots x_n)]$
$\neg NP_{\phi_1}[x_1 \dots x_n, x] \vee NP_{\exists x \phi_1}(x_1 \dots x_n)$	$\neg NP_{\phi_1}[x_1 \dots x_n, x]$

• Coupure avec une formule atomique

La dérivation par Résolution

$$\dfrac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1}, \sigma_1(A_1)} \quad \dfrac{\neg A_2 \vee NP_{A_2}(x_1 \ldots x_n)}{\sigma_2(\neg A_2 \vee NP_{A_2}(x_1 \ldots x_n))}}{C_{\Gamma_1 \Longrightarrow \Delta_1}, \sigma_2(NP_{A_2}(x_1 \ldots x_n))}$$

où A_1 (resp. A_2) désigne une sous-formule atomique de polarité positive (resp. négative) de ϕ et $\sigma_1(A_1) = \sigma_2(A_2) = A$ se traduit dans $LK' + Cut$ par:

$$\dfrac{\mathcal{D}_1}{\Gamma_1 \Longrightarrow \Delta_1, A}$$

• Introduction de $\neg$

$$\dfrac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \neg\sigma(NP_{\phi_1}(x_1 \ldots x_n))} \quad \dfrac{NP_{\phi_1}(x_1 \ldots x_n) \vee NP_{\neg\phi_1}(x_1 \ldots x_n)}{\sigma(NP_{\phi_1}(x_1 \ldots x_n) \vee NP_{\neg\phi_1}(x_1 \ldots x_n))}}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\neg\phi_1}(x_1 \ldots x_n))}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \Longrightarrow \Delta}}{\ldots \text{plusieurs échanges} \ldots}}{\psi_1, \Gamma_1 \Longrightarrow \Delta_1}}{\Gamma_1 \Longrightarrow \Delta_1, \neg\psi_1}$$

Le cas d'une dérivation où la clause positive est

$$\neg NP_{\neg\phi_1}(x_1 \ldots x_n) \vee \neg NP_{\phi_1}(x_1 \ldots x_n)$$

est similaire au cas précédent.

• Introduction de $\vee$ et $\wedge$

$$\dfrac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1}[x_1 \ldots x_n])} \quad \dfrac{\neg NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n)}{\sigma(\neg NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n))}}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n))}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \Longrightarrow \Delta}}{\ldots \text{plusieurs échanges} \ldots}}{\Gamma_1 \Longrightarrow \Delta_1, \psi_1}}{\Gamma_1 \Longrightarrow \Delta_1, \psi_1 \vee \psi_2}$$

Le cas d'une dérivation où la clause positive est

$$\neg NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n)$$

$$\neg NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n) \vee NP_{\phi_1}[x_1 \ldots x_n]$$

$$\neg NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n) \vee NP_{\phi_2}[x_1 \ldots x_n]$$

est similaire au cas précédent.

$$\dfrac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1}[..])} \quad \dfrac{\mathcal{D}_2}{C_{\Gamma_2 \Longrightarrow \Delta_2} \vee \sigma(NP_{\phi_2}[..])} \quad \dfrac{NP_{\phi_1}[..] \vee \neg NP_{\phi_2}[..] \vee NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n)}{\sigma(NP_{\phi_1}[..] \vee \neg NP_{\phi_2}[..] \vee NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n))}}{C_{\Gamma_2, \Gamma_1 \Longrightarrow \Delta_1, \Delta_2} \vee \sigma(NP_{\phi_1 \wedge \phi_2}(x_1 \ldots x_n))}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \Longrightarrow \Delta}}{\ldots \text{plusieurs échanges} \ldots}}{\Gamma_1 \Longrightarrow \Delta_1, \psi_1} \quad \dfrac{\dfrac{\dfrac{\mathcal{D}_2}{\Gamma' \Longrightarrow \Delta'}}{\ldots \text{plusieurs échanges} \ldots}}{\Gamma_2 \Longrightarrow \Delta_2, \psi_2}$$
$$\Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2, \psi_1 \wedge \psi_2$$

Le cas d'une dérivation où la clause positive est

$$\neg NP_{\phi_1 \vee \phi_2}(x_1 \ldots x_n) \vee NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_2}[x_1 \ldots x_n]$$

est similaire au cas précédent.

- Introduction de $\rightarrow$

$$\dfrac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \neg\sigma(NP_{\phi_1}[x_1 \ldots x_n])} \quad \dfrac{NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n)}{\sigma(NP_{\phi_1}[x_1 \ldots x_n] \vee NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n))}}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n))}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \Longrightarrow \Delta}}{\ldots \text{plusieurs échanges} \ldots}}{\psi_1, \Gamma_1 \Longrightarrow \Delta_1}}{\psi_1, \Gamma_1 \Longrightarrow \Delta_1, \psi_2}$$
$$\Gamma_1 \Longrightarrow \Delta_1, \psi_1 \rightarrow \psi_2$$

$$\dfrac{\dfrac{\mathcal{D}_2}{C_{\Gamma_2 \Longrightarrow \Delta_2} \vee \sigma(NP_{\phi_2}[x_1 \ldots x_n])} \quad \dfrac{\neg NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n)}{\sigma(\neg NP_{\phi_2}[x_1 \ldots x_n] \vee NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n))}}{C_{\Gamma_2 \Longrightarrow \Delta_2} \vee \sigma(NP_{\phi_1 \rightarrow \phi_2}(x_1 \ldots x_n))}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_2}{\Gamma' \Longrightarrow \Delta'}}{\ldots \text{plusieurs échanges} \ldots}}{\Gamma_2 \Longrightarrow \Delta_2, \psi_2}}{\psi_1, \Gamma_2 \Longrightarrow \Delta_2, \psi_2}$$
$$\Gamma_2 \Longrightarrow \Delta_2, \psi_1 \rightarrow \psi_2$$

$$\dfrac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1}[..])} \quad \dfrac{\mathcal{D}_2}{C_{\Gamma_2 \Longrightarrow \Delta_2} \vee \neg\sigma(NP_{\phi_2}[..])} \quad \dfrac{\neg NP_{\phi_1 \to \phi_2}(x_1 \ldots x_n) \vee \neg NP_{\phi_1}[..] \vee NP_{\phi_2}[..]}{\sigma(\neg NP_{\phi_1 \to \phi_2}(x_1 \ldots x_n) \vee \neg NP_{\phi_1}[..] \vee NP_{\phi_2}[..])}}{C_{\Gamma_2,\Gamma_1 \Longrightarrow \Delta_1,\Delta_2} \vee \neg\sigma(NP_{\phi_1 \to \phi_2}(x_1 \ldots x_n))}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_2}{\Gamma' \Longrightarrow \Delta'}}{\ldots \text{plusieurs échanges} \ldots}}{\dfrac{\psi_2, \Gamma_2 \Longrightarrow \Delta_2}{\psi_1 \to \psi_2, \Gamma_1, \Gamma_2 \Longrightarrow \Delta_1, \Delta_2}} \quad \dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \Longrightarrow \Delta}}{\ldots \text{plusieurs échanges} \ldots}}{\Gamma_1 \Longrightarrow \Delta_1, \psi_1}$$

- Introduction de $\exists$ et $\forall$

$$\dfrac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1}[x_1 \ldots x_n, x])} \quad \dfrac{\neg NP_{\phi_1}[x_1 \ldots x_n, x] \vee NP_{\exists x \phi_1}(x_1 \ldots x_n)}{\sigma(\neg NP_{\phi_1}[x_1 \ldots x_n, x] \vee NP_{\exists x \phi_1}(x_1 \ldots x_n))}}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \neg\sigma(NP_{\exists x \phi_1}(x_1 \ldots x_n))}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \Longrightarrow \Delta}}{\ldots \text{plusieurs échanges} \ldots}}{\dfrac{\Gamma_1 \Longrightarrow \Delta_1, \psi_1(x/\sigma(x))}{\Gamma_1 \Longrightarrow \Delta_1, \exists x \psi_1}}$$

Le cas d'une dérivation où la clause positive est

$$\neg NP_{\forall x \phi_1}(x_1 \ldots x_n) \vee NP_{\phi_1}[x_1 \ldots x_n, x]$$

est similaire au cas précédent.

Afin de prendre en compte les clauses contenant des termes de Skolem, il est naturel de vouloir envisager dans un déduction dé-Skolemisée de NP_ϕ la traduction suivante:

$$\dfrac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1}[x_1 \ldots x_n, x])} \quad \dfrac{\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}[..]}{\sigma(\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}[..])}}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \neg\sigma(NP_{\forall x \phi_1}(x_1 \ldots x_n))}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\Gamma \Longrightarrow \Delta}}{\ldots \text{plusieurs échanges} \ldots}}{\dfrac{\Gamma_1 \Longrightarrow \Delta_1, \psi_1(x/v_{\sigma(f_x(x_1 \ldots x_n))})}{\Gamma_1 \Longrightarrow \Delta_1, \forall x \psi_1}}$$

et de procéder de manière similaire si la clause positive de la coupure est:

$$\neg NP_{\exists x \phi_1}(x_1 \ldots x_n) \vee NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)]$$

En règle générale, on ne peut assurer que la variable $v_{\sigma(f_x(x_1 \ldots x_n))}$ n'apparait pas libre dans $\Gamma_1 \Longrightarrow \Delta_1$. Cependant, nous allons montrer que pour certaines classes de dérivations cette condition est vérifiée.

Etant donné une formule $\phi_1 \sqsubseteq \phi$, appelons *occurrence existentielle* de ϕ_1 dans ϕ toute occurrence $O.1$ de ϕ_1 dans ϕ telle que

- $\phi \mid_O = \exists x \phi_1$ et O est désigne une occurrence d'une sous-formule de polarité positive dans ϕ;
- $\phi \mid_O = \forall x \phi_1$ et O est désigne une occurrence d'une sous-formule de polarité négative dans ϕ;

Considérons désormais une formule fermée valide ϕ. Soit F l'ensemble des symboles fonctionnels apparaissant dans $CRn(\neg\phi)$ et soit P l'ensemble des symboles de prédicats apparaissant dans $CRn(\neg\phi)$. Considérons un A-ordre $\succ$ défini sur $A(P,F)$ et vérifiant les propriétés suivantes pour tout $A_1, A_2 \in A(P,F)$:

1. $A_1 \succ A_2$ si $A_1 = NP_{\phi_1}(t_1 \ldots t_m)$ est une instance d'une formule atomique de ϕ ou d'un littéral de Skolem de ϕ qui correspond au renommage d'une sous-formule non existentielle de ϕ et $A_2 = NP_{\phi_2}(s_1 \ldots s_n)$ est une instance d'un littéral de Skolem de ϕ qui correspond au renommage d'une sous-formule existentielle de ϕ tels qu'il existe un terme $t_{i_0} (1 \leq i_0 \leq m)$ tel que t_{i_0} contient tous les termes s_j $j = 1, \ldots, n$ comme sous-termes.

2. $A_1 \succ A_2$ si $A_1 = NP_{\phi_1}(t_1 \ldots t_m)$, $A_2 = NP_{\phi_2}(s_1 \ldots s_n)$ sont deux instances de littéraux de Skolem de ϕ qui correspondent chacun au renommage d'une sous-formule existentielle de ϕ et il existe un terme $t_{i_0} (1 \leq i_0 \leq m)$ tel que t_{i_0} contient *strictement* tous les termes s_j $j = 1, \ldots, n$ comme sous-termes.

On peut toujours construire un tel A-ordre en combinant un ordre sur les symboles de prédicats avec un ordre de simplification $\succ_{T(F,V)}$ ([Der87]) défini sur $T(F,V)$ qui vérifie la propriété du sous-terme:

$$f(\ldots t \ldots) \succ_{T(F,V)} t$$

Considérons une réfutation $\mathcal{R}$ par Résolution de $CRn(\neg\phi)$. Par application du théorème 5 de normalisation, on peut transformer cette réfutation en une CC-réfutation par Résolution Sémantique Ordonnée utilisant la partition I_ϕ et un A-ordre $\succ$ vérifiant les propriétés énoncés ci-dessus.

En utilisant l'argument brièvement décrit au début de ce chapitre, on peut en effaçant les clauses $NP_{\neg\phi}, \neg NP_{\neg\phi} \vee \neg NP_{\phi}$ obtenir une déduction de NP_{ϕ} par Résolution Sémantique Ordonnée utilisant I_{ϕ} et $\succ$.

Supposons qu'il y ait dans cette dérivation des applications de la règle de coupure avec comme clause I_{ϕ}-positive une instance de la clause

$$\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}(x_1 \ldots x_n)$$

ou bien une instance de la clause

$$\neg NP_{\exists x \phi_1}(x_1 \ldots x_n) \vee NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)]$$

Cette inférence est de la forme:

$$\frac{\dfrac{\mathcal{D}_1}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1}[x_1 \ldots x_n, x])} \quad \dfrac{\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}[..]}{\sigma(\neg NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)] \vee NP_{\forall x \phi_1}[..])}}{C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \neg\sigma(NP_{\forall x \phi_1}(x_1 \ldots x_n))}$$

où $C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \sigma(NP_{\phi_1}[x_1 \ldots x_n, x]), C_{\Gamma_1 \Longrightarrow \Delta_1} \vee \neg\sigma(NP_{\forall x \phi_1}(x_1 \ldots x_n))$désignent des clauses négatives et $\sigma(NP_{\phi_1}[x_1 \ldots x_n, x]) = \sigma(NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)])$ est un atome maximal de cette clause. D'après les conditions imposées sur le A-ordre considéré, on peut établir les faits suivants:

- $\sigma(f_x(x_1 \ldots x_n))$ n'apparaît dans aucun littéral de $C_{\Gamma_1 \Longrightarrow \Delta_1}$ qui est soit une instance d'une formule atomique de ϕ, soit un littéral de Skolem issu du renommage d'une sous-formule non existentielle de ϕ. (cf. condition 1 imposée à $\succ$).

- les autres littéraux apparaissant dans $C_{\Gamma_1 \Longrightarrow \Delta_1}$ sont nécessairement des littéraux de Skolem associés au renommage d'une sous-formule existentielle dans ϕ. De plus, ces littéraux sont deux à deux distincts (cf. hypothèses du théorème de normalisation). Or $\sigma(f_x(x_1 \ldots x_n))$ n'apparaît dans aucun de ces littéraux. En effet, supposons le contraire: il existe alors un littéral L_2 dont l'atome est de la forme $\sigma_2(NP_{\phi_2}(y_1 \ldots y_m, f_y(y_1 \ldots y_m)))$ et qui contient $\sigma(f_x(x_1 \ldots x_n))$ et donc tous les termes de $\sigma(NP_{\phi_1}[x_1 \ldots x_n, f_x(x_1 \ldots x_n)])$. Comme les littéraux de la clause négative sont deux à deux distincts, $f_x \neq f_y$ et donc $\sigma_2(f_y(y_1 \ldots y_m))$ contient tous les arguments de $\sigma(f_x(x_1 \ldots x_n))$. Ce qui contredit la condition 2 imposée à $\succ$.

5.4 Conclusion

Dans le cas où $CRn(\neg\phi)$ ne contient pas de fonction de Skolem, la traduction des preuves par Résolution dans le calcul $LK + Cut$ préserve la structure des preuves par Résolution.

Considérons par exemple la formule valide $\phi = ((A \vee \neg A) \wedge (A \vee \neg A))$. Afin de pouvoir distinguer les diverses occurrences de la formule atomique A dans ϕ, notons A^i la i-ème occurrence de A dans ϕ et écrivons la formule ϕ sous la forme: $((A^1 \vee \neg A^2) \wedge (A^3 \vee \neg A^4))$. Pour des soucis de lisibilité, considérons la forme clausale obtenue en procédant au renommage exhaustif de $\neg\phi$ sans toute fois renommer les atomes de ϕ. On obtient alors la forme clausale suivante:

1. $\neg NP$
2. $(\neg NP_1 \vee \neg NP_2 \vee NP)$
3. $(\neg A^1 \vee NP_1)$
4. $(NP_3 \vee NP_1)$
5. $(\neg NP_3 \vee A^2)$
6. $(\neg A^3 \vee NP_2)$
7. $(NP_4 \vee NP_2)$
8. $(\neg NP_4 \vee A^4)$

où NP (resp. NP_1, NP_2) désigne le littéral de Skolem correspondant au renommage de ϕ (resp. du premier et du second conjoint de ϕ) et NP_3 (resp. NP_4) désigne le littéral de Skolem correspondant au renommage de $\neg A^2$ (resp. $\neg A^4$).

Soit $\mathcal{D}_1$ la dérivation

$$\dfrac{\dfrac{\dfrac{\neg A^1 \vee NP_1 \quad \neg NP_3 \vee A^2}{\neg NP_3 \vee NP_1} \quad NP_3 \vee NP_1}{NP_1 \vee NP_1}}{NP_1}$$

et soit $\mathcal{D}_2$ la dérivation

$$\dfrac{\dfrac{\dfrac{\neg A^3 \vee NP_2 \quad \neg NP_4 \vee A^4}{\neg NP_4 \vee NP_2} \quad NP_4 \vee NP_2}{NP_2 \vee NP_2}}{NP_2}$$

La dérivation suivante de NP

$$\dfrac{\dfrac{\mathcal{D}_1}{NP_1}\ \dfrac{\mathcal{D}_2}{NP_2}\ \neg NP_1 \vee \neg NP_2 \vee NP}{NP}$$

se traduit dans le calcul LK par

$$\dfrac{\dfrac{\dfrac{\dfrac{A \Longrightarrow A}{\Longrightarrow A, \neg A}}{\Longrightarrow A, A \vee \neg A}}{\Longrightarrow A \vee \neg A, A}}{\dfrac{\Longrightarrow A \vee \neg A, A \vee \neg A}{\Longrightarrow A \vee \neg A}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{A \Longrightarrow A}{\Longrightarrow A, \neg A}}{\Longrightarrow A, A \vee \neg A}}{\Longrightarrow A \vee \neg A, A}}{\dfrac{\Longrightarrow A \vee \neg A, A \vee \neg A}{\Longrightarrow A \vee \neg A}}$$

$$\Longrightarrow ((A \vee \neg A) \wedge (A \vee \neg A))$$

tandis que la dérivation

$$\dfrac{\dfrac{\neg A^1 \vee NP_1 \quad \neg A^3 \vee NP_2 \quad \neg NP_1 \vee \neg NP_2 \vee NP}{\neg A^1 \vee \neg A^3 \vee NP}}{\neg A \vee NP} \qquad \dfrac{\dfrac{A^2 \vee NP_1 \quad A^4 \vee NP_2 \quad \neg NP_1 \vee \neg NP_2 \vee NP}{A^2 \vee A^4 \vee NP}}{A \vee NP}$$

$$\dfrac{NP \vee NP}{NP}$$

dans laquelles les clauses $\neg A^3 \vee NP_2$ et $A^2 \vee NP_1$ sont obtenues par les déductions

$$\dfrac{\neg A^3 \vee \neg NP_4 \quad NP_4 \vee NP_2}{\neg A^3 \vee NP_2} \qquad \dfrac{A^2 \vee \neg NP_3 \quad NP_3 \vee NP_1}{A^2 \vee NP_1}$$

se traduit dans $LK + cut$ par:

$$\dfrac{\dfrac{\dfrac{A \Longrightarrow A}{A \Longrightarrow A \vee \neg A} \quad \dfrac{A \Longrightarrow A}{\Longrightarrow A, A \vee \neg A}}{A, A \Longrightarrow ((A \vee \neg A) \wedge (A \vee \neg A))}}{A \Longrightarrow ((A \vee \neg A) \wedge (A \vee \neg A))} \qquad \dfrac{\dfrac{\dfrac{A \Longrightarrow A}{\Longrightarrow A, \neg A} \quad \dfrac{A \Longrightarrow A}{\Longrightarrow A, \neg A}}{\Longrightarrow A, A \vee \neg A \quad \Longrightarrow A, A \vee \neg A}}{\dfrac{\Longrightarrow A, A, ((A \vee \neg A) \wedge (A \vee \neg A))}{\Longrightarrow A, ((A \vee \neg A) \wedge (A \vee \neg A))}}$$

$$\dfrac{\Longrightarrow ((A \vee \neg A) \wedge (A \vee \neg A)), ((A \vee \neg A) \wedge (A \vee \neg A))}{\Longrightarrow ((A \vee \neg A) \wedge (A \vee \neg A))}$$

Cette dernière dérivation par Résolution dans laquelle on résoud en premier sur les littéraux de Skolem, correspond en fait à la dérivation que l'on obtient en utilisant la mise sous forme clausale standard.

Dans le cas où $CRn(\phi)$ ne contient pas de fonction de Skolem, nous avons caractérisé les déductions de $CRn(\phi)$ qui correspondent à des preuves sans coupure dans le calcul LK' comme étant des déductions par Résolution Sémantique utilisant I_ϕ. Il convient de noter que l'utilisation d'autres stratégies permet d'obtenir des preuves correspondant elles aussi à des preuves sans coupure dans le calcul LK'. On peut mentionner par exemple la stratégie d'ordonnancement décrite dans [PG86] qui consiste à

- résoudre d'abord sur les littéraux de ϕ avant de résoudre sur les littéraux de Skolem issus du renommage de ϕ;
- si ϕ_1 est une sous-formule de ϕ_2 dans ϕ, résoudre d'abord sur les littéraux de symboles de Skolem NP_{ϕ_1} avant de résoudre sur les littéraux de Skolem de symboles de prédicats NP_{ϕ_2}.

Dans [PG86], l'emploi de cette stratégie est justifié par le fait que l'utilisation de la stratégie d'ordonnancement inverse produit les clauses que l'on aurait obtenues en utilisant la mise sous forme clausale standard. Ce qui n'est pas souhaitable en général puisqu'on cherche à éviter les redondances de cette transformation, celles-ci rendant inopérant le principe de Résolution pour certaines classes de problèmes [Ble77] [And81].

Dans le cas général, nous avons donc montré comment extraire de toute réfutation par Résolution de $CRn(\neg\phi)$ une preuve sans coupure dans le calcul des séquents de la validité de ϕ. Cette construction utilise l'existence de théorèmes de normalisation pour la Résolution Sémantique Ordonnée mais ne nécessite pas d'utiliser des structures de données auxilliaires comme dans [Pfe84]. Une autre approche eut été de produire une dérivation dans le calcul LK' de la forme *skolemisée* de ϕ, puis de transformer cette preuve en une preuve de ϕ. Une transformation similaire est décrite dans [Gal86] pp 344-349 dans la preuve du théorème d'Herbrand pour les formules prénexes. Cette transformation consiste à remplacer dans la déduction dans LK' de la forme skolemisée de ϕ chaque terme de Skolem par une variable appropriée puis à permuter les inférences de la pseudo-déduction de LK' ainsi obtenue jusqu'à ce que les conditions à imposer sur les variables propres soient vérifiées. Remarquons que les traitements réalisés dans le cadre de la Résolution par les procédures de normalisation sont similaires. Cependant, si dans le cadre de la Résolution la permutation des inférences se fait relativement simplement, il n'en est pas de même pour le calcul LK'. La nécessité de respecter les conditions imposées sur les variables propres des inférences $\exists - IA$, $\forall - IC$ ainsi que l'impossibilité de permuter deux inférences si la formule principale de l'une est une formule secondaire de l'autre rendent très complexe la description de la transformation de dé-skolemisation. Signalons enfin que l'utilisation des arbres d'expansion permettent d'apporter une solution simple et élégante à ce problème [Mil87]. Cependant, ceci se fait au prix d'une perte totale de la structure des preuves par Résolution.

Si l'on traduit une preuve par Résolution en preuve du calcul $LK + cut$ et si l'on considère la règle d'instanciation qui permet sous certaines conditions d'obtenir une

déduction de $\Gamma(x/t) \Longrightarrow \Delta(x/t)$ à partir de la déduction de $\Gamma \Longrightarrow \Delta$ comme une règle d'inférence dérivée, la preuve par Résolution et la preuve ainsi obtenue dans $LK + cut + instanciation$ sont du même ordre de grandeur.

Un tel résultat ne peut être obtenu pour la méthode de traduction qui repose sur l'existence d'un théorème de normalisation des preuves par Résolution en CC-dérivation par Résolution Sémantique Ordonnée. Ceci est du aux deux problèmes suivants:

- la transformation d'une preuve par Résolution en une CC-dérivation peut produire une preuve dont la taille mesurée en nombre d'inférences (et donc en nombre de symboles) est exponentiellement plus grande que la taille de la preuve initiale;
- il existe des formules insatisafisables sous forme clausale admettant des réfutations par Résolution de taille linéaire mais dont toute réfutation par Résolution Sémantique Ordonnée est de taille exponentielle.

Illustrons ces deux points à l'aide d'exemples. Considérons la formule insatisfaisable suivante:

$$\forall x_1 Q_1(x_1) \vee \forall x_2 Q_2(x_1, x_2) \vee \ldots \vee \forall x_n Q_n(x_1, \ldots, x_n)$$
$$\wedge \wedge_{i=1}^{n} (\neg Q_i(x_1, \ldots, x_{i-1}, a) \vee \neg Q_i(x_1, \ldots, x_{i-1}, b))$$

On peut obtenir en utilisant le renommage exhaustif un ensemble de clauses pour lequel il existe une réfutation par Résolution Sémantique Ordonnée simulant la Déduction Naturelle et comportant $O(n)$ inférences. Cependant la CC-dérivation associée comporte un nombre exponentiel d'applications de règles d'inférence. En fait, toute preuve dans LK de l'insatisfaisabilité de cette formule comporte elle aussi un nombre exponentiel de séquents.

Considérons la formule ϕ suivante [Ede90]:

$$[p(0) \wedge (\forall x p(x) \rightarrow p(f(x)))] \rightarrow p(f^{2^n}(0))$$

où $f^{2^n}(0)$ désigne le terme $\underbrace{f(\ldots(f}_{2^n}(0))\ldots)$ On peut dériver par Résolution la clause $\neg p(x) \vee p(f^{2^{i+1}}(x)))$ à partir de la clause $\neg p(x) \vee p(f^{2^i}(x)))$ comme suit:

$$\frac{\neg p(x) \vee p(f^{2^i}(x)) \quad \dfrac{\neg p(x) \vee p(f^{2^i}(x))}{\neg p(f^{2^i}(x)) \vee p(f^{2^{i+1}}(x))}}{\neg p(x) \vee p(f^{2^{i+1}}(x))}$$

Donc il existe une preuve linéaire de l'insatisfaisabilité de ϕ. Cependant, toute preuve par Résolution Sémantique comportera un nombre exponentiel d'inférences. De même, toute preuve dans le calcul des séquents LK comportera un nombre exponentiel de séquents.

Cependant, si l'on munit le calcul LK des règles d'instanciation et de coupure, on peut traduire ces preuves linéaires par Résolution en preuves linéaires dans le calcul des séquents.

L'intérêt de produire par Résolution des preuves qui puisent être facilement traduite en preuve par déduction naturelle réside dans le fait que l'on dispose d'une procédure de preuve efficace capable de fournir une justification lisible et compréhensible des preuves qu'elle produit. Très certainement la taille de la preuve est un facteur déterminant de lisibilité. Ceci impose d'utiliser les règles de coupure et d'instanciation lors de la traduction en Déduction Naturelle.

Chapitre 6

Amélioration de l'efficacité de la Résolution

Afin de montrer par Résolution l'insatisfaisabilité d'une formule ϕ, nous avons vu qu'il est possible de renommer tout ou partie des sous-formules de ϕ avant de procéder à la mise sous forme clausale. Jusqu'à présent, nous n'avons considéré que le renommage exhaustif des sous-formules de ϕ, et ce dans le but de mettre en évidence les rapports entre Résolution et Déduction Naturelle.

Dans ce chapitre, nous montrerons qu'il y a en général avantage à ne procéder qu'à un renommage partiel de la formule à réfuter afin de faciliter la recherche d'une preuve par Résolution. Nous présenterons plusieurs autres formes de renommage et nous tenterons de définir des critères pour l'obtention d'une "bonne" forme clausale. Nous discuterons les critères obtenus sur un exemple [BdlTC90a], [BdlTC90b].

Nous présenterons également un théorème permettant de restreindre l'espace de recherche de la Résolution et nous illustrerons sur quelques exemples l'intérêt de l'utilisation de simplifications non clausales.

6.1 Faiblesses du renommage exhaustif

Considérons la suite de formules insatisfaisables:

$$\begin{aligned} \phi_1 &= A_1 \wedge \neg A_1 \\ \phi_{n+1} &= \phi_n \vee (A_{n+1} \wedge \neg A_{n+1}) \end{aligned}$$

On montre facilement que:

- la mise sous forme clausale standard de ϕ_n produit 2^n clauses comportant chacune n littéraux distincts;
- l'ensemble de clauses ainsi obtenu est *minimalement* insatisfaisable.

Donc, toute réfutation par Résolution de cet ensemble de clauses comporte au moins 2^n applications des règles de coupure et de contraction.

Or le séquent $\phi_n \Longrightarrow$ est dérivable dans LK' en $O(n)$ inférences. En effet, on déduit $A_i \wedge \neg A_i \Longrightarrow$ comme suit:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A_i \Longrightarrow A_i}{\neg A_i, A_i \Longrightarrow}}{A_i \wedge \neg A_i, A_i \Longrightarrow}}{A_i, A_i \wedge \neg A_i \Longrightarrow}}{A_i \wedge \neg A_i, A_i \wedge \neg A_i \Longrightarrow}}{A_i \wedge \neg A_i \Longrightarrow}$$

Il reste alors à appliquer $n-1$ fois la règle $\vee - IA$ pour déduire ϕ_n. D'après les résultats des chapitres précédents, en appliquant le renommage exhaustif à la formule ϕ_n, on peut trouver par Résolution une réfutation comportant un nombre d'inférences *linéaire en fonction de n*. Donc pour certaines classes de formules, l'utilisation du renommage exhaustif permet d'obtenir *une réduction exponentielle* de la taille de la preuve la plus courte.

Cependant, il existe aussi des cas dans lesquels l'utilisation du renommage exhaustif ne présente pas d'intérêt. Par exemple, étant donnée une formule sous forme clausale, l'application du renommage exhaustif à cette formule conduira en général à la production d'un ensemble de clauses dont la réfutation la plus courte nécessitera plus d'inférences que la réfutation la plus courte que l'on obtiendrait en réfutant directement la formule initiale. Les résultats expérimentaux décrits dans [PG86] confortent cette analyse: pour les formules admettant une forme clausale très simple, il y a intérêt à ne pas procéder au renommage exhaustif. Par contre, pour des formules plus complexes, l'utilisation du renommage exhaustif permet des gains spectaculaires en efficacité.

Ceci suggère qu'il vaut mieux procéder uniquement à un renommage partiel des sous-formules d'une formule à réfuter par Résolution. Ce renommage devra éviter les renommages inutiles. Par renommages inutiles, nous entendons ceux qui peuvent être omis sans qu'il en résulte une augmentation de la taille de la forme clausale dans le pire des cas et qui permettent de réduire la taille de la réfutation la plus courte, au moins pour une classe de stratégies données.

6.2 Elimination des inférences triviales

Etant donnée une formule ϕ, soit R un sous-ensemble des sous-formules de ϕ. Notons $Rn(\phi, R)$ la formule obtenue en renommant l'ensemble R des sous-formules de ϕ et notons $CRn(\phi, R)$ la formule obtenue par mise sous forme clausale de $Rn(\phi, R)$. Enfin, notons $SCRn(\phi, R)$ le multi-ensemble des clauses de $CRn(\phi, R)$.

Nous avons mentionné précédemment que si ψ est une sous-formule de ϕ appartenant à R, le multi-ensemble de clauses $SCRn(\phi, R \setminus \{\psi\})$ peut être obtenu à partir de $SCRn(\phi, R)$ en utilisant la Résolution. En fait, le renommage R de la formule ϕ produit un ensemble de conjonctions $\wedge_{i=1}^{n} C_i$. L'un de ces conjoints correspond à la définition du littéral de Skolem NP_ψ et est de la forme

$$\forall x_1 \ldots x_n NP_\psi(x_1 \ldots x_n) \rightleftharpoons^p \psi'$$

La mise sous forme clausale de $Rn(\psi, R)$ va donc produire des clauses de la forme

- $\neg NP_\psi(x_1 \ldots x_n) \vee C^+_{\psi'}$ si $p \geq 0$;
- $NP_\psi(x_1 \ldots x_n) \vee C^-_{\psi'}$ si $p \leq 0$;

Notons respectivement par S_1^+ et par S_1^- le multi-ensemble de ces clauses et posons $S_1 = S_1^+ \cup S_1^-$.

D'autre part, il existe un conjoint $\phi'[NP_\psi(x_1 \ldots x_n)]$ de $Rn(\phi, R)$ contenant une occurrence de polarité opposée de $NP_\psi(x_1 \ldots x_n)$. Après mise sous forme clausale de ce conjoint, on obtient des clauses de la forme

- $\neg NP_\psi(t_1 \ldots t_n) \vee C^+_{\phi'}$ si $p \geq 0$;
- $NP_\psi(t_1 \ldots t_n) \vee C^-_{\phi'}$ si $p \leq 0$;

où t_i $1 \leq i \leq n$ est soit la variable x_i, soit un terme de Skolem de la forme $f_{x_i}(y_1 \ldots y_m)$ où $\{y_1 \ldots y_m\}$ est le sous-ensemble de l'ensemble des termes $\{t_1 \ldots t_m\}$

qui sont des variables. Notons respectivement par S_2^+ et par S_2^- le multi-ensemble de ces clauses et posons $S_2 = S_2^+ \cup S_2^-$.

Il est donc toujours possible de produire par Résolution des clauses de la forme

- $C^+_{\phi'} \vee C^+_{\psi'}(x_1/t_1 \ldots x_n/t_n)$ en résolvant une clause de S_1^+ avec une clause de S_2^+ sur l'atome $NP_\psi(t_1 \ldots t_n)$
- $C^-_{\phi'} \vee C^-_{\psi'}(x_1/t_1 \ldots x_n/t_n)$ en résolvant une clause de S_1^- avec une clause de S_2^- sur l'atome $NP_\psi(t_1 \ldots t_n)$.

Notons respectivement par R^+ et par R^- le multi-ensemble des résolvantes ainsi produites. Ces clauses correspondent en fait aux clauses que l'obtient si l'on considère la mise sous forme clausale de $\phi'[NP_\psi(x_1 \ldots x_n) \leftarrow \psi']$. Le multi-ensemble de clauses $SCRn(\phi, R \setminus \{\psi\})$ peut donc être obtenu à partir de $SCRn(\phi, R)$ en considérant l'ensemble de clauses

$$S = (SCRn(\phi, R) \setminus (S_1 \cup S_2)) \cup R^+ \cup R^-$$

Par la suite, nous utiliserons cette propriété pour montrer que le renommage d'une sous-formule ψ de ϕ est inutile en transformant une réfutation $\mathcal{R}$ de $SCRn(\phi, R)$ en une réfutation $\mathcal{R}'$ de S telle que $\mathcal{R}'$ comprend moins de pas d'inférence que $\mathcal{R}$.

Pour ce faire nous utiliserons les deux résultats (essentiellement techniques) suivants:

Soit S un ensemble de clauses et soit P un symbole de prédicat d'arité n. Soit $P(t_1 \ldots t_n)$ un atome tel que tout atome figurant dans S et ayant P pour symbole de prédicat soit une instance de $P(t_1 \ldots t_n)$. Etant donné un littéral L, définissons pour tout atome A le littéral A^* par :

- $A^* = \sigma(L)$ si $A = \sigma(P(t_1 \ldots t_n))$
- $A^* = A$ sinon.

De même, étant donné un littéral $L_1 = \neg A$, notons L_1^* le littéral complémentaire de A^*. Etant donnée une clause $C = (L_1 \vee \ldots \vee L_p)$, notons C^* la clause $L_1^* \vee \ldots \vee L_p^*$. Enfin, étant donnée une séquence de clauses $\mathcal{D} =< C_1, \ldots, C_q >$ notons $\mathcal{D}^*$ la séquence $< C_1^*, \ldots, C_q^* >$.

Lemme 17 : *Si $\mathcal{D}$ est une déduction d'une clause C à partir d'un ensemble S de clauses et si aucune variable de L ne figurant pas dans $P(t_1 \ldots t_n)$ n'apparaît*

pas dans $\mathcal{D}$, alors $\mathcal{D}^*$ est une déduction de C^* à partir de S^*, appelée la déduction obtenue à partir de $\mathcal{D}$ par remplacement uniforme *de l'atome $P(t_1 \ldots t_n)$ par le littéral L.*

Preuve: Par induction sur la taille de $\mathcal{D}$. C.Q.F.D.

Lemme 18 *Soit S un ensemble insatisfaisable de clauses et soit $\mathcal{R}$ une réfutation par Résolution de S. S'il existe dans S une clause C contenant un littéral L et s'il n'existe pas dans $S \setminus \{C\}$ d'occurrence d'un littéral de même symbole de prédicat et de même polarité que L, alors il existe une réfutation d'un ensemble d'instances de clauses de S comportant le même nombre d'application des règles de coupure et de contraction que $\mathcal{R}$ et dans laquelle toute occurrence d'un littéral de même symbole de prédicat que L et de polarité opposée à celle de L a pour atome une instance de l'atome de L.*

Preuve: Supposons que L soit le littéral $P(t_1 \ldots t_n)$. Puisqu'il n'existe pas dans $S \setminus \{C\}$ d'occurrence d'un littéral de même symbole de prédicat et de même polarité que L, toute occurrence d'un littéral de la forme $\neg P(t'_1 \ldots t'_n)$ dans une clause C' de $\mathcal{R}$ est unifiable avec L. Pour chaque clause C' de $\mathcal{R}$, il existe donc une substitution la plus générale, disons $\mu_{C'}$, telle que toute occurrence de $\neg P(s_1 \ldots s_n)$ dans $\mu_{C'}(C')$ soit une instance de L.

Nous pouvons donc construire par induction sur la taille de $\mathcal{R} =< C_1 \ldots C_p >$, déduction d'une clause C_p, une déduction $\mathcal{R}^*$ d'une instance de C_p telle que

- toute occurrence négative d'un littéral de symbole de prédicat P est une instance de L;
- à toute occurrence d'une clause C_i $1 \leq i \leq p$ de $\mathcal{R}$, il correspond une occurrence d'une clause $\mu_{C_i}(C_i)$ de $\mathcal{R}^*$.

Posons $\mathcal{R} = \mathcal{R}'. < C_p >$.

- Si $C_p \in S$, alors la déduction $\mathcal{R}'^*. < \mu_{C_p}(C_p) >$ est la déduction recherchée.
- Si $C_p = \sigma(C')$ est obtenue par instanciation à partir d'une clause $C_k = C'$, alors il existe dans $\mathcal{R}'^*$ une clause de la forme $\mu_{C_k}(C')$. Puisque μ_{C_k} est la substitution la plus générale telle que toute occurrence négative d'un littéral de symbole de prédicat P de $\mu_{C_k}(C')$ soit une instance de L, et puisque toute occurrence négative d'un littéral de symbole de prédicat P de $\mu_{C_p}(\sigma(C'))$

est une instance de L, il existe une substitution σ' telle que $\mu_{C_p}(\sigma(C')) = \sigma'(\mu_{C_k}(C'))$. On peut donc appliquer la règle d'instanciation sur la clause $\mu_{C_k}(C')$: on a $\mathcal{R}^* = \mathcal{R}'^*. < \mu_{C_p}(C_p) >$.

- Si $C_p = C' \vee L$ est obtenue par contraction à partir d'une clause $C_k = C' \vee L \vee L$, alors il existe dans $\mathcal{R}'^*$ une clause de la forme $\mu_{C_k}(C' \vee L \vee L)$ sur laquelle on peut appliquer la règle de contraction: on a $\mathcal{R}^* = \mathcal{R}'^*. < \mu_{C_p}(C_p) >$.

- Si $C_p = C'$ est obtenue par coupure à partir des clauses $C_j = C' \vee L$, $C_k = C' \vee L^c$, alors il existe dans $\mathcal{R}'^*$ des clauses de la forme $\mu_{C_j}(C' \vee L)$, $\mu_{C_k}(C' \vee L^c)$. On peut obtenir à partir de ces dernières les clauses $\mu_{C_p}(C' \vee L)$, $\mu_{C_p}(C' \vee L^c)$ par application de la règle de substitution. Il suffit alors d'appliquer la règle de coupure pour obtenir le résultat voulu. On a donc: $\mathcal{R}^* = \mathcal{R}'^*. < \mu_{C_p}(C_i), \mu_{C_p}(C_j), \mu_{C_p}(C_p) >$.

C.Q.F.D.

Soit $Rn+(\phi)$ la formule obtenue à partir de ϕ en renommant toutes les sous-formules de ϕ sauf les sous-formules atomiques et les sous-formules immédiates des sous-formules dont le connectif principal est une négation. Ce renommage est celui défini dans [PG86]. Soit $CRn+(\phi)$ la formule obtenue par mise sous forme clausale de $Rn+(\phi)$.

Théorème 9 *Soit ϕ une formule insatisfaisable et soit $\mathcal{R}$ une réfutation par Résolution de $CRn(\phi)$. Alors il existe une réfutation $\mathcal{R}'$ de $CRn+(\phi)$ qui comporte moins de pas d'inférence que la réfutation $\mathcal{R}$.*

Preuve: La réfutation $\mathcal{R}'$ est obtenue à partir de la réfutation $\mathcal{R}$ comme suit: soit $\{A_1 \ldots A_p\}$ l'ensemble des sous-formules atomiques de ϕ et soit $\{\phi_1 \ldots \phi_q\}$ l'ensemble des sous-formules immédiates des sous-formules ϕ dont le connectif principal est une négation. Soit $\mathcal{R}^*$ la déduction obtenue par remplacement uniforme des littéraux de Skolem $NP_{A_i}(x_1 \ldots x_{m_i})$ par A_i, $1 \leq i \leq p$ et par remplacement uniforme des littéraux de Skolem $NP_{\phi_i}(x_1 \ldots x_{n_i})$ par $\neg NP_{\neg\phi_i}(x_1 \ldots x_{n_i})$, $1 \leq i \leq q$ dans la réfutation $\mathcal{R}$. $\mathcal{R}^*$ comporte le même nombre d'inférences que $\mathcal{R}$. Puisque des sous-formules atomiques de polarité positive (resp. négative) ont été renommées, la déduction $\mathcal{R}$ comporte des clauses de la forme $\neg NP_{A_i}(x_1 \ldots x_{m_i}) \vee A_i$ (resp. $NP_{A_i}(x_1 \ldots x_{m_i}) \vee \neg A_i$). De même, le renommage des sous-formules immédiates des sous-formules de ϕ dont le connectif principal est une négation et dont la

polarité est positive (resp. négative) induit la présence de clauses de la forme: $\neg NP_{\neg\phi_i}(x_1 \ldots x_{n_i}) \vee \neg NP_{\phi_i}(x_1 \ldots x_{n_i})$ (resp. $NP_{\phi_i}(x_1 \ldots x_{n_i}) \vee NP_{\neg\phi_i}(x_1 \ldots x_{n_i})$). Après remplacement uniforme, ces clauses deviennent des clauses de la forme: $\neg A_i \vee A_i$, $\neg NP_{\neg\phi_i}(x_1 \ldots x_{n_i}) \vee NP_{\neg\phi_i}(x_1 \ldots x_{n_i})$. Donc la déduction $\mathcal{R}^*$ contient des tautologies. Ces tautologies peuvent être éliminées (cf. chapitre 1) et la déduction résultante comporte moins de pas d'inférence que $\mathcal{R}^*$ et donc que $\mathcal{R}$. Soit $\mathcal{R}'$ cette déduction. $\mathcal{R}'$ est une déduction de $CRn + (\phi)$ car chaque remplacement dans une clause initiale C de $\mathcal{R}$ de $NP_{A_i}(t_1 \ldots t_{m_i})$ (resp. $\neg NP_{A_i}(t_1 \ldots t_{m_i})$) par $A_i(x_1/t_1, \ldots, x_{m_i}/t_{m_i})$ (resp. $\neg A_i(x_1/t_1, \ldots, x_{m_i}/t_{m_i})$) peut être simulé par une application de la règle de coupure entre C et la clause $\neg NP_{A_i}(x_1 \ldots x_{m_i}) \vee A_i$ (resp. $NP_{A_i}(x_1 \ldots x_{m_i}) \vee \neg A_i$). De même, les remplacements des littéraux positifs (resp. négatifs) dont le symbole de prédicat est NP_{ϕ_i} peut être simulé par une application de la règle de coupure entre C et la clause $\neg NP_{\neg\phi_i}(x_1 \ldots x_{n_i}) \vee \neg NP_{\phi_i}(x_1 \ldots x_{n_i})$ (resp. $NP_{\phi_i}(x_1 \ldots x_{n_i}) \vee NP_{\neg\phi_i}(x_1 \ldots x_{n_i})$). C.Q.F.D.

On peut utiliser le lemme précédent pour éviter d'autres renommages inutiles: en particulier on montre que si ϕ une formule insatisfaisable, alors l'introduction d'un littéral de Skolem pour ϕ constitue un renommage inutile.

Nous allons tenter d'éliminer d'autres renommages inutiles. Pour ce faire, nous condèrerons désormais des formules ne contenant pas d'équivalence. Nous étudierons le cas de formules contenant des sous-formules de polarité nulle dans la section suivante.

Soit $Rn++(\phi)$ la formule obtenue à partir de ϕ en renommant toutes les sous-formules de ϕ à l'exception des sous-formules suivantes:

- les sous-formules atomiques
- les sous-formules immédiates ϕ_1 de sous-formules de la forme $\neg\phi_1$
- les sous-formules immédiates ϕ_1 de sous-formules de la forme $Qx\phi_1$, $Q \in \{\forall, \exists\}$

Théorème 10 *Soit ϕ une formule insatisfaisable et soit $\mathcal{R}$ une réfutation par Résolution de $CRn + (\phi)$. Alors il existe une réfutation par Résolution de $CRn++(\phi)$ comportant moins d'applications des règles de coupure et de contraction que $\mathcal{R}$.*

Preuve: Soit $\{\phi_1 \ldots \phi_m\}$ l'ensemble des sous-formules de ϕ telles que ϕ_i $1 \leq i \leq m$ soit une sous-formule immédiate d'une sous-formule de la forme $Q_i y_i \phi_i$. On a :

$Var(Q_i y_i \phi_i) \subseteq Var(\phi_i)$, $1 \leq i \leq m$. Nous supposerons que ces formules sont de polarité positive, le cas de formules de polarité négative étant similaire.

Suivant le quantificateur Q_i, le renommage de $Q_i y_i \phi_i$ produit l'une des clauses suivantes:

- si $Q_i = \forall$ et $\{x_1 \ldots x_n\} = Var(\forall y_i \phi_i)$:

$$\neg NP_{\forall y_i \phi_i}(x_1 \ldots x_n) \vee NP_{\phi_i}[x_1 \ldots x_n, y_i]$$

  $NP_{\phi_i}[x_1 \ldots x_n, y_i]$ est alors la seule occurrence d'un littéral positif ayant NP_{ϕ_i} pour symbole de prédicat et toute occurrence dans une clause de $CRn + (\phi)$ d'un littéral négatif ayant NP_{ϕ_i} pour symbole de prédicat est une instance de $\neg NP_{\phi_i}[x_1 \ldots x_n, y_i]$.

- si $Q_i = \exists$ et $\{x_1 \ldots x_n\} = Var(\exists y_i \phi_i)$:

$$\neg NP_{\exists y_i \phi_i}(x_1 \ldots x_n) \vee NP_{\phi_i}[x_1 \ldots x_n, f_{y_i}(x_1 \ldots x_n)]$$

  $NP_{\phi_i}[x_1 \ldots x_n, f_{y_i}(x_1 \ldots x_n)]$ est alors la seule occurrence d'un littéral positif ayant NP_{ϕ_i} pour symbole de prédicat et toute occurrence dans une clause de $\mathcal{R}$ d'un littéral négatif ayant NP_{ϕ_i} pour symbole de prédicat est unifiable avec $\neg NP_{\phi_i}[x_1 \ldots x_n, f_{y_i}(x_1 \ldots x_n)]$. On peut alors obtenir une déduction $\mathcal{R}'$ dans laquelle toute occurrence dans une clause de $\mathcal{R}$ d'un littéral négatif ayant NP_{ϕ_i} pour symbole de prédicat est une instance de $NP_{\phi_i}[x_1 \ldots x_n, f_{y_i}(x_1 \ldots x_n)]$.

Soit $\mathcal{R}''$ la réfutation obtenue à partir de $\mathcal{R}'$:

- par remplacement uniforme de $NP_{\phi_i}[x_1 \ldots x_n, y_i]$ par $NP_{\forall y_i \phi_i}(x_1 \ldots x_n)$
- par remplacement uniforme de $NP_{\phi_i}[x_1 \ldots x_n, f_{y_i}(x_1 \ldots x_n)]$ par $NP_{\exists y_i \phi_i}(x_1 \ldots x_n)$

Cette déduction contient le même nombre d'applications des règles de coupure et de contraction que $\mathcal{R}'$. Mais elle contient des tautologies de la forme:

- $\neg NP_{\forall y_i \phi_i}(t_1 \ldots t_n) \vee NP_{\forall y_i \phi_i}(t_1 \ldots t_n)$
- $\neg NP_{\exists y_i \phi_i}(t_1 \ldots t_n) \vee NP_{\exists y_i \phi_i}(t_1 \ldots t_n)$

issues des remplacements uniformes dans les clauses

$$\neg NP_{\forall y_i \phi_i}(x_1 \ldots x_n) \vee NP_{\phi_i}[x_1 \ldots x_n, y_i]$$

$$\neg NP_{\exists y_i \phi_i}(x_1 \ldots x_n) \vee NP_{\phi_i}[x_1 \ldots x_n, f_{y_i}(x_1 \ldots x_n)]$$

Ces tautologies peuvent être éliminées.

Comme dans le théorème précédent, chaque remplacement uniforme dans une clause C de $CRn+(\phi)$ figurant dans la déduction après élimination des tautologies peut être obtenu par application de la règle de coupure entre C et l'une des clauses:

$$\neg NP_{\forall y_i \phi_i}(x_1 \ldots x_n) \vee NP_{\phi_i}[x_1 \ldots x_n, y_i]$$

$$\neg NP_{\exists y_i \phi_i}(x_1 \ldots x_n) \vee NP_{\phi_i}[x_1 \ldots x_n, f_{y_i}(x_1 \ldots x_n)]$$

La réfutation obtenue après élimination des tautologies de $\mathcal{R}''$ est en fait une réfutation de $CRn++(\phi)$. C.Q.F.D.

Remarquons que si l'on renomme les sous-formules immédiates d'une conjonction $\bigwedge_{i=1}^n \phi_i$ de polarité positive ou d'une disjonction $\bigvee_{i=1}^n \phi_i$ de polarité négative, on obtient des clauses de la forme

$$\neg NP_{\bigwedge_{i=1}^n \phi_i}(x_1 \ldots x_n) \vee NP_{\phi_i}[x_1 \ldots x_n]$$

$$NP_{\bigvee_{i=1}^n \phi_i}(x_1 \ldots x_n) \vee \neg NP_{\phi_i}[x_1 \ldots x_n]$$

Il est tentant de vouloir éliminer ces renommages de manière similaire. Cependant , on ne peut appliquer la même technique que précédemment car en général $Var(\bigwedge_{i=1}^n \phi_i) \not\subseteq Var(\phi_i)$. Or dans ce cas, le phénomène suivant peut se produire:

Soit $S = \{C \vee R, \neg R \vee P(x), \neg P(a) \vee \neg P(b)\}$ un ensemble de clauses tel que C soit une disjonction de littéraux dans laquelle la variable x n'apparait pas et R soit une proposition (c'est à dire un symbole de prédicat d'arité nulle). Si, dans la déduction suivante,

$$\dfrac{C \vee R \qquad \dfrac{\dfrac{\dfrac{\neg R \vee P(x)}{\neg R \vee P(b)} \qquad \dfrac{\dfrac{\neg R \vee P(x)}{\neg R \vee P(a)} \qquad \neg P(a) \vee \neg P(b)}{\neg R \vee \neg P(b)}}{\neg R \vee \neg R}}{\neg R}}{C}$$

on cherche à remplacer uniformément R par $P(x)$ afin d'établir une déduction de C à partir de $\{C \vee P(x), \neg P(a) \vee \neg P(b)\}$, on obtient la déduction

$$\dfrac{\dfrac{\dfrac{\dfrac{C \vee P(x)}{C \vee P(b)} \qquad \dfrac{\dfrac{C \vee P(x)}{C \vee P(a)} \qquad \neg P(a) \vee \neg P(b)}{C \vee \neg P(b)}}{C \vee C}}{\text{plusieurs contractions}}}{C}$$

dans laquelle le nombre de contractions est plus important.

Nous allons cependant montrer que l'on peut éliminer de tels renommages si l'on utilise *certaines* stratégies de Résolution. Ce résultat est donc plus faible que ceux obtenus lors des deux théorèmes précédents, qui permettent d'identifier les renommages inutiles *quelle que soit* la stratégie utilisée.

Théorème 11 *Soit ϕ une formule insatisfaisable ne comportant pas d'équivalence et soient ϕ_1, ϕ_2 deux sous-formules de ϕ telles que ϕ_2 soit une sous-formule de ϕ_1. Soit R^1_ϕ la formule obtenue par renommage d'un sous-ensemble R_1 des sous-formules de ϕ et soit CR^1_ϕ la formule obtenue par mise sous forme clausale de R^1_ϕ. Soit R^2_ϕ la formule obtenue en renommant le sous-ensemble de sous-formules $R_1 \setminus \{\phi_2\}$ de ϕ et soit CR^2_ϕ la formule obtenue par mise sous forme clausale de R^2_ϕ. Supposons que CR^1_ϕ comporte une clause $L_1 \vee L_2$ telle que:*

- *L_1 est un littéral négatif dont le symbole de prédicat est NP_{ϕ_1}*
- *L_2 est un littéral positif dont le symbole de prédicat est NP_{ϕ_2} et il n'existe pas dans CR^1_ϕ d'autre occurrence de littéral L de même symbole de prédicat et de même polarité que L_2.*

Alors, pour toute réfutation $\mathcal{D}$ de CR^1_ϕ telle que toute application de la règle de coupure sur un littéral de symbole de prédicat NP_{ϕ_2} soit de la forme:

$$\frac{\dfrac{\mathcal{D}_1}{C \vee \sigma(L_2^c)} \quad \dfrac{L_1 \vee L_2}{\sigma(L_1 \vee L_2)}}{C \vee \sigma(L_1)}$$

il existe une réfutation de CR^2_ϕ comportant moins d'applications des règles de coupure et de contraction que $\mathcal{D}$.

Preuve: On peut transformer la réfutation $\mathcal{D}$ en une réfutation $\mathcal{D}'$ comportant le même nombre d'applications de la règle de coupure et de contraction que $\mathcal{D}$ et telle que toute occurrence d'un littéral de symbole de prédicat NP_{ϕ_2} dans $\mathcal{D}'$ a pour atome une instance de l'atome du littéral L_2.

Soit $V_1 = \{z_1 \ldots z_p\}$ l'ensemble des variables de L_1 ne figurant pas dans L_2 et soit ρ une substitution de renommage de domaine V_1 telle que pour $1 \leq i \leq p$ $\rho(z_i) = z'_i$ ne figure pas dans $\mathcal{D}'$. Soit L'_1 le littéral $\rho(L_1)$. Soit $\mathcal{D}''$ la déduction obtenue à partir de $\mathcal{D}'$ par substitution uniforme de L_2 par L'_1. Dans cette déduction toute les applications de la règle de coupure de la forme

$$\frac{\dfrac{\mathcal{D}_1}{C \vee \sigma(L_2^c)} \quad \dfrac{L_1 \vee L_2}{\sigma(L_1 \vee L_2)}}{C \vee \sigma(L_1)}$$

ont été remplacées par des applications de la règle de coupures de la forme:

$$\frac{\dfrac{\mathcal{D}_1}{C \vee \sigma(L_1'^c)} \quad \dfrac{L_1 \vee L_1'}{\sigma(L_1 \vee L_1')}}{C \vee \sigma(L_1)}$$

Soit μ la substitution de domaine $\{z_1' \ldots z_p'\}$ définie par $\mu(z_i') = \sigma(z_i)$, $1 \leq i \leq p$. On a:

$$\sigma(L_1) = \mu(\sigma(L_1'))$$

On peut remplacer dans $\mathcal{D}''$ les applications de la règle de coupure de la forme

$$\frac{\dfrac{\mathcal{D}_1}{C \vee \sigma(L_1'^c)} \quad \dfrac{L_1 \vee L_1'}{\sigma(L_1 \vee L_1')}}{C \vee \sigma(L_1)}$$

par une application de la règle d'instanciation en utilisant la substitution μ

$$\frac{\dfrac{\mathcal{D}_1}{C \vee \sigma(L_1'^c)}}{C \vee \sigma(L_1)}$$

Soit $\mathcal{D}^*$ la déduction ainsi obtenue.

Toute clause obtenue par remplacement dans une clause C de CR_ϕ^1 d'un littéral de symbole de prédicat NP_{ϕ_2} par une instance de L_1' est une variante de la clause que l'on peut construire par application de la règle de coupure entre C et $L_1 \vee L_2$. La réfutation $\mathcal{D}^*$ est en fait une réfutation de CR_ϕ^2. C.Q.F.D.

Définissons par induction mutuelle les ensembles de formules suivants:

- si A est une formule atomique alors A est une F^+-formule, A est une F^--formule,A est une F^0-formule;
- si A_1, A_2 sont des F^+-formules, B_1, B_2 sont des F^-- formules, C est une F^0-formule alors
  - $A_1 \wedge A_2$, QxA_1, $\neg B_1$ sont des F^+-formules ($Q \in \{\forall, \exists\}$);
  - $B_1 \vee B_2$, $A_1 \rightarrow B_2$, QxB_1, $\neg A_1$ sont des F^--formules ($Q \in \{\forall, \exists\}$);
  - QxC, $\neg C$ sont des F^0-formule ($Q \in \{\forall, \exists\}$).

Soit ϕ une formule insatisfaisable et soit $\prec$ un ordre sur les littéraux de $SCRn(\phi)$ défini par $L_1 \prec L_2$ si et seulement si:

- L_2 est une sous-formule atomique de ϕ et L_1 est un littéral de Skolem correspondant au renommage d'une sous-formule ϕ_1 de ϕ;

- L_1 et L_2 sont deux littéraux de Skolem correspondant respectivement au renommage d'une sous-formule ϕ_1 et d'une sous-formule ϕ_2 de ϕ telles que ϕ_2 soit une sous-formule de ϕ_1.

Théorème 12 *Soit ϕ une formule insatisfaisable. Supposons que l'on se propose de montrer l'insatisfaisabilité de ϕ en utilisant une stratégie de Résolution vérifiant que, pour toute application de la règle de coupure utilisant comme prémisse* une instance d'une clause C de $CRn + (\phi)$, *le littéral résolu de C soit maximal pour l'ordre $\prec$. Soit ϕ_1 une sous-formule de ϕ.*
Si ϕ_1 est une F^+-formule de polarité positive dans ϕ alors tout renommage d'une sous-formule stricte de ϕ_1 est inutile;
Si ϕ_1 est une F^--formule de polarité négative dans ϕ alors tout renommage d'une sous-formule stricte de ϕ_1 est inutile.

Preuve: Il suffit de vérifier pour chaque type de formule ϕ_1 que le renommage exhaustif de ϕ_1 ne produit que des clauses de la forme $L_1 \vee L_2$ et d'appliquer le lemme précédent à ces clauses. C.Q.F.D.

Nous pensons pouvoir étendre ce théorème aux formules comportant des équivalences en montrant que si ϕ_1 est une F^0-formule de polarité nulle dans ϕ alors tout renommage d'une sous-formule de ϕ_1 est inutile pour une stratégie utilisant l'ordre $\prec$.

6.3 Traitement des équivalences

Les résultats de la section précédente ne sont pas suffisants pour permettre de traiter efficacement le problème de formules contenant des équivalences. En effet, on sait qu'il est nécessaire de renommer de telles formules *avant* de procéder à la mise sous forme clausale afin d'éviter le comportement exponentiel de cette transformation. Cependant, dans bien des cas, il apparait souhaitable de limiter le plus possible le renommage de formules contenant des équivalences. En effet, si l'on considère la formule

$$P_1 \leftrightarrow (P_2 \leftrightarrow (P_1 \leftrightarrow P_2))$$

le renommage exhaustif produit le formule

$$(P_1 \leftrightarrow R_1) \wedge (R_1 \leftrightarrow (P_2 \leftrightarrow R_2)) \wedge (R_2 \leftrightarrow (P_1 \leftrightarrow P_2))$$

qui est plus difficile à réfuter que la formule originale!

On pourrait songer à linéariser une formule avant de la renommer mais la linéarisation d'une formule est une transformation exponentielle en taille. En fait on montre que [BdlT91]

$$|Lin(\phi)| \leq (|\phi| + 2N_{\leftrightarrow}(\phi))2^{D_{\leftrightarrow}(\phi)}$$

où $N_{\leftrightarrow}(\phi)$ désigne le nombre d'équivalences figurant dans ϕ et $D_{\leftrightarrow}(\phi)$, la profondeur en équivalences de ϕ, est défini par:

- $D_{\leftrightarrow}(A) = 0$ si A est une formule atomique
- $D_{\leftrightarrow}(\phi_1 \leftrightarrow \phi_2) = 1 + Max(D_{\leftrightarrow}(\phi_1), D_{\leftrightarrow}(\phi_2))$
- $D_{\leftrightarrow}(*_{i=1}^{n}\phi_i) = Max(D_{\leftrightarrow}(\phi_1) \ldots D_{\leftrightarrow}(\phi_n))$ où $* \in \{\vee, \neg, \wedge, \rightarrow\}$
- $D_{\leftrightarrow}(Qx\phi_1) = D_{\leftrightarrow}(\phi_1)$ où $Q \in \{\exists, \forall\}$

La formule 6.3 met en évidence qu'il est nécessaire de minimiser la profondeur en équivalences de $Rn(\phi)$. Rappelons que, si l'on emploie le renommage exhaustif, la profondeur en équivalences est au plus de deux. Ceci suggère de procéder en deux temps pour renommer une formule ϕ:

- lors d'une première phase, on parcours la formule en utilisant un algorithme descendant. Dès que l'on rencontre une formule ϕ_1 surmontée de deux équivalences ($D_{\leftrightarrow}(\phi_1) = 2$), celle-ci est renommée, ce qui crée la formule $\forall x_1 \ldots x_n \phi_1 \leftrightarrow NP_{\phi_1}(x_1 \ldots x_n)$ dans laquelle ϕ_1 n'est plus surmontée que d'une seule équivalence. On applique de nouveau la première phase à cette nouvelle formule.
- puis, on linéarise la formule obtenue et l'on applique sur le résultat une mise sous forme clausale utilisant le renommage exhaustif en tirant parti des nombreuses restrictions proposées dans la section précédente.

Lors de la linéarisation, il peut être utile de favoriser les études par cas: en effet, il peut être intéressant de linéariser une formule de la forme $C \wedge (\phi_1 \leftrightarrow \phi_2)$ où ϕ_1 et ϕ_2 sont des formules closes en $C \wedge ((\phi_1 \wedge \phi_2) \vee (\neg\phi_1 \wedge \neg\phi_2)$ puis de réfuter séparément les formules $C \wedge \phi_1 \wedge \phi_2$, $C \wedge \neg\phi_1 \wedge \neg\phi_2$.

6.4 Minimisation du nombre de clauses

Une manière de généraliser les conditions permettant d'éviter de renommer une sous-formule consiste à remarquer que l'élimination des renommages inutiles fait

décroître le nombre de clauses. D'où l'idée explorée par T. Boy de la Tour de chercher à définir une transformation qui minimise le nombre de clauses.

Dans [BdlT90], [BdlT91], on étudie une famille d'algorithmes qui sélectionnent l'ensemble des sous-formules à renommer en procédant comme suit:

$R_{inf}(\psi)$:

$R := \emptyset$

Tant qu'il existe une sous formule ϕ de ψ telle que

$|SCRn(\psi, R \cup \{\phi\})| \leq |SCRn(\psi, R)|$

Faire $R := R \cup \{\phi\}$

On obtient à partir de R_{inf} un algorithme de sélection des renommages à effectuer en choisissant une stratégie de sélection des sous-formules ϕ.

On montre que l'on peut obtenir un renommage optimal (c'est-à-dire un renommage parmi tout les renommages $R \subseteq SF(\phi)$ qui conduira à un nombre minimal de clauses) en calculant les renommages à effectuer en utilisant R_{inf}. Le nombre de clauses obtenues est en $O(SF(\psi))$ mais la taille de $CRn(\psi, R_{inf})$ est en $O(N|\psi|SF(\psi))$ où N est le nombre maximal de variables libres d'une sous-formule de ϕ, c'est-à-dire que $|CRn(\psi, R_{inf})|$ est en $O(|\psi|^3)$. Enfin, il est montré qu'il existe un algorithme R_{Opt} de la famille R_{inf} qui parcourt la formule ψ de haut en bas et qui calcule un renommage optimal dans le cas d'une formule ne comportant pas d'équivalence.

Un des aspects caractéristiques du renommage R_{Opt} peut être illustré sur l'exemple suivant. Considérons une formule de la forme $C \wedge (\vee_{i=1}^{p} C_i)$ où C, C_1, ... C_p désigne des clauses. Si l'on utilise le renomage exhaustif en tenant compte des améliorations décrites précédemment, on obtient les $p+1$ clauses suivantes: $C \vee NP_1(x_1 \ldots x_n)$, $\neg NP_1(x_1 \ldots x_n) \vee C_1$, $\ldots \neg NP_1(x_1 \ldots x_n) \vee C_p$. Par contre, si l'on utilise le renommage R_{Opt}, aucun renommage n'est effectué et l'on obtient les p clauses $C \vee C_1$, $\ldots C \vee C_p$. On remarque alors qu'il est nécessaire dans ce cas d'utiliser une stratégie de Résolution – comme la "Lock Resolution" – telle que les littéraux provenant des clauses $C_1 \ldots C_p$ soient résolus avant les littéraux provenant de C. Dans le cas contraire, il faudrait répéter les mêmes applications de règles d'inférence sur les littéraux de C pour chacune des clauses $C \vee C_i$ et pour chacune de leurs résolvantes contenant des littéraux issus de C, ce qui serait particulièrement inefficace.

Plaçons nous dans le cadre du calcul propositionnel et supposons que l'on résolve en premier lieu les littéraux de clauses C_i. Suppons que deux clauses C_i, C_j

possèdent des littéraux complémentaires. La résolvante de ces deux clauses contiendra deux fois les littéraux de C et il faudra $|C|$ contractions pour les factoriser. La même application de la règle de coupure sur les clauses $\neg NP_1 \vee C_i$, $\neg NP_1 \vee C_j$ produit une clause ne nécessitant qu'une seule application de la règle de contraction pour factoriser NP_1. De manière générale, les expérimentations réalisées en logique du premier ordre ont montré que, pour un problème et une stratégie de Résolution donnés, les réfutations d'ensembles de clauses obtenus en utilisant le renommage R_{Opt} comportent moins de pas de résolution et plus de pas de factorisation que les réfutations d'ensembles de clauses produits en utilisant le renommage exhaustif amélioré. Dans le cas du démonstrateur par Résolution d'ATINF, la factorisation est une opération moins coûteuse que la production de résolvantes par résolution. Aussi le renommage R_{Opt} permet-il un gain d'efficacité.

Un autre renommage intéressant est le renommage qui tente de minimiser le nombre de littéraux. Une telle transformation est décrite dans [BdlT91]. Comme R_{Opt}, un tel renommage évite les renommages inutiles décrits précédemment.

Cependant, on peut montrer que la minimisation du nombre de clauses ou de littéraux ne facilite pas toujours la recherche d'une réfutation. Considérons la formule

$$\begin{gathered}\forall x_1 Q_1(x_1) \vee \forall x_2 Q_2(x_1, x_2) \vee \ldots \vee \forall x_n Q_n(x_1, \ldots, x_n) \\ \wedge \wedge_{i=1}^{n} (\neg Q_i(x_1, \ldots, x_{i-1}, a) \vee \neg Q_i(x_1, \ldots, x_{i-1}, b))\end{gathered}$$

Pour une telle formule, que l'on minimise le nombre de clauses ou de littéraux, le renommage optimal est le renommage vide. Remarquons que si l'on résoud sur Q_i avant de résoudre sur Q_{i+1}, la preuve obtenue comprend 2^n pas de résolution. Si l'ordre inverse est choisi, on obtient une preuve comprenant n pas de résolution mais $n(n-1)$ pas de factorisation. La preuve la plus courte est donc en $O(n^2)$ inférences.

Considérons la formule sous forme clausale suivante, qui peut être obtenue à partir de la précédente par renommage:

$$\begin{gathered}\forall x_1 Q_1(x_1) \vee NP_1(x_1) \\ \wedge_{i=2}^{n-1} (\forall x_1, \ldots, x_i \neg NP_{i-1}(x_1, \ldots, x_{i-1}) \vee Q_i(x_1, \ldots, x_i) \vee NP_i(x_1, \ldots, x_i)) \\ \wedge \forall x_1, \ldots, x_n \neg NP_{n-1}(x_1, \ldots, x_{n-1}) \vee Q_n(x_1, \ldots, x_n) \\ \wedge \wedge_{i=1}^{n} (\neg Q_i(x_1, \ldots, x_{i-1}, a) \vee \neg Q_i(x_1, \ldots, x_{i-1}, b))\end{gathered}$$

Il existe une réfutation de cet ensemble de clauses comportant $3n - 1$ pas de résolution et $n - 1$ pas de factorisation.

6.5 Résultats expérimentaux

Nous avons essayé plusieurs transformations sous forme clausale sur un problème réputé difficile du à P. Andrews [HLO+80]:

$$\begin{aligned} I \;=\; & \neg[\exists x \forall y (P(x) \leftrightarrow P(y)) \leftrightarrow (\exists x Q(x) \leftrightarrow \forall y P(y)) \\ & \leftrightarrow \exists x \forall y (Q(x) \leftrightarrow Q(y)) \leftrightarrow (\exists x P(x) \leftrightarrow \forall y Q(y))] \end{aligned}$$

Dans un premier temps (I), nous avons utilisé le fait que cette formule est de la forme $\neg(\phi_1 \leftrightarrow \phi_2)$ où ϕ_1, ϕ_2 sont des formules sans variables libres, pour transformer cette formule en une disjonction de la forme $(\neg\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \neg\phi_2)$. Cette formule est insatisfaisable si et seulement si chacun des disjoints est insatisfaisable. Nous avons donc chercher à réfuter chacun des disjoints séparément.

Le deuxième jeux de résulats (II) est obtenu en réfutant directement la formule sans procéder à une quelconque étude de cas.

Chacun des essais I et II est divisé en deux parties: dans la première (1), les renommages sont appliqués directement à la formule considérée tandis que dans la seconde (2) les renommages sont appliqués au résultat de la linéarisation de la formule considérée.

Enfin, nous considérons (III) la formule obtenue à partir de I en renommant sytématiquement les sous-formules dont la profondeur en équivalence est 2, comme expliqué au paragraphe 6.3 relatif au traitement des équivalences. Puis la formule est linéarisé et chacun des renommages est appliquée à la formule ainsi obtenue.

Les trois transformations sous forme clausale qui ont été étudiées ici sont la forme clausale standard (correspondant au cas noté $\emptyset$ car cette transformation renomme un ensemble vide de sous-formule), le forme clausale obtenue en utilisant le renommage R_{Opt} et enfin une modification du renommage décrit dans [PG86]: ce renommage, que nous appelerons ici $sp+$, ne renomme pas les sous-formules strictes des F^0-formules de polarité non nulle; de plus, on ne renomme au plus qu'une seule sous-formule immédiate d'une formule $\phi_1 \leftrightarrow \phi_2$ et ce afin de limiter le nombre de renommages en présence d'une équivalence (cf. 6.3). Cette transformation est donc exponentielle sauf dans l'expérimentation III.

Il convient de noter que, pour le problème d'Andrews, les résultats obtenus en utilisant le renommage minimisant le nombre de littéraux sont à peu de choses près identiques à ceux obtenus en utilisant le renommage R_{Opt}. De même, un renommage mettant pleinement à profit l'existence du théorème 12 et supposant également que

Table 6.1: Resultats

		p	$\|R\|$	t	r	$\bar{n}$	f
I.1	$\emptyset$	$16+16$	$0+0$	$8.3+6.4$	$12+10$	$11.5+10.5$	$25\%+30\%$
	R_{opt}	$16+16$	$0+0$	$8.3+6.4$	$12+10$	$11.5+10.5$	$25\%+30\%$
	$sp+$	$16+16$	$2+2$	$241+347$	$19+14$	$60+93$	$2.4\%+2.9\%$
I.2	$\emptyset$	$16+16$	$0+0$	$8.3+6.4$	$12+10$	$11.5+10.5$	$25\%+30\%$
	R_{opt}	$16+16$	$0+0$	$8.3+6.4$	$12+10$	$11.5+10.5$	$25\%+30\%$
	$sp+$	$32+32$	$20+20$	$46+45$	$28+28$	$26.8+26.4$	$10.5\%+10.5\%$
II.1	$\emptyset$	128	0	—	—	—	—
	R_{opt}	24	3	28.3	19	27.5	10.2%
	$sp+$	24	3	587	26	60	3.2%
II.2	$\emptyset$	128	0	—	—	—	—
	R_{opt}	32	2	816	36	62	2.7%
	$sp+$	66	44	—	—	—	—
III	$\emptyset$	24	4	27.5	22	23.2	10.5%
	R_{opt}	24	4	27.5	22	23.2	10.5%
	$sp+$	46	26	200	50	26.3	10.1%

le renommage des F^0 formules de polarité nulle est inutile permettrait d'obtenir des résultats du même ordre de grandeur que ceux obtenus en utilisant R_{Opt}.

Les expérimentations sont effectuées avec le démonstrateur par Résolution que nous avons écrit dans le cadre d'Atinf. La stratégie employée est une stratégie d'ordonnancement vérifiant les conditions suivantes:

- les littéraux P, Q figurant dans la formule initiale sont résolus avant les littéraux de Skolem et P est résolu avant Q.
- les littéraux de Skolem sont ordonnés de sorte que, si ϕ_1 est une sous-formule de ϕ_2, alors le littéral de Skolem de symbole de prédicat NP_{ϕ_1} est résolu avant le littéral de Skolem de symbole de prédicat NP_{ϕ_2}.

De plus, il est important de préciser que, dans le cas où deux littéraux de même symbole de prédicat figurant dans une même clause peuvent être résolus, on tente de ne résoudre que sur un seul de ces deux littéraux en comparant les termes *non variables* de ceux-ci. Donc, moins une clause comporte de variables libres et plus l'ordre utilisé est discriminant.

Par la suite, nous appelerons "cycle" la recherche parmi un ensemble de clauses des clauses les plus courtes et la production à partir de celles-ci de toutes les résolvantes ayant au moins une de ces clauses pour parent ainsi que le traitement (factorisation, subsomption, calcul des littéraux sur lesquels la résolution est possible, rangement dans la base de clauses, ...) de ces résolvantes.

Les résultats sont décrits à la figure 6.5: p désigne le nombre de clauses obtenues lors de la mise sous forme clausale, $|R|$ désigne le nombre de renommages, t est la durée exprimée en secondes de la recherche d'une réfutation, r est le nombre de cycles ("runs"), $\bar{n}$ est le nombre moyen de clauses engendrées par cycle et f est le "focus", c'est-à-dire le rapport du nombre de clauses utilisées dans la preuve au nombre de clauses générées. Remarquons que $\bar{n}$ et f donnent des indications sur la taille de l'espace de recherche.

Considérons tout d'abord les résultats obtenus lors de l'expérimentation III. Remarquons que la minimisation de la profondeur en équivalences permet de trouver rapidement et assez facilement une réfutation et ce, même si la forme clausale standard est utilisée. Le fait que R_{Opt} n'effectue pas d'autres renommages après limitation de la profondeur en équivalence à deux témoigne de ce que, dans la cas du problème d'Andrews, ces renommages sont suffisants pour qu'une forme clausale "simple" soit obtenue par utilisation de la mise sous forme clausale standard. En

revanche, l'utilisation du renommage $sp+$ crée de nombreux renommages inutiles. Pour éliminer ces renommages, il faut effectuer de nombreux pas de Résolution et donc plus de cycles sont nécessaires. Il en résulte une augmentation du nombre de résolvantes produites et donc du temps de traitement.

Considérons maintenant le problème I. Ici encore la profondeur en équivalences de chacun des sous-problèmes est de deux: ceci est obtenu en favorisant les études par cas. Comme dans l'exemple III, les formes clausales obtenues en utilisant le renommage R_{Opt} ou le renommage vide (mise sous forme clausale standard) sont identiques. Le problème d'Andrews est alors facilement réfuté. Les résultats obtenus avec le renommage $sp+$ sont tout autre:

- dans le cas I.1 où ce renommage est appliqué avant la linéarisation, seulement deux formules sont renommées et le nombre de clauses est identique à celui obtenu avec les deux autres transformations. Cependant, les clauses issues de la transformation $sp+$ contiennent plus de variables libres: ceci augmente le nombre de littéraux potentiellement unifiables et rend aussi moins efficace la stratégie d'ordonnancement. Il en résulte une dégradation sensible de l'espace de recherche (cf valeurs de $\bar{n}$ et f) et donc des performances.

- dans le cas I.2, ce phénomène se reproduit dans une moindre mesure (cf valeurs de $\bar{n}$ et f) mais se conjugue avec un nombre excessif de renommages inutiles ($|R| = 20$). Ici encore, il en résulte une augmentation du temps nécessaire à l'obtention d'une réfutation.

L'expérimentation II.1 met en évidence que la création de définitions comprenant le moins possible de variables libres est un facteur déterminant d'efficacité. En effet, les transformations R_{Opt} et $sp+$ sont appliquées directement sur la formule I et produisent le même nombre de clauses en utilisant le même nombre de renommages. Cependant, l'algorithme calculant R_{Opt} est un algorithme descendant et renomme des sous-formules plus proches de la racine (si l'on considère une représentation sous forme d'arbre de la formule I) que le renommage $sp+$: les définitions ainsi crées comportent alors peu de variables libres. Remarquons enfin que, dans le cas II.2 où les différents renommmages sont appliqués après linéarisation de I, seul le renommage R_{Opt} permet de trouver une réfutation. Cependant, les performances sont très mauvaises: ceci montre qu'il n'est pas souhaitable de linéariser une formule avant de la renommer .

En conclusion, ces expérimentations montre que l'élimination des renommages

inutiles et que la limitation de la profondeur en équivalences contribuent de manière sensible à l'amélioration de l'efficacité de la Résolution. De plus, il est nécessaire de minimiser le nombre de variables libres figurant dans les clauses obtenues après renommage afin de ne pas trop augmenter l'espace de recherche.

6.6 Une restriction de l'espace de recherche de la Résolution

Nous allons montrer que, si l'on considère une stratégie de Résolution permettant de construire des preuves correspondants à des déductions sans coupure dans LK', alors l'espace de recherche de la Résolution est plus grand que celui du calcul des séquents. En effet, il est possible de déduire par Résolution des clauses qui ne seront pas utilisées dans la réfutation car elles ne correspondent pas à des séquents pouvant figurer dans une déduction de LK'.

Considérons le calcul des séquents LK'' qui comprend les mêmes axiomes et règles d'inférence que LK à l'exception des règles $\wedge - IA$, $\vee - IC$, $\rightarrow - IA$ qui sont respectivement remplacées par les règles:

- $\wedge - IA''$ $\dfrac{\phi_1, \phi_2, \Gamma \Longrightarrow \Delta}{\phi_1 \wedge \phi_2, \Gamma \Longrightarrow \Delta}$
- $\vee - IC''$ $\dfrac{\Gamma \Longrightarrow \Delta, \phi_1, \phi_2}{\Gamma \Longrightarrow \Delta, \phi_1 \vee \phi_2}$
- $\rightarrow - IA''$ $\dfrac{\Gamma \Longrightarrow \Delta, \phi_1, \quad \phi_2, \Gamma \Longrightarrow \Delta}{\phi_1 \rightarrow \phi_2, \Gamma \Longrightarrow \Delta}$

Toute preuve du calcul LK'' peut être traduite en une preuve de LK comme le montrent les trois déductions suivantes:

- $$\dfrac{\dfrac{\dfrac{\dfrac{\phi_1, \phi_2, \Gamma \Longrightarrow \Delta}{\phi_1 \wedge \phi_2, \phi_2, \Gamma \Longrightarrow \Delta}}{\phi_2, \phi_1 \wedge \phi_2, \Gamma \Longrightarrow \Delta}}{\phi_1 \wedge \phi_2, \phi_1 \wedge \phi_2, \Gamma \Longrightarrow \Delta}}{\phi_1 \wedge \phi_2, \Gamma \Longrightarrow \Delta}$$
- $$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \Longrightarrow \Delta, \phi_1, \phi_2}{\Gamma \Longrightarrow \Delta, \phi_1, \phi_1 \vee \phi_2}}{\Gamma \Longrightarrow \Delta, \phi_1 \vee \phi_2, \phi_1}}{\Gamma \Longrightarrow \Delta, \phi_1 \vee \phi_2, \phi_1 \vee \phi_2}}{\Gamma \Longrightarrow \Delta, \phi_1 \vee \phi_2}$$
- $$\dfrac{\dfrac{\phi_2, \Gamma \Longrightarrow \Delta \quad \Gamma \Longrightarrow \Delta, \phi_1}{\phi_1 \rightarrow \phi_2, \Gamma, \Gamma \Longrightarrow \Delta, \Delta}}{\dfrac{\text{échanges et contractions}}{\phi_1 \rightarrow \phi_2, \Gamma \Longrightarrow \Delta}}$$

Considérons la formule $\phi = ((A \vee \neg A) \wedge (A \vee \neg A))$. Il existe une unique dérivation dans LK'' de ϕ: $\dfrac{\dfrac{\dfrac{A \Longrightarrow A}{\Longrightarrow A, \neg A}}{\Longrightarrow A \vee \neg A} \quad \dfrac{\dfrac{A \Longrightarrow A}{\Longrightarrow A, \neg A}}{\Longrightarrow A \vee \neg A}}{\Longrightarrow ((A \vee \neg A) \wedge (A \vee \neg A))}$

Afin de pouvoir distinguer les diverses occurrences de la formule atomique A dans ϕ, notons A^i la i-ème occurrence de A dans ϕ et écrivons la formule ϕ sous la forme: $((A^1 \vee \neg A^2) \wedge (A^3 \vee \neg A^4))$. $Rn + (\neg\phi)$ est la formule

$$(\neg NP \wedge ((NP_1 \wedge NP_2) \rightarrow NP) \wedge ((A^1 \vee \neg A^2) \rightarrow NP_1) \wedge ((A^3 \vee \neg A^4) \rightarrow NP_2)$$

$CRn + (\neg\phi)$ contient les clauses suivantes:

1. $\neg NP$
2. $(\neg NP_1 \vee \neg NP_2 \vee NP)$
3. $(\neg A^1 \vee NP_1)$
4. $(A^2 \vee NP_1)$
5. $(\neg A^3 \vee NP_2)$
6. $(A^4 \vee NP_2)$

Soit $I = \{A\}$ une partition de $CRn + (\neg\phi)$ et soit $NP < NP_2 < NP_1 < A$ un ordre sur les littéraux de $CRn + (\neg\phi)$. Si l'on utilise la Résolution Sémantique Ordonnée, il existe quatre résolvantes possibles à partir de $CRn + (\neg\phi)$:

- $(NP_1 \vee NP_1)$, résolvante de 3 et 4
- $(NP_2 \vee NP_2)$, résolvante de 5 et 6
- $(NP_1 \vee NP_2)$, résolvante de 3 et 6
- $(NP_1 \vee NP_2)$, résolvante de 4 et 5

Seules les deux premières résolvantes conduisent à une réfutation. Les deux dernières ne peuvent produire que des tautologies et sont donc inutiles. En appliquant la propriété de la sous-formule au calcul LK'', nous allons donner une condition permettant d'empêcher la production de ces résolvantes.

Définition: Soit ϕ une formule valide ne comportant pas d'équivalence et soient O_1, O_2 deux occurrences de sous-formules atomiques A_1, A_2 de ϕ telles qu'il existe deux substitutions σ_1, σ_2 telles que $\sigma_1(A_1) = \sigma_2(A_2)$.

(A_1, σ_1) et (A_2, σ_2) sont *potentiellement connectées* si et seulement si $PC(\phi, O_1, \sigma_1, O_2, \sigma_2, 1) = 1$ où

$PC(A, \Lambda, \sigma_1, \Lambda, \sigma_2, p) = -1$
si A est une formule atomique
$PC(\Lambda_{i=1}^n \phi_i, i_1.O_1, \sigma_1, i_2.O_2, \sigma_2, p) =$
$-p$ si $i_1 \neq i_2$
$PC(\phi_i, O_1, \sigma_1, O_2, \sigma_2, p)$ si $i_1 = i_2 = i$
$PC(\bigvee_{i=1}^n \phi_i, i_1.O_1, \sigma_1, i_2.O_2, \sigma_2, p) =$
p si $i_1 \neq i_2$
$PC(\phi_i, O_1, \sigma_1, O_2, \sigma_2, p)$ si $i_1 = i_2 = i$
$PC(\phi_1 \rightarrow \phi_2, i_1.O_1, \sigma_1, i_2.O_2, \sigma_2, 1) =$
p si $i_1 \neq i_2$
$PC(\phi_1, O_1, \sigma_1, O_2, \sigma_2, -p)$ si $i_1 = i_2 = 1$
$PC(\phi_2, O_1, \sigma_1, O_2, \sigma_2, p)$ si $i_1 = i_2 = 2$
$PC(\neg\phi_1, 1.O_1, \sigma_1, 1.O_2, \sigma_2, p) =$
$PC(\phi_1, O_1, \sigma_1, O_2, \sigma_2, -p)$
$PC(\forall x \phi_1, 1.O_1, \sigma_1, 1.O_2, \sigma_2, p) =$
1 si $\sigma_1(x) \neq \sigma_2(x)$ et $p = -1$
$PC(\phi_1, O_1, \sigma_1, O_2, \sigma_2, p)$ sinon
$PC(\exists x \phi_1, 1.O_1, \sigma_1, 1.O_2, \sigma_2, p) =$
1 si $\sigma_1(x) \neq \sigma_2(x)$ et $p = 1$
$PC(\phi_1, O_1, \sigma_1, O_2, \sigma_2, p)$ sinon

Les sous-formules $\sigma_1(A_1)$ et $\sigma_2(A_2)$ sont dites *compatibles* si

- A_1 et A_2 sont deux sous-formules de polarité opposée
- (A_1, σ_1) et (A_2, σ_2) sont potentiellement connectées

$\diamond$

Dans l'exemple ci-dessus, les sous-formules A^1 et A^2 (resp. A^3 et A^4) sont compatibles alors que les sous-formules A^1 et A^4 (resp. A^2 et A^3) ne le sont pas.

Théorème 13 *Soit ϕ une formule valide. Alors il existe une déduction dans LK' de ϕ telle que tout axiome $S = A \Longrightarrow A$ soit tel que*

- *la formule A est partie antécedent (resp. conséquent) de S est une instance d'une sous-formule atomique de ϕ, apparaissant à l'occurrence O_1 (resp. O_2) dans ϕ et $A = \sigma_1(\phi\,|_{O_1})$ (resp. $A = \sigma_2(\phi\,|_{O_2})$)*
- *les sous-formules $\sigma_1(\phi\,|_{O_1})$ et $\sigma_2(\phi\,|_{O_2})$ sont compatibles*

Preuve: Pour montrer ce théorème, on utilise la preuve de complétude du calcul des séquents LK telle qu'elle est présentée dans [Gal86]:

- on montre d'abord la complétude d'un calcul des séquents (appelé système G). Ce calcul est essentiellement le calcul LK'' .
- puis on montre que toute preuve du calcul G peut être traduite en une preuve du calcul LK.

La preuve de complétude du système G est similaire à la preuve de complétude de la méthode des tableaux [Smu68]. Gallier montre en fait que:

Théorème 14 *Soit ϕ une formule valide. Alors il existe une dérivation dans LK'' du séquent $\Longrightarrow \phi$. De plus, dans cette dérivation, toutes les applications des règles $\forall - IA$,$\exists - IC$ et des règles de contractions sont de la forme:*

$$\dfrac{\dfrac{\dfrac{\phi_1(x/t_1),\ldots,\phi_1(x/t_n),\Gamma \Longrightarrow \Delta}{n \text{ applications de } \forall - IA}}{\forall x\phi_1,\ldots,\forall x\phi_1,\Gamma \Longrightarrow \Delta}}{\dfrac{n \text{ contractions de } \forall x\phi_1}{\forall x\phi_1,\Gamma \Longrightarrow \Delta}} \qquad \dfrac{\dfrac{\dfrac{\Gamma \Longrightarrow \Delta,\phi_1(x/t_1),\ldots,\phi_1(x/t_n)}{n \text{ applications de } \exists - IC}}{\Gamma \Longrightarrow \Delta,\exists x\phi_1,\ldots,\exists x\phi_1}}{\dfrac{n \text{ contractions de } \exists x\phi_1}{\Gamma \Longrightarrow \Delta,\exists x\phi_1}}$$

où $t_1,\ldots,t_n$ sont des termes deux à deux distincts.

Identifions ces déductions à des règles d'inférence et supposons que le calcul LK'' comporte désormais les règles

- $\forall - IA''\ \dfrac{\phi_1(x/t_1),\ldots,\phi_1(x/t_n),\Gamma \Longrightarrow \Delta}{\forall x\phi_1,\Gamma \Longrightarrow \Delta}$
- $\exists - IC''\ \dfrac{\Gamma \Longrightarrow \Delta,\phi_1(x/t_1),\ldots,\phi_1(x/t_n)}{\Gamma \Longrightarrow \Delta,\exists x\phi_1}$

à la place des règles $\forall - IA''$, $\exists - IC''$ et de la règle de contraction.

Etant donnée une déduction $\mathcal{D}$ d'une formule valide ϕ dans ce nouveau calcul, on vérifie aisément que

Lemme 19 *Soient ϕ_1,ϕ_2 deux formules* distinctes *d'un séquent de $\mathcal{D}$ et soit A_1 (resp. A_2) une sous-formule atomique de ϕ_1 (resp. ϕ_2). Alors il existe une occurrence O_1 (resp. O_2) d'une sous-formules atomique de ϕ et une substitution σ_1 (resp.*

σ_2) telles que $\sigma_1(\phi\,|_{o_1}) = A_1$ (resp. $\sigma_2(\phi\,|_{o\ 2}) = A_2$) et $\sigma_1(\phi\,|_{o_1}), \sigma_2(\phi\,|_{o_2})$ désigne des occurrences compatibles.

En effet, cette propriété est triviallement vérifiée par le séquent $\Longrightarrow \phi$ puisque ce séquent ne comporte pas deux formules distinctes. Pour chacune des règles d'inférence de LK', si la propriété est vérifiée par la conclusion de la règle d'inférence alors elle est vérifiée par la (ou les) prémisse(s) de la règle. (Il s'agit en fait d'appliquer la propriété de la sous-formule au calcul LK''.) Si l'on applique ce lemme à un axiome de $\mathcal{D}$, on obtient le résultat recherché pour le calcul LK''. Ce résultat est aussi applicable aux calculs LK et LK' étant donnée la nature des traductions des preuves du calcul LK'' dans ces deux calculs. C.Q.F.D.

Corollaire 3 *Soit ϕ une formule valide ne comportant pas d'équivalence et soit une stratégie de Résolution permettant de construire des preuves correspondant à des déductions sans coupure dans LK'. Alors il existe une réfutation par Résolution dans laquelle toute application de la règle de coupure sur des instances de sous-formules atomiques de ϕ est une application de la règle de coupure entre formules compatibles.*

L'application de ce résultat permet d'éviter la production des résolvantes inutiles dans l'exemple précédent. En pratique, il est facile d'établir une correspondance entre occurrences de formules atomiques dans la formule initiale et occurrence de formules atomiques dans la forme clausale et d'utiliser les résultats précédents pour limiter l'espace de recherche de la Résolution.

6.7 Skolemisation

Soit ϕ une formule ne comportant pas d'équivalence et soit N le nombre maximal de variables libres d'une sous-formule de ϕ. On a $N \leq |SF(\phi)|$. La skolemisation de ϕ conduit à remplacer plusieurs occurrences de variables x par un terme de Skolem de la forme $f_x(x_1 \ldots x_n)$ comportant au plus $N+1$ symboles. Le nombre d'occurrences de telles variables x dans une sous-formule atomique A de ϕ est plus $|A|$ et donc le nombre total d'occurrences de telles variables est inférieur à $\sum_{A \in SFA(\phi)} |A| \leq |\phi|$. Donc si ϕ' est obtenue à partir de ϕ par skolemisation, $|\phi'|$ est en $O(|\phi||SF(\phi)|)$.

Cependant, il peut être intéressant en pratique de skolemiser une formule ϕ avant de procéder au renommage de ϕ car l'introduction de termes de Skolem réduit le nombre de littéraux potentiellement unifiables, diminue le nombre de variables

libres apparaissant dans les sous-formules de ϕ et contribue à limiter la taille de l'espace de recherche de la Résolution. Ceci est particulièrement intéressant dans le cas de formules de la forme $\exists x_1 \ldots x_n \phi$ car la skolemisation introduit des termes clos. En outre, dans ce cas particulier, la skolemisation peut rendre toutes les occurrences de prédicats n-aires ($n \geq 2$) assimilables à des prédicats unaires qui pourront ultérieurement donner lieu à une transformation de spécification visant à introduire des sortes à la place de ces prédicats unaires. Une telle transformation de spécification est décrite dans [SS87].

6.8 Utilisation de simplifications non clausales

L'utilisation de règles de simplification permet de faire décroitre la taille de la formule à réfuter et facilite donc la recherche d'une réfutation. Nous présentons ici quelques règles de simplification en utilisant la notation simplifiée $\frac{\psi[\phi]}{\psi[\phi']}$ pour désigner que l'on remplace dans la formule ψ la sous-formule ϕ par la formule ϕ':

1. $\frac{\psi[(\forall x_1 \ldots x_n L) \wedge \phi]}{\psi[(\forall x_1 \ldots x_n L) \wedge \phi']}$ où L est un littéral et ϕ' est la formule obtenue en remplaçant dans ϕ
   - toute occurrence d'un littéral de la forme $\sigma(L)$ tel que $Dom(\sigma) = \{x_1 \ldots x_n\}$ par $Vrai$.
   - toute occurrence d'un littéral de la forme $\sigma(L^c)$ tel que $Dom(\sigma) = \{x_1 \ldots x_n\}$ par $Faux$.
2. $\frac{\psi[(\exists x_1 \ldots x_n L) \vee \phi]}{\psi[(\exists x_1 \ldots x_n L) \vee \phi']}$ où L est un littéral et ϕ' est la formule obtenue en remplaçant dans ϕ
   - toute occurrence d'un littéral de la forme $\sigma(L)$ tel que $Dom(\sigma) = \{x_1 \ldots x_n\}$ par $Faux$.
   - toute occurrence d'un littéral de la forme $\sigma(L^c)$ tel que $Dom(\sigma) = \{x_1 \ldots x_n\}$ par $Vrai$.
3. $\frac{\psi[\exists x(x = t \wedge \phi)]}{\psi[\exists x \phi(x/t)]}$ où x est une variable n'apparaissant pas dans t
4. $\frac{\psi[\forall x(x \neq t \vee \phi)]}{\psi[\phi(x/t)]}$ où x est une variable n'apparaissant pas dans t
5. $\frac{\psi[\phi \leftrightarrow \phi]}{\psi[true]}$

6. $\dfrac{\psi[\phi_1 \leftrightarrow \neg\phi_2]}{\psi[\neg(\phi_1 \leftrightarrow \phi_2)]}$

7. $\dfrac{\psi[\phi_1 \leftrightarrow (\phi_2 \leftrightarrow \phi_3)]}{\psi[(\phi_1 \leftrightarrow \phi_2) \leftrightarrow \phi_3]}$

8. $\dfrac{\psi[(\phi_1 \leftrightarrow \phi_2) \leftrightarrow (\phi_3 \leftrightarrow \phi_4)]}{\psi[(\phi_1 \leftrightarrow (\phi_2 \leftrightarrow (\phi_3 \leftrightarrow \phi_4)))]}$

On peut montrer que les quatre dernières règles de simplification permettent de décider de la validité d'une formule propositionnelle ϕ construite sur l'ensemble de connectifs $\{\leftrightarrow, \neg\}$ en temps $O(|\phi| \log(|\phi|))$.

L'utilisation des quatre premières règles de simplification permet en fait de réaliser directement sur la formule ϕ des simplifiations que l'on pourrait effectuer sur la forme clausale standard de ϕ.

Par exemple, considérons la formule propositionnelle $A \vee (B \wedge ((B \wedge C) \vee D))$. La mise sous forme clausale de cette formule produit les clauses $A \vee B$, $A \vee B \vee D$, $A \vee C \vee D$. La deuxième clause est subsumée par la première et peut être éliminée. L'ensemble de clauses restant correspond à la mise sous forme clausale de $A \vee (B \wedge (C \vee D))$.

De même, la mise sous forme clausale de la formule $A \vee (B \wedge ((\neg B \wedge C) \vee D))$ produit les clauses $A \vee B$, $A \vee \neg B \vee D$, $A \vee C \vee D$. En résolvant les deux premières clauses entre elles, on obtient la clause $A \vee D$ qui, par subsomption, permet d'éliminer les clauses $A \vee \neg B \vee D$, $A \vee C \vee D$. L'ensemble de clauses restant correspond à la mise sous forme clausale de $A \vee (B \wedge D)$.

Remarquons que si l'on renomme la sous formule $(B \wedge C)$ dans le premier exemple (ou la formule $(\neg B \wedge C)$ dans le deuxième exemple), on ne peut plus appliquer ces simplifications. Il donc nécessaire de simplifier une formule avant de procéder à un quelconque renommage.

Ces règles de simplification permettent de démontrer facilement des théorèmes simples de la théorie naïve des ensembles, tels ceux étudiés dans [PG86]. Les axiomes de cette théorie sont:

- $A =_{set} B \equiv_{def} ((A \subseteq B) \wedge (B \subseteq A))$
- $A \subseteq B \equiv_{def} (\forall x\ x \in A \rightarrow x \in B)$
- $x \in (A \cup B) \equiv_{def} (x \in A \vee x \in B)$
- $x \in (A \cap B) \equiv_{def} (x \in A \wedge x \in B)$

- $x \in P(A) \equiv_{def} (x \subseteq A)$

Nous allons montrer sur deux exemples comment la combinaison de toutes les techniques décrites jusqu'à présent permettent de démontrer des théorèmes simples de cette théorie naïve des ensembles. Nous reproduisons en fait dans le cadre de la Résolution certaines techniques utilisées dans le démonstrateur Imply [Ble71] [BB74].

Considérons le théorème $A \cup B =_{set} B \cup A$. Pour établir la validité de cette formule, on procède comme suit:

- on remplace systématiquement dans la formule à démontrer chaque definiendum par sa définition. La formule précédente devient

$$(A \cup B \subseteq B \cup A) \wedge (B \cup A \subseteq A \cup B)$$

puis

$$(\forall x\ (x \in A\ \vee\ x \in B) \rightarrow (x \in B\ \vee\ x \in A)) \wedge (\forall x\ (x \in B\ \vee\ x \in A) \rightarrow (x \in A\ \vee\ x \in B))$$

- On considère la négation de la formule ainsi obtenue et l'on favorise les études par cas. La formule correspondant au premier conjoint de la formule précédente est donc:

$$\neg(\forall x\ (x \in A\ \vee\ x \in B) \rightarrow (x \in B\ \vee\ x \in A))$$

- Afin de simplifier la présentation des transformations à suivre, nous supposerons que l'on applique une transformation de mise sous forme normale négative. Cette étape n'est pas nécessaire en pratique. La formule précédente devient alors:

$$(\exists x\ (x \in A\ \vee\ x \in B) \wedge x \notin B\ \wedge\ x \notin A))$$

- on procède à la skolemisation:

$$(x_0 \in A\ \vee\ x_0 \in B) \wedge x_0 \notin B\ \wedge\ x_0 \notin A))$$

- on applique les règles de simplification:

$$(Faux \vee\ Faux) \wedge x_0 \notin B\ \wedge\ x_0 \notin A))$$

La formule précédente se simplifie à $Faux$ et est donc réfutée.

- Dans le cas où les règles de simplification ne permettent pas de conclure, on applique une technique de renommage et l'on refute par Résolution l'ensemble de clauses ainsi obtenues.

Considérons le théorème plus compliqué $P(A \cap B) \subseteq P(A) \cap P(B)$. Ceci revient à montrer:

$$\forall x(\forall uu \in x \ \rightarrow (u \in A \wedge u \in B)) \rightarrow ((\forall uu \in x \ \rightarrow u \in A) \wedge (\forall uu \in x \ \rightarrow u \in B))$$

Nous allons montrer ce théorème en combinant des techniques de simplification avec des méthodes de transformation de spécification visant à introduire des sortes à la place de prédicats unaires (cf. paragraphe précédent). Si l'on skolemise la forme normale négative de la négation de cette formule, on obtient

$$(\forall uu \notin x_0 \ \vee (u \in A \wedge u \in B)) \wedge ((u_1 \in x_0 \ \wedge u_1 \notin A) \vee (u_2 \in x_0 \ \wedge u_2 \notin B))$$

où x_0, u_1, u_2 sont des constantes de Skolem. On peut renommer toutes les occurrences de $t \in x_0$ en $P_0(t)$ et introduire une sorte s_0 à la place du prédicat unaire P_0. On obtient alors la formule

$$(\forall u : s_0(u \in A \wedge u \in B)) \wedge (u_1 \notin A \vee u_2 \notin B)$$

où u_1, u_2 sont désormais des constantes de Skolem de sorte s_0. En utilisant la première règle de simplification, on montre l'insatisfaisabilité de cette formule.

Partie II

Techniques d'implémentation

Chapitre 7

Techniques d'implémentation pour des démonstrateurs utilisant la Résolution

Parmi les procédures de preuve pour la logique du premier ordre, les systèmes utilisant la Résolution et, si l'on traite aussi l'égalité, la Paramodulation se sont révélés être parmi les plus efficaces. Une part importante de notre travail a consisté à étudier diverses techniques d'implémentation de tels calculs et à mettre en œuvre un ensemble de primitives à partir desquelles un démonstrateur utilisant la Résolution et la Paramodulation peut être facilement programmé.

Cet ensemble de primitives a été choisi afin d'obtenir un compromis réaliste entre les exigences de généralité d'un travail de recherche et les besoins pratiques de construire des démonstrateurs offrant des temps de réponse raisonnables. L'ensemble logiciel a été conçu pour satisfaire aux contraintes suivantes:

- permettre de construire des démonstrateurs d'une efficacité suffisante pour que l'expérimentation sur des problèmes de difficulté moyenne soit praticable;

- permettre une manipulation facile des preuves obtenues à partir des démonstrateurs ainsi construits;
- être d'une modularité la plus grande possible.

Nous présentons rapidement quelques unes des solutions retenues dans le cadre d'ATINF [BdlTCC88].

7.1 Le partage de structures de Boyer et Moore

Le partage de structures de Boyer et Moore [BM72] permet de représenter de manière compacte les clauses générées par un démonstrateur par Résolution et Paramodulation.

L'utilisation de cette technique nécessite – du point de vue de l'implémentation – de considérer les clauses comme des séquences de littéraux [1] et de considérer les opérations de résolution, factorisation et paramodulation comme des opérations définies sur des séquences et qui donc tiennent compte de l'ordre des littéraux dans une clause.

L'idée de cette représentation est la suivante: soient C_1, C_2 deux clauses telles que le i-ème littéral de C_1 et le j-ième littéral de C_2 sont des littéraux de signe opposé dont les atomes sont unifiables par un unificateur le plus général σ. Alors la résolvante binaire de C_1 et C_2 sur les littéraux i, j est représentée sous la forme d'un quintuplet $< C_1, i, C_2, j, \sigma >$. De même, soit C_1 une clause telle que les i-ème et j-ème littéraux de C_1 $(i < j)$ sont des littéraux de même signe dont les atomes sont unifiables par un unificateur le plus général σ. Alors le facteur de C_1 sur le j-ème littéral est représenté par le triplet $< C_1, j, \sigma >$. La représentation d'une paramodulante peut être définie de manière similaire.[2] De plus, toutes les opérations usuelles dans un démonstrateur par Résolution et Paramodulation peuvent être appliquées à des clauses représentées sous ce format sans qu'il soit nécessaire de construire une autre représentation de ces clauses.

Dans le cadre d'ATINF,

[1] Ce point de vue est couramment adopté lorsqu'il s'agit d'implémenter des calculs utilisant la forme clausale et ne pose pas de difficultés particulières.

[2] Le traitement de l'égalité n'est pas considéré dans [BM72]. Nous avons développé un ensemble d'algorithmes permettant de prendre en compte la Paramodulation. Conformément à l'esprit du partage de structures, ces algorithmes sont conçus pour fonctionner sans recopie inutile. Cependant, une description de la solution retenue nous amènerait à présenter en détail certains aspects du partage de structures. Ce que nous ne voulons pas faire ici.

l'intérêt de cette représentation des clauses déduites réside dans le fait qu'elle donne l'historique des déductions appliquées: elle est donc aussi une représentation de la *déduction* de ces clauses.

Ainsi, définir un ensemble de primitives pour la manipulation de formules sous forme clausale dans cette représentation, c'est définir des primitives pour la manipulation de preuves.

7.2 Inconvénients du partage de structures

L'inconvénient majeur de cette représentation tient en ce que, étant donnée une résolvante R, on ne dispose pas de la séquence des littéraux de R mais on dispose de la séquence d'inférences ayant produit la clause R. Aussi, certaines opérations élémentaires sur les clauses doivent calculer la "valeur" des littéraux de R à partir de la déduction de R. Il en résulte une augmentation du temps nécessaire pour effectuer ces opérations; cette augmentation est évidemment d'autant plus grande que la taille de la déduction de R est grande.

Or, de plus en plus, les ordinateurs disposent d'une grande quantité de mémoire et la nécessité d'une représentation compacte des clauses se fait moins sentir. Aussi, la technique de partage de structures a-t-elle été abandonnée peu à peu au profit de techniques réalisant une *recopie de structures*,[3] voire même construisant explicitement les clauses déduites.

Bien entendu, si l'on désire garder le partage de structures afin de conserver la représentation de la déduction des clauses déduites, il est possible d'adjoindre à la représentation compacte d'une clause une représentation construite.

En fait, nous pensons que disposer d'une représentation construite d'une résolvante n'est pas forcément une bonne solution pour obtenir un système de démonstration automatique efficace.

Le point crucial de l'efficacité d'un démonstrateur réside, étant donnée une clause C, dans la possibilité de pouvoir sélectionner rapidement quelles sont les clauses susceptibles d'être résolues avec C, d'être subsumées par C ou de subsumer C.

Ce point a été souligné dans [LMO82], [BLMO86]:

[3] Ceci est vrai aussi pour les démonstrateurs utilisant les acquis de la technologie d'implémentation des interpréteurs PROLOG (voir [Sti86a]). La plupart des interpréteurs PROLOG utilisent maintenant une technique dite de "structure copying" [Bru82].

[LMO82]:
... While some have claimed that the speed of the unification algorithm is the principal determinant of the overall speed of a theorem prover because of the frequency of its invocation, our experience indicates that this may be an oversimplification. By using techniques which efficiently select objects with high probability of unifying with a given object, one can ensure that most invocations will provide successful unification. In this environment a poor implementation could still absorb as much as fifty percent of overall theorem-prover execution time.

[BLMO86]:
... The database component of our previous systems has always been a major factor in their effectiveness... We reiterate the necessity of separating the database and deduction components in automated theorem proving.

Nous présentons une structure de données qui permet de stocker de manière compacte un multi-ensemble S de termes. Cette structure de données permet de trouver rapidement et de manière incrémentale tous les termes de S qui s'unifient, filtrent ou sont filtrés par un terme t. Cette représentation des termes permet de répondre efficacement à de nombreux problèmes clés en démonstration automatique de théorèmes:

- production de toutes les résolvantes à partir d'une clause;
- recherche de clauses subsumant une clause C;
- recherche de clauses subsumées par une clause C;
- calcul de paires critiques;
- simplification par des règles de réécriture ...

Nous avons appelé cette structure de données un *arbre de sélection.* Les arbres de sélection sont bien adaptés à la gestion de multi-ensembles de termes qui évoluent de manière dynamique. Dans le cas de problèmes plus spécifiques (multi-ensembles statiques, restriction au filtrage, ...), des solutions plus complexes permettant des gains en temps de recherche et en espace mémoire sont certainement possibles.

7.3 Arbres de sélection

Un arbre de sélection est une structures de données permettant de stocker de manière compacte un multi-ensemble S de termes en réalisant un partage de structure entre les termes de S qui, quand représentés sous forme préfixée, admettent un préfixe commun.

Soit un ensemble V de variables $\{v_1, v_2, \ldots\}$ disjoint de l'ensemble de variables figurant dans les termes du multi-ensemble de termes à représenter. Etant donné un terme t de ce multi-ensemble, nous appelerons *terme linéaire associé à* t le terme t^l dont la représentation préfixée est obtenue en substituant dans la représentation préfixée de t la variable v_i à la i-ème occurrence d'une variable de t.

Par exemple, si $t = f(g(x,y),x)$ alors $t^l = f(g(v_1,v_2),v_3)$;
si $t = f(g(x,y),z)$ alors $t^l = f(g(v_1,v_2),v_3)$.

Donc deux termes identiques au nom de leurs variables près ont même terme linéaire associé.

On appelera *système d'équations associé* à t l'ensemble S_t de paires (v_i, x) correspondant au remplacement de x par v_i dans l'écriture préfixée de t^l à partir de l'écriture de t. Etant donnés les termes t, t^l et le système $S_t = \{(v_1,x_1),\ldots,(v_n,x_n)\}$, on a:

$$t = t^l(v_1/x_1, \ldots v_n/x_n)$$

Par exemple, les systèmes d'équations associés aux deux termes de l'exemple précédent sont respectivement:

$$S_t = \{(v_1,x),(v_2,y),(v_3,x)\}$$

$$S_t = \{(v_1,x),(v_2,y),(v_3,z)\}$$

On représentera un multi-ensemble de termes construits sur un ensemble de symboles fonctionnels F par un arbre dont les feuilles contiennent les termes de S et dont les nœuds intérieurs seront étiquetés par des symboles de $F \cup V \cup \{\Lambda\}$. La racine de cet arbre est l'*unique* nœud intérieur étiqueté par Λ. Tout nœud intérieur admet un ensemble non vide de successeurs, représentés par une liste de nœuds. Cette liste comprend

- soit des nœuds intérieurs dont les étiquettes sont toutes différentes;

- soit un unique nœud feuille .

Chaque nœud feuille contient un multi-ensemble de paires (t, S_t) où t est un terme du multi-ensemble de termes que l'on veut stocker et S_t est le système d'équations associé à t. De plus, pour toute paire $(t_1, S_{t_1}), (t_2, S_{t_2})$ d'un nœud feuille, on impose que $t_1^l = t_2^l$. C'est pourquoi on dira aussi que t_1^l est *le* terme linéaire associé au nœud feuille contenant (t_1, S_{t_1}).

Appelons *chemin* une séquence de nœuds $< N_0, \ldots, N_{p+1} >$ telle que:

- N_0 est le nœud racine;
- N_i est une nœud intérieur, successeur immédiat du nœud N_{i-1}, $1 \leq i \leq p$;
- N_{p+1} est une nœud feuille successeur immédiat du nœud N_p.

Etant donné un tel chemin $< N_0, \ldots, N_{p+1} >$, désignons par e_i l'étiquette du nœud N_i, $1 \leq i \leq p$. On impose alors, que pour tout terme t figurant dans le nœud feuille N_{p+1}, la séquence de caractères $e_1 \ldots e_p$ soit identique à l'écriture préfixée de t^l.

Exemple: considérons la famille de termes suivante:

$$s = \{g(f(a), b),\ g(f(b), a),\ g(a, b),\ g(a, x),\ g(a, y),\ f(a)\}$$

L'arbre associé à cet ensemble est similaire à celui dessiné figure 7.1:

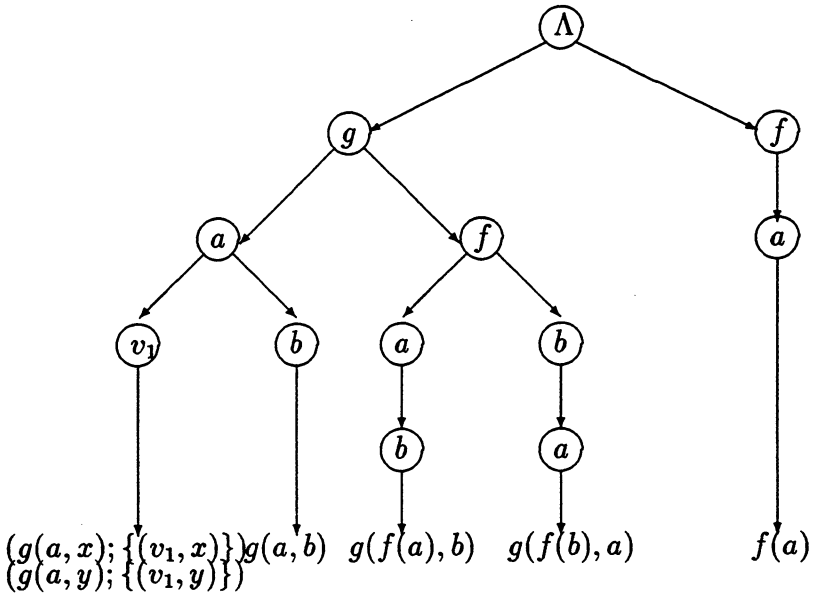
On remarque que les termes g(a,x) et g(a, y) sont stockés dans le même nœud feuille car ces deux termes sont identiques au nom de leurs variables près. Plus généralement, étant donnés deux termes t_1, t_2, si les séquences de caractères correspondant à l'écriture préfixée de t_1^l et t_2^l ont un même préfixe, alors t_1 et t_2 sont stockés dans des nœuds feuilles N_{p+1}^1, N_{q+1}^2 tels que les chemins $< N_0^1, \ldots, N_{p+1}^1 >$ et $< N_0^2, \ldots, N_{q+1}^2 >$ comportent une partie commune correspondant *au plus grand préfixe commun* de t_1^l et t_2^l.

7.4 Algorithmes de sélection

Etant donnée une telle structure de données, il est facile de trouver tous les termes s'unifiant avec un terme t_0. On procède en deux temps:

- par un parcours de l'arbre de sélection de la racine vers les feuilles, on peut déterminer quelles sont les feuilles dont le terme linéaire associé est unifiable

Figure 7.1: Un exemple d'arbre de sélection



avec t_0: ce parcours s'apparente à un algorithme d'unification sans occur-check [Hue76] entre termes représentés sous forme préfixée[4]. On associe alors à chaque feuille dont le terme linéaire associé est t^l une substitution σ_0 telle que $\sigma_0(t^l) = \sigma_0(t_0)$.

- Puis, pour chacune des paires (t, S_t) figurant dans un nœud feuille, on cherche en utilisant S_t à modifier la substitution σ_0 en une substitution σ telle que $\sigma(t^l) = \sigma(t_0)$.

Nous présentons sous forme d'un programme PROLOG l'algorithme permettant de réaliser la première étape, le traitement de la seconde étant sans difficulté. Nous adoptons les conventions d'écriture couramment employées dans la syntaxe des interpréteurs PROLOG: les variables sont représentées par des identificateurs commençant par une majuscule, [X | L] désigne une liste dont le premier élément est X et le reste est L, ...

Supposons que l'on dispose des primitives suivantes:

- `suivants(Nœud, Liste_de_nœuds)`
  réussit si et seulement si `Liste_de_nœuds` contient les successeurs du nœud `Nœud`;

- `nœud_etiquete(Liste_de_nœuds, Nœud, Symbole)`
  réussit si et seulement si `Liste_de_nœuds` contient un nœud `Nœud` étiqueté par le symbole `Symbole`;

- `premier_sous_terme(L1, L2, L3)`
  réussit si la liste `L1` est le résultat de la concaténation des deux listes `L2`, `L3` et `L2` contient une liste de symboles correspondant à l'écriture préfixée d'un terme;

- `premier_sous_arbre(N1, N2, L)`
  réussit si le nœud `N1` est la racine d'un arbre contenant un sous-arbre de racine `N2` telle que `L` est la liste des symboles étiquettant tous les nœuds du chemin partant de `N1` exclus au nœud `N2` inclus. De plus, `L` contient une liste de symboles correspondant à l'écriture préfixée d'un terme;

- `est_liee((Var, Terme), Subst)`
  réussit si la variable `Var` est liée au terme `Terme` dans la substitution `Subst`,

[4] Un algorithme d'unification de deux termes dont les ensembles de variables sont disjoints peut omettre le test d'occur-check si l'un des deux termes à unifier est linéaire.

représentée par une liste de paires [(Var1, Terme1), ..., (VarN, TermeN)] telle que VarI $\neq$ VarJ si $i \neq j$.

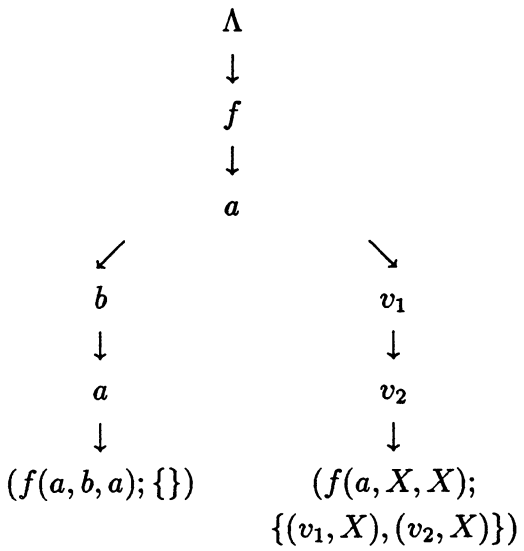
- `non_liee(Var, Subst)`
  réussit si la variable `Var` n'est liée à aucun terme dans la substitution `Subst`.
- `var(Symbole)`
  réusit si `Symbole` désigne un symbole de variable;
- `fonct(Symbole)`
  réusit si `Symbole` désigne un symbole de fonction;

On a alors le programme suivant:

```
unif([], Nœud1, Feuille, Subst) :-
      suivants(Nœud1, [Feuille]).
unif([F | L], Nœud1, Nœud3, Subst) :-
      fonct(F) &
      suivants(Nœud1, Liste_de_nœuds) &
      nœud_etiquete(Liste_de_nœuds, Nœud2, F) &
      unif(L, Nœud2, Nœud3, Subst).

unif([F | L], Nœud1, Nœud3, Subst) :-
      fonct(F) &
      suivants(Nœud1, Liste_de_nœuds) &
      nœud_etiquete(Liste_de_nœuds, Nœud2, V) &
      premier_sous_terme([F | L], Terme, Reste) &
      unif(Reste, Nœud2, Nœud3, [(V, Terme) | Subst]).

unif([V | L], Nœud1, Nœud3, Subst) :-
      var(V) &
      est_liee((V, Terme), Subst) &
      unif(Terme, Nœud1, Nœud2, Subst) &
      unif(L, Nœud2, Nœud3, Subst).

unif([V | L], Nœud1, Nœud3, Subst) :-
      var(V) &
      non_liee(V, Subst) &
```

```
premier_sous_arbre(Nœud1, Nœud2, Terme) &
unif(L, Nœud2, Nœud3, [(V, Terme) | Subst]).
```

Considérons l'arbre de sélection suivant:



que nous représenterons de manière abrégée en omettant les flèches et en écrivant les nœuds feuille t au lieu de (t, S_t).

Etant donnée la requète `:- unif([f,V,b,V], Λ ..., X,[])`, on obtient alors la trace des appels décrit à la figure 7.2. On remarque que la deuxième solution produit une substitution qui ne peut être étendue en un unificateur de $f(a, X, X)$ et de $f(V, b, V)$.

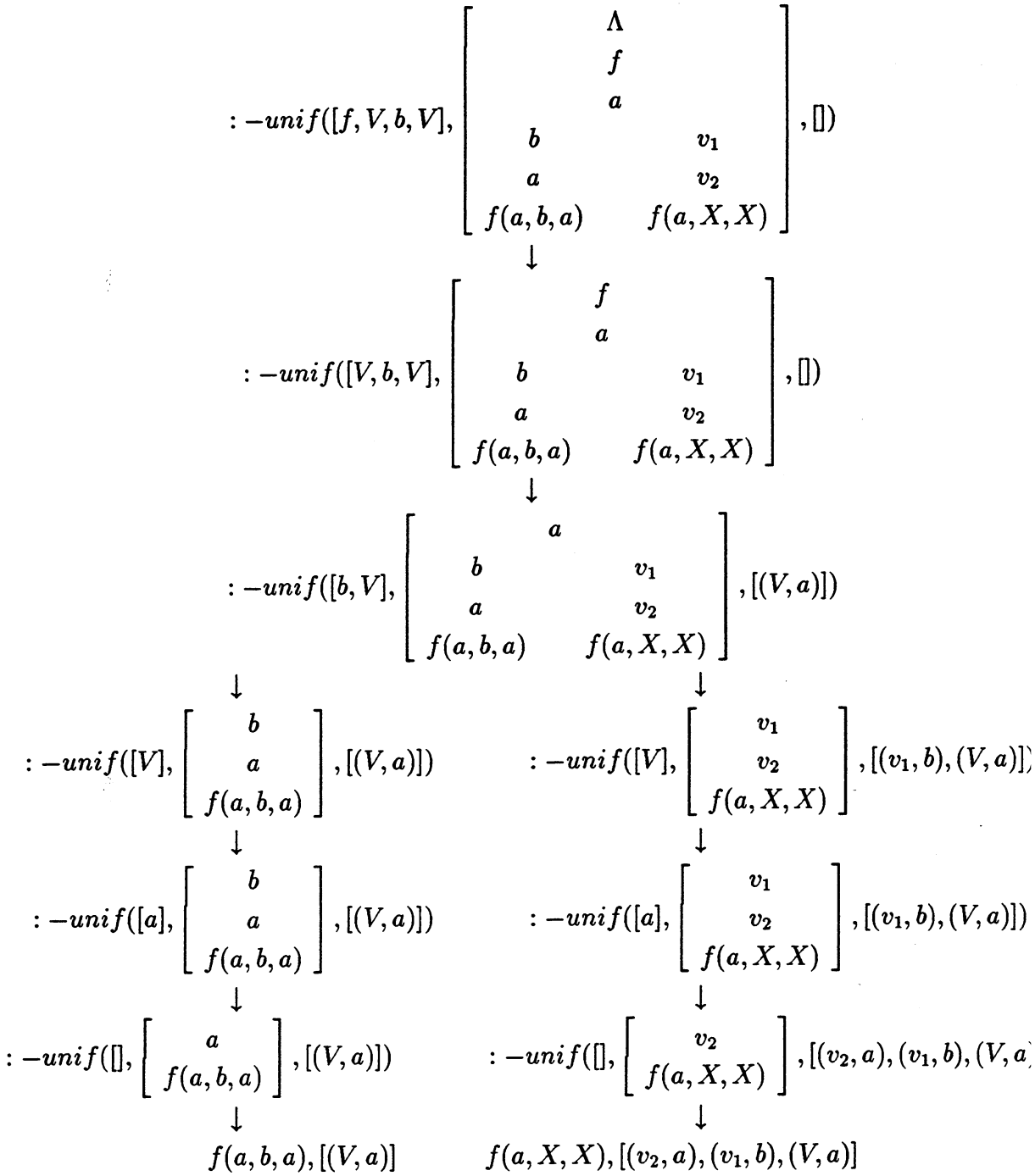
On peut définir de manière similaire un algorithme qui, étant donné un terme t et un arbre de sélection A, permet d'obtenir:

- tous les termes des nœuds feuilles de A contenant des termes qui sont des instances de t;
- tous les termes des nœuds feuilles de A contenant des termes dont t est une instance.

7.5 Comparaisons avec d'autres méthodes

Des structures de données similaires ont été décrites dans [McC88], [Gre86]. Il s'agit de structures de données arborescentes dont les feuilles sont utilisées pour stocker

Figure 7.2: Exemple de trace des appels pour le programme UNIF



des termes. Cependant, la séquence des étiquettes d'un chemin de la racine à une feuille correspont à l'écriture préfixée *du terme* à stocker et non pas à l'écriture préfixée *du terme linéaire* associé au terme à stocker. Nous appelerons de telles structures de données des *arbres de discrimination.*

La différence - a priori mineure - entre arbres de sélection et arbres de discrimination n'est pas sans conséquences.

La place mémoire nécessaire pour représenter un ensemble de termes par un arbre de discrimination est plus importante: en effet, le partage de structures réalisé dans l'arbre de discrimination est toujours moins important que celui réalisé avec notre méthode. Par exemple, les termes $f(x, y, a, a)$ et $f(x, x, a, b)$ n'admettent que "fx" comme préfixe commun alors que les termes linéaires associés $f(v_1, v_2, a, a)$ et $f(v_1, v_2, a, b)$ admettent "fv_1v_2a" comme préfixe commun. En particulier, à chaque feuille d'un arbre de discrimination est associé un unique terme à stocker alors que notre représentation permet de stocker dans le même nœud feuille plusieurs termes ayant même terme linéaire associé.

Si l'on veut trouver tous les termes stockés dans un arbre de discrimination qui s'unifient avec un terme donné, il est nécessaire d'utiliser un algorithme de parcours de l'arbre qui intègre le test d'occur-check, car la séquence des étiquettes d'un chemin de la racine à une feuille correspont à l'écriture préfixée d'un terme non linéaire. L'algorithme de parcours de l'arbre devient alors très compliqué à mettre en œuvre et s'avère peu interessant en pratique selon [McC88]. Aussi, dans cet article, ces structures de données ne sont utilisées que pour des opérations de recherche de termes utilisant le *filtrage*, à savoir les opérations de subsomption unitaire et de simplification par un système de réécriture de termes. Dans [Gre86], les arbres de discrimination sont aussi utilisés pour effectuer le test de subsomption entre clauses non unitaires et pour trouver tous les termes qui s'unifient avec un terme t donné.

Enfin, remarquons qu'en utilisant les arbres de sélection, nous obtenons directement la solution de chaque problème d'unification ou de filtrage sous la forme d'un système d'équations en forme résolue (au sens de [JK90]): ceci correspond à l'information qui doit être stockée dans la représentation d'une résolvante dans le partage de structures de Boyer et Moore. Il n'y a donc plus besoin d'utiliser l'algorithme d'unification adapté au partage de structures: cet algorithme, proposé dans [BM72], a une complexité exponentielle dans le pire des cas [MM76]. Par contre, il semble que la représentation des clauses utilisées dans [Gre86] nécessite un recalcul des substitutions pour chaque terme sélectionné.

7.6 Applications

Il existe dans ATINF plusieurs arbres de sélection; chaque feuiile de ces arbres contient un ensemble de triplets de la forme:

$$< C, i, S >$$

qui indique que l'atome du i-ème littéral de la clause C peut être obtenu à partir du terme linéaire correspondant au nœud feuille considéré par application de la substitution définie par le système S.

Les différents arbres de sélection sont les suivants:

- deux arbres de sélection contenant uniquement les littéraux provenant de clauses unitaires (un pour les littéraux positifs, l'autre pour les littéraux négatifs). Ceci permet de réaliser très efficacement les opérations suivantes:
  - subsomption par une clause unitaire
  - simplification clausale: cette opération consiste, étant donnée une clause de la forme $C \vee L$ à chercher une clause unitaire C' contenant un litteral de signe opposé à L et dont L est une instance. La résolvante binaire de ces deux clauses est alors la clause C qui subsumme la clause $C \vee L$: cette dernière peut être détruite (en fait, marquée comme inutilisable).
  - simplification équationnelle: si l'on dispose d'un ensemble de règles de réécriture, c'est-à-dire un ensemble d'équations orientées de la forme $l \rightarrow r$, il s'agit de réécrire toute occurrence d'un terme de la forme $\sigma(l)$ en $\sigma(r)$.
  - calcul des paires critiques
- deux arbres de sélection contenant les littéraux sélectionnés des clauses non unitaires. Lorsqu'une clause contient plusieurs littéraux, on cherche à restreindre le nombre d'applications d'une règle d'inférence en limitant le nombre de littéraux de cette clause susceptibles d'être utilisés dans une application de cette règle d'inférence. Ces littéraux sont appelés littéraux sélectionnés. Il existe un arbre de sélection pour les littéraux positifs et un autre pour les littéraux négatifs.
- deux arbres de sélection contenant tous les littéraux de toutes les clauses non unitaires de la base de clauses. Il existe un arbre de sélection pour les littéraux positifs et un autre pour les littéraux négatifs.

Cette dernière structure n'est construite que si l'on désire tester la θ-subsomption entre clauses non unitaires. Etant donnée une clause $C = L_1 \vee \ldots \vee L_n$, on cherche pour chacun des littéraux L_i quelles sont les clauses C' contenant un littéral L tel que $L_i = \sigma(L)$. Si C' ne contient pas plus de littéraux que C et si L est le j-ème littéral de C' alors le triplet $< C', j, \sigma >$ est mémorisé. Pour chaque clause C' contenant p littéraux et telle que tout littéral L_j de C' est plus géneral qu'au moins un littéral de C, on dispose d'un ensemble de tels triplets $< C', j, \sigma_j^i > 1 \leq j \leq p, 1 \leq i \leq q_j$ si q_j désigne le nombre de littéraux de C qui sont des instances de L_j. Etant donnée une suite de substitutions $\sigma_1^{i_1} \ldots \sigma_p^{i_p}$, on essaie de composer ces substitutions. En cas de succès, la clause C est subsumée par C'.

Ce test est très couteux mais nous ne connaissons pas de technique permettant de sélectionner efficacement parmi N clauses une clause subsumant une clause C donnée.[5] En pratique, il est souvent nécessaire de considérer des tests plus simples qui peuvent être mis en œuvre rapidement à l'aide des arbres de décision. Un première solution consiste à ne considérer que les clauses C' dont le j-ème littéral L_j est plus général qu'un littéral L_i de C et tels que $L_i^l = L_j^l$. Ceci limite le parcours de l'arbre de décision à un parcours d'un unique chemin de la racine vers la feuille correpondant à L_i^l. Nous appelons ce test de θ-subsomption *test délimination des variantes* (ou "variant-checking"). Remarquons qu'il est suffisant pour tester la subsomption entre clauses closes. Une deuxième possibilité consiste à ne conserver que les clauses C' telle que le i-ème littéral de C est une instance du i-ème littéral de C'. Cette dernière retriction est particulièrement bien adaptée aux procédures de preuves du type Model Elimination ou SL-resolution du fait de la forme des déductions qui sont générées.

Les expérimentations faites jusqu'ici tendent à montrer que, dans les cas où il est nécessaire d'effectuer la subsomption entre clauses non-unitaires, le test d'élimination des variantes est le plus avantageux.

7.7 Un exemple test: le problème dit du "Steamroller"

Nous illustrons l'intérêt qu'il y a à utiliser les arbres de sélection en comparant les performances de plusieurs démonstrateurs dans la recherche d'une solution à un problème à forte combinatoire: le problème du "Steamroller". Pour résoudre ce

[5] On trouve dans [GL85] une très bonne étude de la compléxité du test de subsomption entre *deux* clauses ainsi que la description d'un algorithme sophistiqué pour effectuer ce test. Mais utiliser N fois cette algorithme est plus couteux que d'utiliser le procédé décrit ci-dessus.

problème formulé en logique du premier ordre sans sorte, il est nécessaire d'utiliser de manière intensive des procédures de subsomption et de simplification clausale.

Le problème du "Steamroller" est le suivant:

> Les loups, les renards, les oiseaux, les escargots et les chenilles sont des animaux et il existe un représentant de chaque espèce. Il existe aussi des graines et les graines sont des plantes. Chaque animal aime manger soit toutes les plantes soit tous les animaux qui sont plus petits que lui et qui aiment manger des plantes. Les chenilles et les escargots sont plus petits que les oiseaux, qui sont eux même plus petits que les renards, ces derniers étant plus petits que les loups. Les loups ne mangent ni graine ni renard tandis que les oiseaux aiment manger les chenilles mais pas les escargots. Les chenilles et les escargots aiment manger des plantes. Donc il existe un animal qui aime manger un animal mangeur de graines.

Cette description en langue naturelle a donné naissance à plusieurs formulations en logique du premier ordre sans sorte. Nous utilisons la formulation désignée par (**) dans [Sti86b] .

A chaque cycle, notre démonstrateur sélectionne parmi un ensemble de résolvantes toutes les clauses de poids minimal en fonction d'une fonction heuristique donnée (ici, le nombre de littéraux de la clause a été utilisé). Ces clauses sont ajoutées à la base de clauses (c'est-à-dire aux différents arbres de sélection); les autres clauses sont mises en attente. Puis, toutes les résolvantes obtenues par résolution binaire et/ou par paramodulation à partir des clauses retenues sont calculées. Chaque résolvante est ensuite traitée comme suit:

- Si l'on dispose d'un système de réécriture, tous les littéraux de la clause sont normalisés.
- Puis, les clauses étant considérées comme des séquences de littéraux, on recherche s'il existe deux occurrences distinctes de littéraux ayant des atomes identiques:
  - s'il existe deux littéraux de signe contraire, alors la clause est une tautologie et elle est éliminée;
  - s'il existe deux littéraux de même signe alors l'un des deux littéraux est éliminé.

- On recherche ensuite si la clause ainsi obtenue est subsumée par une clause générée précédemment. On considére d'abord les clauses unitaires puis les clauses non unitaires.
- On applique systématiquement la simplification clausale.
- On recherche si la clause ainsi obtenue subsume une clause générée précédemment.
- On calcule les littéraux sélectionnés de la clause.
- Enfin, on calcule les facteurs de cette clause; chacun des facteurs est ensuite traité comme précédemment.

La stratégie employée ordonne les littéraux des clauses suivant le principe suivant:

- les littéraux négatifs sont sélectionnés avant les littéraux positifs;
- parmi les littéraux négatifs, les littéraux dont le symbole de prédicat est maximal par rapport à un ordre sur les symboles de prédicats sont sélectionnés en priorité;
- parmi ceux ci, uniquement le littéral apparaissant en première position si la clause est considérée comme une séquence de littéraux est sélectionné.
- si une clause ne contient que des littéraux positifs, tous les littéraux dont le symbole de prédicat est maximal par rapport à un ordre sur les symboles de prédicats sont sélectionnés.

Cette stratégie est donc un raffinement de la P1-Résolution ordonnée [HK69], [Rus87], une restriction de la Résolution qui impose que dans toute application de la règle de Résolution un parent ne contienne que des littéraux positifs et l'autre soit ordonné suivant un ordre sur les symboles de prédicats.

Les principaux résultats sont donnés par les tableaux suivants: on considère la stratégie évoquée plus haut soit avec subsomption unitaire et simplification clausale (essai I) soit avec élimination des variantes et simplification clausale (essai II). Pour la stratégie considérée, le nombre de pas de résolution est du même ordre de grandeur que le nombre d'unification réussis. Les expérimentations ont été réalisées sur un SUN3-60 (8 MO); le démonstrateur par Résolution d'ATINF est écrit en COMMON LISP.

Figure 7.3: Steamroller: Résultats généraux

	Temps (en s)	nb cycles	nb clauses générées	nb res.
I	16,82	21	1084	1147
II	16,14	21	298	334

Figure 7.4: Steamroller: Résultats cycle par cycle pour la méthode I

cycle	1	2	3	4	5	6	7	8	9	10	
clauses retenues	0	6	10	10	9	13	33	50	1	5	
clauses en attente	27	21	11	11	2	2	2	2	1	1	
cycle	11	12	13	14	15	16	17	18	19	20	21
clauses retenues	1	17	1	5	2	1	33	1	28	32	28
clauses en attente	18	1	0	0	22	22	22	21	22	22	22

Figure 7.5: Steamroller: Résultats cycle par cycle pour la méthode II

cycle	1	2	3	4	5	6	7	8	9	10	
clauses retenues	0	6	10	10	9	13	33	50	1	5	
clauses en attente	27	21	11	11	2	2	2	2	1	1	
cycle	11	12	13	14	15	16	17	18	19	20	21
clauses retenues	1	17	1	5	2	1	33	1	1	32	1
clauses en attente	18	1	0	0	22	19	19	18	14	14	10

Les preuves trouvées avec ou sans test d'élimination des variantes sont identiques. De plus, ces deux méthodes génèrent *presque exactement les mêmes clauses* des cycles 1 à 18 inclus : 261 clauses sont produites et 270 pas de résolution sont effectués. Si le test d'élimination des variantes n'est pas effectué, alors ces opérations sont effectuées en 7.0 secondes; Si le test d'élimination des variantes est effectué, alors ces opérations sont effectuées en 14.4 secondes.

A l'étape 19, la procédure I génére 28 clauses qui sont toutes des variantes de la clause:

$$\neg Plant(X) \vee Eat(a\_fox, X)$$

Cette clause n'étant pas une clause unitaire, les 28 variantes ne peuvent être éliminées par subsomption unitaire. Or cette clause produit 36 résolvantes unitaires. Donc la procédure I génère au pas suivant $36 \times 28 = 1008$ résolvantes. Ces résolvantes étant des clauses unitaires, les clauses identiques sont éliminées par subsomption unitaire. La production de ces résolvantes et le test de subsomption s'effectuent en environ 8.4 secondes. La réfutation est ensuite trouvée en environ 1.2 secondes. Si l'on considère la production de 1000 clauses unitaires et le test de subsomption unitaires de ces 1000 clauses comme représentant 2000 sélections réussies dans un arbre de discrimination, on obtient alors un taux d'environ 240 sélections réussies par seconde dans un arbre de sélection avec des termes contenant en moyenne entre 3 et 7 symboles.

Dans le cas de la procédure II utilisant le test d'élimination des variantes, le passage du cycle 19 au cycle 20 se fait en 1.2 secondes; la réfutation est obtenue à partir de là en 0.460 secondes.

Nous mentionons à titre indicatif les résulats obtenus par d'autres démonstrateurs sur des version identiques du problème du Steamroller ([Sti86b]).

CG est une implementation de la méthode des graphes de connexion ([Kow75], [Sti82]) réalisée sur une machine LISP Symbolics 3600. Cette implémentation utilise la subsomption unitaire, la simplification clausale et donne la préférence aux pas de résolution dont au moins un parent est une clause unitaire. De plus, une stratégie (non précisée) d'ordonnancement des atomes est utilisée. Les suffixes SOS et TR indiquent l'emploi de la stratégie du support [WRC67] et/ou de la T-résolution [Sti85].

ITP [LMO84] et TP0 sont des démonstrateurs écrits en PASCAL à partir de LMA [LMO82]. Ces démonstrateurs utilisent soit la "Unit Resulting"-Résolution, une forme d'hyper-résolution qui cherche à produire des clauses unitaires, soit la

"Qualified Unit Resulting"-Résolution, une adaptation à la stratégie précédente d'un rafinement de l'hyper-résolution [Win85]. Les expérimentations ont été réalisée sur un VAX 11/780.

MKRP [BES+81] est une implementation très sophistiquée de la méthode des graphes de connexion. MKRP est écrit en INTERLISP et a été utilisé sur une machine Siemens 1660.

Enfin, le démonstrateur de S. Greenbaum [Gre86] est un démonstrateur écrit en Franz-LISP. Il utilise la même stratégie que celle que nous avons employée et il dispose des mêmes outils de simplification ou de subsomption. Rappelons qu'il utilise une structure de données similaire à la notre pour sélectionner les clauses en vue d'un pas de résolution ou d'un test de subsumption. Cependant, il ne tire pas profit du fait qu'il est possible de calculer un unificateur (ou une substitution de filtrage) de deux termes lors de la sélection comme nous l'avons fait. De plus, ce démonstrateur utilise une technique de recopie de structures, le partage de structure de Boyer et Moore ayant été jugé trop inefficace. La représentation des clauses a été optimisée en fonction de la stratégie de Résolution décrite ci-dessus, cette stratégie étant la seule disponible dans le démonstrateur de Greenbaum.

Ces résultats, établis sur un problème réputé difficile, montre l'intérêt qu'il y a à utiliser les arbres de sélection et à calculer les substitutions lors du parcours des arbres de sélection. Ils montrent aussi que l'utilisation du partage de structure de Boyer et Moore n'est pas pénalisante en temps de calcul si l'on utilise les arbres de sélection.

Enfin, nous donnons des résulats toujours tirés de [Sti86b] du traitement avec sortes ordonnées de ce problème. (Le démonstrateur tp0 simule la Résolution avec sortes ordonnées mais ne met pas en œuvre le calcul de C. Walther [Wal83].)

La description du traitement de la Résolution avec sortes ordonnées dans ATINF fait l'objet du chapitre suivant.

Figure 7.6: Steamroller: Exemples de performances de systèmes

Système	nb clauses	unifications	Temps (en sec.)
CG	135	1340	25
CG-SOS	?	?	?
CG-TR	38	764	13
CG-SOS-TR	151	5840	124
ITP/UR	127	6371	340
ITP/QUR	131	3053	155
tp0/UR	131	26190	660
tp0/QUR	92	2663	120
MKRP	60	?	262
Greenbaum/Locking	427	1324	48.5

Figure 7.7: Steamroller avec sortes ordonnées: Exemples de performance

Système	nb clauses	unifications	Temps (en sec.)
CG	16	76	4
CG-SOS	13	137	6
tp0/UR	8	8	59
MKRP	10	48	3

Figure 7.8: Steamroller avec sortes ordonnées: Exemples de performance d'ATINF

Système	nb clauses	nb res.	Temps (en sec.)
ATINF/I	8	14	0.740
ATINF/II	8	14	0.920

Chapitre 8

Techniques d'implémentation efficace de la Résolution avec sortes ordonnées

8.1 Introduction

Bien que n'ajoutant rien au pouvoir d'expression des logiques sans sortes, les logiques avec sortes ordonnées permettent d'exprimer de manière concise, naturelle et élégante des informations de nature taxonomique. A chaque classe d'individus est associé un symbole appelé *sorte*. Les relations d'inclusion entre classes d'individus se traduisent par un ordre partiel sur les symboles de sorte correspondants. De plus, pour chaque symbole de fonction (resp. de prédicat), les sortes permettent de préciser les domaines sur lesquels est bien définie la fonction (resp. la relation) représentée par ce symbole.

Les domaines d'application de ces logiques sont nombreux: programmation logique, spécifications algébriques, mécanisation du raisonnement, ...Dans ce

dernier cas, l'utilisation de mécanismes d'inférence qui tiennent compte des informations sémantiques reflétées par les sortes permet d'améliorer le comportement des procédures de preuve, tant en ce qui concerne la maitrise de l'espace de recherche que la longueur ou le caractère naturel des preuves ainsi trouvées.

Nous nous intéresserons dans ce chapitre aux calculs des prédicats utilisant la Résolution et la Paramodulation tels qu'ils ont été présentés dans [Wal83], [SS85a]. Pour l'essentiel, ces calculs ne diffèrent de la Résolution et la Paramodulation usuelles (c'est-à-dire sans sortes) que par l'utilisation d'un algorithme d'unification spécialisé qui tient compte des sortes. Ces calculs apparaissent donc comme des extensions naturelles de la Résolution et la Paramodulation sans sortes. Ceci n'est pas le cas pour d'autres calculs, tel celui présenté dans [Coh87], qui nécessitent l'utilisation de règles d'inférence supplémentaires pour assurer leur complétude.

Le but de ce chapitre est d'étudier comment mettre en œuvre de façon efficace les calculs de C. Walther et M. Schmidt-Schauss. En particulier, une des caractéristiques de ces calculs est que tout problème d'unification admet un ensemble minimal et complet de solutions. Mais cet ensemble n'est généralement pas un singleton et ceci n'est pas sans conséquences en regard des opérations usuellement mise en œuvre dans un démonstrateur automatique de théorèmes utilisant la Résolution et la Paramodulation. Nous étudierons quelles structures de données et quels algorithmes utiliser pour mettre en œuvre des règles de transformation dérivées de celles décrites dans [SS87] [Kir88] [JK90] permettant de résoudre un problème d'unification avec sortes ordonnées et nous considérerons une représentation possible des clauses dont les variables sont sortées.

8.2 Préliminaires

Les notations utilisées ici sont principalement celles de [Kir88].

On rappelle que, étant donné un ensemble S partiellement ordonné par la relation $\leq$, $Max(S) = \{x \in S / \forall x' \in S,\ x \leq x' \rightarrow x = x'\}$. Par abus de notation, on notera aussi par $\leq$ la relation d'ordre entre séquences d'éléments de S définie par: $[x_1 \ldots x_n] \leq [y_1 \ldots y_n]$ si et seulement si $x_i \leq y_i (1 \leq i \leq n)$

Soit S un ensemble de symboles appelés *sortes* et dont les éléments seront dénotés par $s_1, s_2, \ldots$ Cet ensemble est partiellement ordonné par la relation $\leq$. Cette relation peut être définie comme le plus petit ordre partiel contenant un ensemble SD de *déclarations de sous-sortes* de la forme $s_1 \gg s_2$. Appelons F un ensemble de sym-

boles fonctionnels d'arité fixe dont les éléments seront dénotés par $f, g, h, a, \ldots$. Le *profile* d'un symbole fonctionnel f d'arité n est un ensemble non vide de *déclarations de fonctions* de la forme $f : s_1 \times \ldots \times s_n \mapsto s$. De même, soit P un ensemble de symboles de prédicat d'arité fixe et dont les éléments seront dénotés par $p, q, r, \ldots$ Le *domaine* d'un symbole de prédicat p d'arité n est un ensemble non vide de *déclarations de prédicat* de la forme $p : s_1 \times \ldots \times s_n$. Soit OD l'ensemble des déclarations de fonction et de prédicat ainsi définies. On appelera *signature avec sortes ordonnées* le quadruplet $\Sigma = (S, \leq, SD, OD)$. Cette signature est dite *finie* si chacun de ces composants est un ensemble fini.

Soit V un ensemble de variables dont les éléments seront dénotés par les symboles $x, y, z, u, v, w, \ldots$. Etant donné un sous-ensemble X de V, on appelle *assignation de sorte aux variables de* X toute application δ définie de X dans S. Par la suite, il sera souvent utile de rappeler qu'une assignation de sorte δ associe à une variable x la sorte s en écrivant $x : s$ ou x^s.

Selon les notations habituelles, on notera $T(F, X)$ l'ensemble des termes construits à partir de l'ensemble de symboles fonctionnels F et de l'ensemble de variables X. $T(F, \emptyset)$, l'ensemble des termes clos construits sur F sera aussi noté $T(F)$. Etant donné un terme t, on note $|t|$ le nombre de symboles fonctionnels figurant dans t.

Etant donnée une assignation de sorte δ définie de X dans S, un terme t est de sorte $s \in S$ si et seulement si:

- t est une variable et $\delta(t) \leq s$;
- t est terme non variable de la forme $f(t_1 \ldots t_n)$ et $f : s'_1 \times \ldots \times s'_n \mapsto s' \in OD$ et chaque terme t_i est de sorte s'_i et $s' \leq s$.

L'ensemble $WST(X, \delta)$ des *termes bien sortés* selon δ est défini par:

$$WST(X, \delta) = \{t / t \in T(F, X) \text{ et } \exists s \in S, t \text{ est de sorte } s\}$$

Une signature est dite *régulière* si et seulement si, pour toute assignation de sortes δ définie sur $X \subseteq V$, chaque terme bien sorté a une (unique) plus petite sorte que l'on notera $ls_\delta(t)$ (ou, plus simplement, $ls(t)$ si δ est sous-entendue).

Remarque 8 *On ne s'interessera par la suite qu'à des signatures régulières.*

Une substitution est une application de V dans $T(F, V)$ qui est l'identité presque partout. Toute substitution peut être étendue en un endomorphisme de $T(F, V)$.

Etant donné une substitution σ, on notera:

- $Dom(\sigma) = \{x/x \in V \text{ et } \sigma(x) \neq x\}$;
- $Cod(\sigma) = \{\sigma(x)/x \in V \text{ et } \sigma(x) \neq x\}$;
- $VCOD(\sigma) = \cup_{x \in Dom(\sigma)} VAR(\sigma(x))$.

Etant donné un ensemble V de variables, on note $\preceq_V$ la relation d'ordre partiel définie entre substitutions par $\sigma \preceq_V \sigma'$ si et seulement si il existe une substitution σ'' telle que $\forall x \in V,\ \sigma(x) = \sigma''(\sigma'(x))$. De plus, on notera $\approx_V$ la relation entre substitutions définie par: $\sigma \approx_V \sigma'$ si et seulement si $\sigma \preceq_V \sigma'$ et $\sigma' \preceq_V \sigma$.

L'ensemble $WSS(X, \delta)$ des substitutions bien sortés selon δ est défini par:

$$WSS(X, \delta) = \{\sigma/(VCOD(\sigma) \cup Dom(\sigma)) \subseteq X \text{ and } \forall x \in X, ls_\delta(\sigma(x)) \leq \delta(x)\}$$

8.3 Unification avec sortes ordonnées

Nous présentons un algorithme permettant de résoudre tout problème d'unification avec sortes ordonnées. Suivant une approche désormais classique [MM82], [JK90], cet algorithme sera d'abord décrit à l'aide d'un ensemble de *règles de transformation* qui permettent de transformer tout problème d'unification P en un problème d'unification P' en forme *résolue* à partir duquel un ensemble complet de solutions bien sortées pour le problème P peut être calculé. Puis nous présenterons un algorithme [Cha88] mettant en œuvre ces règles de transformation.

8.3.1 Notations

Une *équation* est une paire de termes notée $t_1 == t_2$. Par la suite, $e, e_1, e_2, \ldots$ désigneront des équations. Un *problème d'unification*, que l'on notera P, est

- soit une équation e
- soit un ensemble d'équations que l'on appellera un *système*, que l'on désignera par $S_1, S_2, \ldots$ et que l'on notera $e_1 \wedge \ldots \wedge e_n$,

Un ensemble de systèmes est aussi appelé une *disjonction de systèmes* et est noté $S_1 \vee \ldots \vee S_n$.

Une substitution σ est une *solution* d'un problème d'unification $t_1 == t_2$ si et seulement si $\sigma(t_1) = \sigma(t_2)$. Une substitution σ est une solution d'un système si σ est une solution de chacune des équations de ce système. Une substitution σ est

une solution d'une disjonction de systèmes si σ est une solution d'au moins un de ces systèmes. On notera $\perp$ un système sans solution.

Etant donnés un problème d'unification P et un ensemble de variables X tel que $VAR(P) \subseteq X$, l'ensemble des solutions σ de P telles que $Dom(\sigma) \subseteq X$ est appelé *l'ensemble des solutions de P définies sur X*. On le note $SU(P, X)$.

En pratique, on cherche à calculer un *ensemble complet de solutions* pour le problème d'unification P. On note un tel ensemble $CSU(P)$. Un ensemble Φ est un ensemble complet de solutions pour un problème d'unification P par rapport à un ensemble de variables $W \supseteq Var(P)$ si et seulement si:

- $\forall \sigma \in \Phi, Dom(\sigma) \subseteq W$ et $VCOD(\sigma) \cap W = \emptyset$;
- $\forall \sigma \in \Phi, \sigma \in SU(P)$;
- $\forall \sigma \in SU(P), \exists \sigma' \in \Phi, \sigma \preceq_{Var(P)} \sigma'$

De plus Φ est dit *minimal* si

- $\forall \sigma, \sigma' \in \Phi, \sigma \preceq_{Var(P)} \sigma' \rightarrow \sigma \approx_{Var(P)} \sigma'$

On définit de manière analogue les notions de *solutions bien sortées*, *ensemble complet de solutions bien sortées* en imposant dans les définitions ci-dessus que toute substitution soit une substitution bien sortée.

8.3.2 Règles de transformation

Définition: Un problème d'unification P est en *forme quasi-résolue* si et seulement si il consiste en un système d'équations de la forme $\bigwedge_{i=1}^{n} x_i == t_i$ tel que:

- pour tout $1 \leq i < j \leq n$, $x_i \neq x_j$
- pour tout $1 \leq i \leq j \leq n$, $x_i \notin Var(t_j)$
- pour tout $1 \leq i \leq n$, $ls(t_i) \leq ls(x_i)$
- pour tout $1 \leq i \leq n$, si $t_i \in Var(P)$ alors pour tout $1 \leq j \leq n$, $x_i \notin Var(t_j)$

De plus étant donné un ensemble W de variables, ce système est en *forme résolue par rapport à W* s'il vérifie en outre:

- pour tout $1 \leq i \leq n$, si $x_i \notin W$ alors il existe j, $1 \leq j < i$ tel que $t_j[x_i]_p$ où $p \neq \Lambda$.

- pour tout $1 \leq i \leq n$, si $x \in Var(t_i)$ et $x \in W$ alors il existe j, $i < j \leq n$ tel que $x == x_j$.

$\diamond$

Cette notion de forme résolue est semblable à celle de "DAG solved form" utilisée dans [JK90]. Remarquons que la quatrième condition de la définition d'une forme quasi-résolue et la deuxième condition de la définition d'une forme résolue par rapport à W imposent que, dans un système en forme résolue par rapport à W, il n'existe pas d'équation de la forme $u == v$ où $u \notin W$ et $v \notin W$.

Afin de pouvoir extraire un ensemble complet de solutions d'un problème d'unification P à partir de la forme résolue obtenue par transformation, il est nécessaire que les transformations vérifient les propriétés suivantes:

Définition: Une règle de transformation Tr *préserve les solutions* d'un problème d'unification P si et seulement si elle vérifie:

- $SU(Tr(P)) \mid_{Var(P)} \subseteq SU(P)$ (correction)
- $SU(P) \subseteq SU(Tr(P))$ (complétude).

Les problèmes P et $Tr(P)$ sont dits *équivalents.* $\diamond$

On considère un ensemble de règles de transformations appelé **U.S.O.** (Unification avec Sortes Ordonnées). Ces règles, décrites figures 8.1 et 8.2, sont applicables à une paire $(P; S)$ où

- P est un multi-ensemble d'équations
- S est une séquence de paires ordonnées de termes de la forme $< x, t >$.

Chaque paire $(P; S) = (\bigwedge_{i=1}^{m} t_i == t'_i; << x_1, s_1 > \ldots < x_n, s_n >>)$ représente le système d'équations:

$$(\bigwedge_{i=1}^{m} t_i == t'_i) \wedge (\bigwedge_{j=1}^{n} x_j == s_j)$$

On identifiera par la suite une paire $(P; S)$ avec le système d'équations qu'elle représente.

Les règles Affaiblissement1 et Affaiblissement2 sont *non déterministes* en ce sens qu'elles définissent plusieurs transformations qui peuvent être appliquées en même temps sur la même paire $(P; S)$. On définit alors [JK90]:

Figure 8.1: Ensemble de règles de transformation **U.S.O.**

- Elimination

$$\frac{(t == t \wedge P; S)}{(P; S)}$$

- Décomposition

$$\frac{(f(t_1 \ldots t_n) == f(t'_1 \ldots t'_n) \wedge P; S)}{(t_1 == t'_1 \wedge \ldots \wedge t_n == t'_n \wedge P; S)}$$

- Clash1

$$\frac{(f(t_1 \ldots t_n) == g(t'_1 \ldots t'_{n'}) \wedge P; S)}{(\{\}; \bot)}$$

si $f \neq g$

- Occur-check*

$$\frac{(x_1 == t_1[x_2]_{p_1} \wedge \ldots \wedge x_n == t_n[x_1]_{p_n} \wedge P; S)}{(\{\}; \bot)}$$

si $\exists 1 \leq i \leq n,\ p_i \neq \Lambda$.

- Fusion

$$\frac{(x == t_1 \wedge x == t_2 \wedge P; S)}{(x == t_1 \wedge t_1 == t_2 \wedge P; S)}$$

si $0 < |t_1| \leq |t_2|$.

- Transfert

$$\frac{(x == t \wedge P; S)}{(P; S. << x, t >>)}$$

si

  - $x \notin (Var(P) \cup Var(t))$
  - $0 < |t|$
  - $ls(t) \leq ls(x)$

- Remplacement

$$\frac{(x : s == y : s' \wedge P; S)}{(P[x \leftarrow y]; S[x \leftarrow y]. << x : s, y : s' >>)}$$

si $x \neq y$ et $s' \leq s$

Figure 8.2: Ensemble de règles de transformation **U.S.O.** (suite)

Soit $S' \subseteq S$. Posons $D(S') = Max(\{s/s \in S \; et \; \forall s' \in S' s \leq s'\})$.

- Affaiblissement1

$$\frac{(x : s == y : s' \wedge P; S)}{(P[x \leftarrow z : s''; y \leftarrow z : s'']; S[x \leftarrow z : s''; y \leftarrow z : s''].<< x, z : s'' >, < y, z : s'' >>)}$$

si $s'' \in D(\{s, s'\})$, $s \not\leq s'$, $s' \not\leq s$ et $z \notin Var(P) \cup Var(S) \cup \{x, y\}$.

- Clash2

$$\frac{(x : s == y : s' \wedge P; S)}{(\{\}; \perp)}$$

si $D(\{s, s'\}) = \emptyset$ et $s \not\leq s'$ et $s' \not\leq s$.

Posons $IP(f, s) = Max\{[s'_1 \ldots s'_n]/f : s'_1 \times \ldots \times s'_n \mapsto s' \in profile(f) \; et \; s' \leq s\}$

- Affaiblissement2

$$\frac{(x : s == f(t_1 \ldots t_n) \wedge P; S)}{(u_1 : s_1 == t_1 \wedge \ldots \wedge u_n : s_n == t_n \wedge P; S.<< x, f(u_1 \ldots u_n) >>)}$$

si

  - $\forall 1 \leq i \leq n, \; u_i \notin Var(P) \cup Var(S) \cup Var(f(t_1 \ldots t_n))$
  - $[s_1 \ldots s_n] \in IP(f, s)$
  - $ls(f(t_1 \ldots t_n)) \not\leq s$
  - $x \notin (Var(P) \cup Var(f(t_1 \ldots t_n)))$

- Clash3

$$\frac{(x : s == f(t_1 \ldots t_n) \wedge P; S)}{(\{\}; \perp)}$$

si

  - $IP(f, s) = \emptyset$
  - $ls(f(t_1 \ldots t_n)) \not\leq s$
  - $x \notin (Var(P) \cup Var(f(t_1 \ldots t_n)))$

Définition: Une procédure d'unification avec sortes ordonnées est un programme utilisant les règles **U.S.O.** pour générer un arbre de dérivations équitables en appliquant les transformations Affaiblissement1 et Affaiblissement2 de manière non déterministe. $\diamond$

Bien entendu, on a:

Théorème 15 *Les transformations* **U.S.O** *permettent de transformer tout problème d'unification P en un ensemble de problèmes équivalents.*

La preuve de ce théorème s'obtient immédiatement en considérant les résultats obtenus pour des transformations similaires dans [SS87], [Kir88], [JK90]. La différence principale entre les transformations présentées ici et celles que l'on trouve dans les références précitées tient en ce que la règle Affaiblissement2 n'est applicable qu'à des équations de la forme $x == t$ appartenant à un système $x == t \wedge P$ où $x \notin Var(P)$. Cette restriction est motivée par la remarque suivante. La règle Affaiblissement2 impose des contraintes sur la sorte des termes $\sigma(t_i)$ $1 \leq i \leq n$ qui devront être vérifiées pour toute solution σ du problème d'unification considéré. La restriction sur la sélection de l'équation $x == f(t_1 \ldots t_n)$ revient à n'imposer des conditions sur la sorte des des termes $\sigma(t_i)$ que lorsque toutes les conditions imposées à $f(t_1 \ldots t_n)$ ont été calculées.

Par exemple, considérons la signature:

SORTS:

$$s_1 \gg s_2$$
$$s_2 \gg s_3$$

FUNCTION-PROFILES:

$$g: s_1 \mapsto s_1$$
$$g: s_2 \mapsto s_2$$
$$g: s_3 \mapsto s_3$$

et le problème d'unification $x : s_3 == g(y : s_2) \wedge y : s_2 == g(z : s_1)$. Transformer en premier l'équation $x : s_3 == g(y : s_2)$ permet d'imposer de nouvelles conditions sur la sorte de $\sigma(y)$ avant d'imposer des conditions sur la sorte de $\sigma(z)$ comme le montre la déduction suivante:

- $(x : s_3 == g(y : s_2) \wedge y : s_2 == g(z : s_1); <>)$

- $(u : s_3 == y : s_2 \wedge y : s_2 == g(z : s_1); << x : s_3, g(u : s_3) >>)$
- $(u : s_3 == g(z : s_1); << x : s_3, g(u : s_3) >, < y : s_2, u : s_3 >>)$
- $(v : s_3 == z : s_1; << x : s_3, g(u) >, < y : s_2, u : s_3 >, < u : s_3, g(v) >>)$
- $(\{\}; << x : s_3, g(u) >, < y : s_2, u : s_3 >, < u : s_3, g(v) >, < z : s_1, v : s_3 >>)$

Cette restriction sur la sélection de l'équation $x == t$ n'empèche pas de calculer toutes les solutions:

Lemme 20 *A toute paire* $(P; S)$ *au moins une transformation est applicable*

Preuve: Si P contient au moins une équation de la forme $t_1 == t_2$ telle que t_1 et t_2 ne sont pas des variables, alors soit Clash1, soit Décomposition est applicable. Sinon P est de la forme $\bigwedge_{i=1}^{n} x_i == t_i$. Soit $\prec^{OC}$ la relation définie sur $Var(P)$ par $x \prec^{OC} y$ si et seulement si il existe dans P un ensemble d'équations de la forme $x == t_1[x_1]_{p_1} \wedge \ldots \wedge x_n == t_n[y]_{p_n}$ tel que $\exists 1 \leq i \leq n,\ p_i \neq \Lambda$. Alors, [MM82], soit $\prec^{OC}$ contient un cycle et Occur-Check* est applicable, soit $\prec^{OC}$ définit un ordre strict possédant un élément minimal disons x. Alors, l'une des trois possibilités suivantes est réalisée:

1. il existe dans P une équation de la forme $x : s == y : s'$ et soit Remplacement, soit Affaiblissement1, soit Clash2 est applicable.
2. il existe dans P plusieurs équations de la forme $x == t$ avec $|t| > 0$ et Fusion est applicable.
3. il existe dans P exactement une équation de la forme $x == t$ avec $|t| > 0$ et soit Transfert, soit Afaiblissement2, soit Clash3 est applicable.

C.Q.F.D.

Lemme 21 *Si* $(P; S)$ *est obtenue par applications répétées des règles de transformation* **U.S.O.** *à une paire* $(P_0, \{\})$ *alors* S *représente un système en forme quasi-résolue.*

Preuve: Par induction sur le nombre d'applications de règles de transformation. La propriéte étant trivialement vérifiée dans le cas initial, on vérifie ensuite que toutes les transformations de la forme $\frac{(P; S)}{(P'; S'. << x, t >>)}$ vérifient:

- $ls(t) \leq ls(x)$
- $x \notin Var(t)$
- si t est une variable alors $x \notin Var(S')$
- $x \notin Var(P')$ et donc tout transfert ultérieur d'une équation $x' == t'$ en partie gauche vérifie $x' \neq x$ et $x \notin Var(t')$

C.Q.F.D.

D'autre part on peut montrer que:

Lemme 22 *Etant donnée une paire $(P; S)$, l'application répétée des règles* **U.S.O** *termine toujours.*

Preuve: La preuve est très similaire à la preuve de terminaison de l'ensemble de règles MM décrites dans [DJ90], un ensemble de règles de transformation permettant de résoudre tout problème d'unification sans sorte . On considère la relation $\succ_e$ définie par $t_1 == t_2 \succ_e s_1 == s_2$ si et seulement si

- $Max(|t_1|, |t_2|) > Max(|s_1|, |s_2|)$ ou
- $Max(|t_1|, |t_2|) = Max(|s_1|, |s_2|)$ et

$$Max(|t_1|, |t_2|) - Min(|t_1|, |t_2|) > Max(|s_1|, |s_2|) - Min(|s_1|, |s_2|)$$

Soit $\succ_e^*$ l'extension multi-ensemble de $\succ_e$. On vérifie alors que, pour toute règle de transformation $\frac{(P; S)}{(P'; S')}$ de **U.S.O.**, on a: $P \succ_e^* P'$ C.Q.F.D.

Ces propriétés permettent d'établir sans difficulté:

Théorème 16 *Soient P un problème d'unification et soit $W \supseteq Var(P)$ un ensemble de variables. Alors l'application répétée des règles* **U.S.O.** *à $(P, \{\})$ termine toujours et produit un arbre de dérivations dont les feuilles sont étiquetées soit par la paire $(\{\}, \bot)$ si la dérivation correspondante ne produit pas de solution, soit par une paire $(\{\}, S)$ où S correspont à un système en forme quasi-résolue. L'application répétée des règles*

- *Purification*

$$\frac{S' \wedge u == t}{S'}$$

si u est une variable telle que $u \notin Var(P)$ et $u \notin W$

- *Renommage*

$$\frac{S}{S[x : s \leftarrow u : s] \wedge x : s == u : s}$$

si $x \in W$, $x \in Var(S)$, $u \notin Var(S)$ *et il n'existe pas dans* S *d'équation de la forme* $x == t$

à des tels systèmes S *termine toujours et permet d'obtenir un ensemble de systèmes* $\{S'_1 \ldots S'_n\}$ *en forme résolue par rapport à* W. *Chaque système* $S'_i = (\bigwedge_{j=1}^{n_i} x_j^i == t_j^i)$ *définit une substitution* $\sigma_i = \{x_n^i \mapsto t_{n_i} \circ \ldots \circ \{x_1^i \mapsto t_1^i\}^i\}$ *et* $\{\sigma_1 \ldots \sigma_n\}$ *est un ensemble complet de solutions bien sortées par rapport à* W *du problème* P.

8.3.3 Mise en œuvre des règles de transformations

Un problème d'unification P est donc susceptible d'avoir comme ensemble complet de solutions un ensemble contenant plusieurs éléments. Cette propriété peu désirable en pratique est due au caractère non déterministe des règles Affaiblissement1 et Affaiblissement2 présentées plus haut et est illustrée par les exemples suivants.

Exemple1: Considérons la signature:

SORTS:

TOP $\gg$ A, B

A $\gg$ C, D

B $\gg$ C, D

FUNCTION-PROFILES:

$$f : TOP \times \ldots \times TOP \mapsto TOP$$

Dans cette signature, le problème d'unification $x^A == y^B$ possède deux solutions bien sortées:

$$\{x^A \leftarrow z_1^C;\ \ y^B \leftarrow z_2^C\}$$
$$\{x^A \leftarrow z_3^D;\ \ y^B \leftarrow z_4^D\}$$

Aussi le problème d'unification:

$$f(x_1^A, \ldots, x_n^A) == f(y_1^B, \ldots, y_n^B)$$

possède 2^n solutions.

Une solution à ce problème consiste à transformer la signature ci-dessus en une signature vérifiant que, pour tous symboles de sorte s_1, s_2, $D(\{s_1, s_2\})$ est soit vide soit un singleton. Une telle transformation est donnée dans [SS85b] p 26. Cette solution a été retenue dans le cadre d'ATINF. On montre:

Lemme 23 *Etant donnés un ensemble de symboles fonctionnels F et un ensemble de symboles de prédicat P, toute signature $\Sigma =< S, \leq, SD, OD >$ peut être étendue en une signature $\Sigma' =< S', \leq', SD', OD' >$ telle que:*

- $S \subseteq S'$;
- *le graphe de la relation $\leq$ est inclus dans celui de $\leq'$;*
- $OD \subseteq OD'$;
- $\forall s_1, s_2 \in S'$, $D(\{s_1, s_2\})$ *est soit vide, soit un singleton;*
- $\forall s' \in S', \exists s_1, s_2 \in S,\ s_1 \leq' s' \leq' s_2$.

La paire $(S', \leq')$ est appelée un semi-treillis. *De plus, toute formule a un Σ-(E)-modèle si et seulement si elle a un Σ'-(E)-modèle.*

Le résultat de cette transformation appliquée à l'exemple ci-dessus est le suivant:

SORTS:

TOP $\gg$ A, B
A $\gg$ E
B $\gg$ E
E $\gg$ C, D

FUNCTION-PROFILES:

$$f:\ TOP\ \times \ldots \times\ TOP\ \mapsto\ TOP$$

Dans cette nouvelle signature, le problème d'unification $x^A == y^b$ possède une unique solution bien sortée:

$$\{x^A \leftarrow z_5^E; y^B \leftarrow z_6^E\}$$

En fait on peut montrer:

Lemme 24 *Si $\Sigma =< S, \leq, SD, OD >$ est une signature vérifiant:*

- $(S, \leq)$ *est un semi-treillis;*
- *pour tout $s \in S, f \in F$, $IP(f, s)$ contient au plus un élément*

alors tout problème d'unification au plus une solution bien sortée.

Cependant, même si $(S, \leq)$ est un semi-treillis, il est possible d'obtenir un nombre exponentiel de solutions à un problème d'unification avec sortes ordonnées si la condition

pour tout $s \in S, f \in F$, $IP(f,s)$ contient au plus un élément

n'est pas vérifiée.

Exemple 2: On considère une description des entiers naturels au moyen d'une logique avec sortes ordonnées: parmi les entiers naturels (sorte NAT), on distingue la classe (représentée par la sorte Z) des termes dénotant l'entier zéro et la classe des termes dénotant des entiers strictement positifs (représentés par la sorte NZ). On considère alors une description possible des opérateurs d'addition + et de multiplication $\times$ définis sur les entiers naturels.

SORTS:

$$\text{NAT} \gg \text{Z, NZ}$$

FUNCTION-PROFILES:

$$\begin{aligned}
+ &: NAT \times NAT \mapsto NAT \\
+ &: NZ \times NAT \mapsto NZ \\
+ &: NAT \times NZ \mapsto NZ \\
+ &: Z \times Z \mapsto Z \\
\times &: NAT \times NAT \mapsto NAT \\
\times &: Z \times NAT \mapsto Z \\
\times &: NAT \times Z \mapsto Z \\
\times &: NZ \times NZ \mapsto NZ \\
a &: \mapsto NZ
\end{aligned}$$

Le problème d'unification:

$$x^{NZ} == ((x_1^{NAT} + x_2^{NAT}) \times (\ldots \times (x_{2n-1}^{NAT} + x_{2n}^{NAT})))$$

possède 2^n solutions de la forme

$$\begin{aligned}
\{x^{NZ} &== ((z_1^{s_1} + z_2^{s_2}) \times (\ldots \times (z_{2n-1}^{s_{2n-1}} + z_{2n}^{s_{2n}}))), \\
x_1^{NAT} &== z_1^{s_1}, \\
&\ldots \\
x_{2n-1}^{NAT} &== z_{2n-1}^{s_{2n-1}}\}
\end{aligned}$$

avec, pour $1 \leq i \leq n$, (s_{2i-1}, s_{2i}) est soit la paire (NAT, NZ), soit la paire (NZ, NAT). En effet, pour que le produit de n monômes $(x_{2i-1} + x_{2i})$ soit de sorte NZ, il faut que chacun des monômes soit de sorte NZ (cf règle Affaiblissement2 avec $IP(\times, NZ) = \{[NZ, NZ]\}$). Pour ce faire, il faut qu'au moins

une des deux variables x_{2i-1}, x_{2i} soit de sorte NZ. (cf règle Affaiblissement2 avec $IP(+, NZ) = \{[NZ, NAT], [NAT, NZ]\}$).

L'existence du lemme 23 nous permet désormais de ne considérer que des signatures dont l'ensemble des sortes est un semi-treillis. Seule la règle Affaiblissement2 reste donc délicate à mettre en œuvre puisqu'elle permet d'obtenir une disjonction de systèmes à partir d'un système. Bien sûr, il est trop coûteux en mémoire de produire explicitement une représentation de cette disjonction. Une meilleur solution consiste à mettre en œuvre un mécanisme qui calcule l'ensemble des solutions induit par chacun des systèmes de la disjonction l'un après l'autre.

Notons enfin que l'application de cette règle de même que l'application de la règle Transfert nécessite une sélection d'une équation de la forme $x == t$ dans un système $x == t \wedge P$ où $x \notin Var(P)$. Afin de pouvoir réaliser ce test efficacement, il est nécessaire de mettre en œuvre une bonne structure de données représentant le système $x == t \wedge P$. Ce problème est semblable à celui du traitement de l'occur-check dans l'algorithme de [MM82] et reçoit un traitement similaire:

Définition: Un système est dit *décomposé* si et seulement si il est constitué d'un ensemble d'équations $\bigwedge_{i=1}^{n} x_i == t_i$ vérifiant:

- $\forall 1 \leq i < j \leq n,\ x_i \neq x_j$
- $\forall 1 \leq i \leq n$, t_i n'est pas une variable
- la relation $\prec^{OC}$ est sans cycle.

$\diamond$

Etant donné un problème d'unification P et une signature régulière $\Sigma =< S, \leq, SD, OD >$ telle que $(S, \leq)$ est un semi-treillis, on peut soit montrer que P n'admet pas de solution, soit transformer P en un système décomposé en temps linéaire en fonction de $|P|$ en utilisant les règles Elimination, Décomposition, Clash1, Occur-Check*, Fusion, Remplacement, Clash2 et Affaiblissement1. Pour ce faire, il suffit d'adapter les algorithmes [PW78], [MM76]. En pratique, on utilisera de préférence soit l'algorithme décrit dans [MM82] qui "intègre" le test d'occur-check, soit un algorithme dérivé de celui de Huet [Hue76] qui teste la condition d'occur-check lorsqu'aucune autre règle n'est applicable. Ce dernier algorithme est sans doute préférable puisqu'il favorise la détection de l'application des règles Clash1 (voir [MM82]) et Clash2.

Sans perte de généralité, on peut supposer que toutes les équations de la forme $x == f(t_1, \ldots, t_n)$ d'un système décomposé P sont telles que tous les termes t_i, $1 \leq i \leq n$ sont des variables distinctes de celles apparaissant dans le problème initial P_0[1].

La relation $\prec^{OC}$ définie sur $Var(P)$ s'étend alors en une relation $\prec_P$ définies sur les équations de P par

$$x_1 == t_1 \prec_P x_2 == t_2 \text{ si et seulement si } x_1 \prec^{OC} x_2$$

On obtient alors

Lemme 25 *Etant donné un système décomposé P, la relation $\prec_P$ est irréflexive.*

Le graphe de la relation $\prec_P$ peut être représenté par un graphe acyclique dont les nœuds sont étiquetés par un terme t et une variable $x : s$. De plus, si un nœud N est étiqueté par le terme $t = f(x_1 \ldots x_n)$, alors le i-ème nœud successeur N_i de N correspond à l'équation $x_i == t_i$.

On supposera que chaque nœud N correspondant à l'équation $x : s == t$ comporte les informations supplémentaires suivantes:

- un booléen "RÉSOLU"; initialement, ce booléen est à la valeur FAUX;
- la sorte $ls(t)$ du terme t étiquetant le nœud N;
- la liste de (pointeurs vers les) nœuds dont N est le successeur immédiat.

Ce graphe étant acyclique, il possède un ensemble de nœuds non résolus qui ou bien n'ont pas de prédécesseur, ou bien ont tous leurs prédécesseurs marqués résolus. Ces nœuds seront appelés *racines.*

Il est possible de simuler sur ce graphe l'application des règles **U.S.O** de sorte que chaque modification d'un graphe G en un graphe G' correspond en fait à la transformation d'une paire $(P; S)$ en une paire $(P'; S')$ telle que:

- P, P' sont des systèmes décomposés
- le sous-graphe de G (resp. G') ne contenant que les nœuds marqués non résolus représente le graphe de la relation $\prec_P$ (resp. $\prec_{P'}$)

[1]Si un système décomposé P contient une équation de la forme $x == f(\ldots, t_i, \ldots,)$ telle que t_i est soit un terme non variable, soit une variable figurant dans le problème initial, il suffit de remplacer dans le système P cette équation par $x == f(\ldots, y_i, \ldots,) \wedge y_i : ls(t_i) == t_i$ où $y_i \notin Var(P) \cup Var(P_0)$: le nouveau problème d'unification est alors équivalent au problème P.

- les nœuds racines du graphe G (resp. G') représentent les éléments minimaux de P (resp P') pour la relation $\prec_P$ (resp $\prec_{P'}$).

En fait nous allons représenter tout système P d'équations:

$$< e_{p+1} \ldots e_q, < e_1 \ldots e_p >>$$

vérifiant

$$\forall i, j \text{ si } \; p+1 \leq i < j \leq q \text{ alors } e_j \not\prec_P e_i$$

par une pile contenant n nœuds telle que

- $1 \leq i \leq n-1$ le nœud N_i représentant l'équation e_i figure immédiatement au-dessus du nœud N_{i+1} représentant l'équation e_{i+1}
- les nœuds $N_1, \ldots, N_p$ sont marqués résolus
- un pointeur de pile, appelé sommet de pile, désigne le nœud N_{p+1}: c'est le nœud correspondant à l'équation minimale pour l'ordre $\prec_P$. Cette équation doit donc être traitée en priorité.

Exemple: soit la signature

SORTS:

$$s_1 \gg s_2, s_3$$

FUNCTION-PROFILES:

$$\begin{array}{lllll} f: & s_1 & \times & s_1 & \mapsto & s_1 \\ f: & s_1 & \times & s_2 & \mapsto & s_2 \\ f: & s_3 & \times & s_3 & \mapsto & s_2 \end{array}$$

$$\begin{array}{lllll} g: & s_1 & \times & s_1 & \mapsto & s_1 \\ g: & s_2 & \times & s_2 & \mapsto & s_2 \\ g: & s_3 & \times & s_3 & \mapsto & s_3 \end{array}$$

$$\begin{array}{lll} h: & s_1 & \mapsto & s_1 \\ h: & s_2 & \mapsto & s_2 \end{array}$$

Considérons le problème décomposé

$$x : s_2 == f(y,z) \wedge z : s_1 == g(y,y) \wedge y : s_1 == h(u) \wedge u : s_1 == v : s_1$$

Figure 8.3: Représentation d'un système décomposé à l'aide d'une pile

nœud:	1
terme:	$f(y,z)$
variable:	$x : s_2$
résolu:	FAUX
sorte:	s_1
pred:	[]
nœud:	2
terme:	$g(y,y)$
variable:	$z : s_1$
résolu:	FAUX
sorte:	s_1
pred:	[1]
nœud:	3
terme:	$h(u)$
variable:	$y : s_1$
résolu:	FAUX
sorte:	s_1
pred:	[1,2]
nœud:	4
terme:	v
variable:	$u : s_1$
résolu:	FAUX
sorte:	s_1
pred:	[3]

On représente ce système d'équations par la pile décrite à la figure 8.3.

Décrivons comment s'opèrent sur la pile les transformations USO.

Le nœud N figurant en sommet de pile est sélectionné. Ce nœud correspond à une équation $x : s == t$ et seulement trois cas sont possibles:

1. La règle Transfert est alors applicable. Le nœud N est marqué résolu et le sommet de la pile est décrémenté de la taille du nœud N.

2. t est un terme non variable et la règle Affaiblissement2 est applicable. On sélectionne parmi $IP(f, s)$ une séquence $[s_1 \dots s_n]$, les autres choix étant mis en attente. L'application de la règle Affaiblissement2 produit les équations $\bigwedge_{i=1}^{n} u_i : s_i == x_i : s'_i$. Pour chacune de ces équations l'une des règles Clash2, Remplacement, Affaiblissement1 est applicable. Pour déterminer quelle règle appliquer, il suffit de comparer la sorte de la variable u_i avec la valeur du champs *sorte* du nœud N_i, i-ème successeur du nœud N. L'application de la règle Clash2 provoque une remise en cause du choix de $[s_1 \dots s_n]$ dans $IP(f, s)$, ou d'un choix précédent si $IP(f, s)$ ne contient plus d'autre alternative. L'application de la règle Affaiblissement2 provoque un affaiblissement de la sorte de la variable du nœud N_i et donc de la plus petite sorte des nœuds ayant N_i pour successeur. Pour remettre à jour le champs *sorte* de ces nœuds, on parcourt la liste des prédécesseurs de N_i. Si cette remise à jour d'un nœud ancêtre produit une équation $x' : s' == t'$ telle que la règle Clash3 soit applicable, le choix de $[s_1 \dots s_n]$ dans $IP(f, s)$ est remis en cause. Si toutes les équations $\bigwedge_{i=1}^{n} u_i : s_i == x_i : s'_i$ peuvent être traitées sans qu'il y ait d'échec, le nœud N est marqué résolu et le sommet de la pile est décrémenté de la taille du nœud N.

3. t est une variable et la règle Affaiblissement1 est applicable. Comme dans le cas précédent, on cherche à mettre à jour le champ *sorte* des nœuds ayant N comme successeur. En cas d'échec, on remet en cause un choix précédent; sinon, le nœud N est marqué résolu et le sommet de pile est décrémenté.

Remarquons que la règle Clash3 est prise en compte lors des étapes 2 et 3 et ne peut être appliquée à un nœud figurant en sommet de pile.

8.3.4 Modification des règles de transformation

La mise à jour du champ *sorte* d'un nœud – c'est à dire le calcul de $ls(t)$ pour une équation $x : s == t$ – peut dans certains cas nécessiter un temps de calcul

non négligeable. Afin de retrouver facilement les nœuds pour lesquels une telle mise à jour est nécessaire, nous avons utilisé une structure de données dans laquelle un nœud à accès à la liste de ces prédécesseurs immédiats. Cependant, cette représentation, coûteuse en mémoire, n'est pas toujours compatible avec certains choix d'implémentation: en particulier, elle ne peut être mise en œuvre dans le cadre du partage de structures de Boyer et Moore sans utiliser une zone spéciale de recopie.

C'est pourquoi nous avons présenté dans [Cha88] [Cha89] un ensemble de transformations permettant de résoudre un problème d'unification avec sortes ordonnées sans avoir à calculer la plus petite sorte d'un terme. Ces règles de transformations sont décrites à la figure 8.4, 8.5. Les preuves de correction, de complétude et de terminaison de ces transformations sont très similaires à celles données précédemment.

Les différences entre les deux ensembles de transformations reposent sur le fait que, dans le deuxième ensemble de transformations, la règle Transfert n'est applicable qu'à des équations $x : s == t$ où t est une constante. On applique donc systématiquement la règle d'Affaiblissement2 à des équations de la forme $x : s == f(t_1 \ldots t_n)$, $n \geq 1$ sans se soucier de $ls(f(t_1 \ldots t_n))$ alors que, en utilisant le premier ensemble de transformations, on applique l'une des règles Affaiblissement2, Transfert ou Clash2 suivant la valeur de $ls(f(t_1 \ldots t_n))$. Certains cas d'échec ou de succès nécessitent plus d'applications de règles de transformation pour être détectés par notre méthode. Cependant, la plus grande simplicité de notre algorithme et son caractère systématique permettent de compenser le surcoût de calcul pour des problèmes d'unification de taille modérée.

8.3.5 Expérimentations

Afin d'illustrer ce propos, nous avons testé les deux algorithmes sur un problème combinatoire cité dans [JOR88]. Il s'agit, étant données trois boites A, B C de placer n jetons numérotés de 1 à n dans chacune de ces boites de sorte que, si les jetons i et j figurent dans une même boite, alors les jetons $i+j$, $2i$ n'y figurent pas. Ce problème n'a pas de solution pour n plus grand que 14.

On considère sept sortes correspondant à l'ensemble des parties non vides de $\{A, B, C\}$, l'ordre sur les symboles de sortes correspondant à l'ordre de l'inclusion entre les parties de $\{A, B, C\}$. On dispose de trois autres sortes $Bool$, $Vrai$, $Faux$ dénotant les valeurs booléennes usuelles et l'on a $Bool \gg Vrai, Faux$. On considère en outre les déclarations de fonctions suivantes:

Figure 8.4: Ensemble de règles de transformation **U.S.O.2**

- Elimination

$$\frac{(t == t \wedge P; S)}{(P; S)}$$

- Décomposition

$$\frac{(f(t_1 \ldots t_n) == f(t'_1 \ldots t'_n) \wedge P; S)}{(t_1 == t'_1 \wedge \ldots \wedge t_n == t'_n \wedge P; S)}$$

- Clash1

$$\frac{(f(t_1 \ldots t_n) == g(t'_1 \ldots t'_{n'}) \wedge P; S)}{(\{\}; \bot)}$$

si $f \neq g$

- Occur-check*

$$\frac{(x_1 == t_1[x_2]_{p_1} \wedge \ldots \wedge x_n == t_n[x_1]_{p_n} \wedge P; S)}{(\{\}; \bot)}$$

si $\exists 1 \leq i \leq n,\ p_i \neq \Lambda$.

- Fusion

$$\frac{(x == t_1 \wedge x == t_2 \wedge P; S)}{(x == t_1 \wedge t_1 == t_2 \wedge P; S)}$$

si $0 < |t_1| \leq |t_2|$.

- Transfert

$$\frac{(x == t \wedge P; S)}{(P; S. << x, t >>)}$$

si

  - $x \notin (Var(P) \cup Var(t))$
  - $|t| = 1$
  - $ls(t) \leq ls(x)$

- Remplacement

$$\frac{(x : s == y : s' \wedge P; S)}{(P[x \leftarrow y]; S[x \leftarrow y]. << x : s, y : s' >>)}$$

si $x \neq y$ et $s' \leq s$

Figure 8.5: Ensemble de règles de transformation **U.S.O.2** (suite)

Soit $S' \subseteq S$. Posons $D(S') = Max(\{s/s \in S \text{ et } \forall s' \in S' s \leq s'\})$.

- Affaiblissement1

$$\frac{(x : s == y : s' \wedge P; S)}{(P[x \leftarrow z : s''; y \leftarrow z : s'']; S[x \leftarrow z : s''; y \leftarrow z : s''].<< x, z : s'' >, < y, z : s'' >>)}$$

si $s'' \in D(\{s, s'\})$, $s \not\leq s'$, $s' \not\leq s$ et $z \notin Var(P) \cup Var(S) \cup \{x, y\}$.

- Clash2

$$\frac{(x : s == y : s' \wedge P; S)}{(\{\}; \bot)}$$

si $D(\{s, s'\}) = \emptyset$ et $s \not\leq s'$ et $s' \not\leq s$.

Posons $IP(f, s) = Max\{[s'_1 \ldots s'_n]/f : s'_1 \times \ldots \times s'_n \mapsto s' \in profile(f) \text{ et } s' \leq s\}$

- Affaiblissement2

$$\frac{(x : s == f(t_1 \ldots t_n) \wedge P; S)}{(u_1 : s_1 == t_1 \wedge \ldots \wedge u_n : s_n == t_n \wedge P; S.<< x, f(u_1 \ldots u_n) >>)}$$

si

  - $\forall 1 \leq i \leq n,\ u_i \notin Var(P) \cup Var(S) \cup Var(f(t_1 \ldots t_n))$
  - $[s_1 \ldots s_n] \in IP(f, s)$
  - $x \notin (Var(P) \cup Var(f(t_1 \ldots t_n)))$

- Clash3

$$\frac{(x : s == f(t_1 \ldots t_n) \wedge P; S)}{(\{\}; \bot)}$$

si

  - $IP(f, s) = \emptyset$
  - $x \notin (Var(P) \cup Var(f(t_1 \ldots t_n)))$

$$
\begin{array}{lrcrcrcl}
plus: & ABC & \times & ABC & \times & ABC & \mapsto & Bool \\
plus: & A & \times & BC & \times & ABC & \mapsto & Vrai \\
plus: & A & \times & A & \times & BC & \mapsto & Vrai \\
plus: & A & \times & A & \times & A & \mapsto & Faux \\
plus: & B & \times & AC & \times & ABC & \mapsto & Vrai \\
plus: & B & \times & B & \times & AC & \mapsto & Vrai \\
plus: & B & \times & B & \times & B & \mapsto & Faux \\
plus: & C & \times & AB & \times & ABC & \mapsto & Vrai \\
plus: & C & \times & C & \times & AB & \mapsto & Vrai \\
plus: & C & \times & C & \times & C & \mapsto & Faux \\
deux-fois: & ABC & \times & ABC & & & \mapsto & Bool \\
deux-fois: & A & \times & BC & & & \mapsto & Vrai \\
deux-fois: & A & \times & A & & & \mapsto & Faux \\
deux-fois: & B & \times & AC & & & \mapsto & Vrai \\
deux-fois: & B & \times & B & & & \mapsto & Faux \\
deux-fois: & C & \times & AB & & & \mapsto & Vrai \\
deux-fois: & C & \times & C & & & \mapsto & Faux \\
\wedge: & Bool & \times & Bool & & & \mapsto & Bool \\
\wedge: & Faux & \times & Bool & & & \mapsto & Faux \\
\wedge: & Bool & \times & Faux & & & \mapsto & Faux \\
\wedge: & Vrai & \times & Vrai & & & \mapsto & Vrai
\end{array}
$$

On considère alors n variables $x_1 \ldots x_n$ de sortes ABC correspondant aux n jetons et l'on exprime les conditions imposées sur les jetons par une conjonction C de termes de la forme

$$plus(x_i, x_j, x_{i+j}) \quad i \neq j$$

$$deux - fois(x_i, x_{2i})$$

Le problème initial a une solution si et seulement si le problème d'unification $C == x : Vrai$ à une solution. Par exemple, pour $n = 4$, la solution

$$\{x_1 \leftarrow y_1 : AB; x_2 \leftarrow y_2 : C; x_3 \leftarrow y_3 : C; x_4 \leftarrow y_4 : AB\}$$

indique que les jetons numérotés 1 et 4 peuvent être placés indifféremment dans les deux premières boites pourvu que les jetons numérotés 2 et 3 soient placés dans la troisième boite.

Nous donnons le temps (en secondes) de calcul de toutes les substitutions trouvées par les algorithmes pour des valeurs de n comprises entre 10 et 14, ce

qui représente des termes de taille considérable. Nous indiquons en outre le nombre de solutions, le nombre de substitutions trouvées (une substitution pouvant dénoter plusieurs solutions, comme dans l'exemple précédent) ainsi que le nombre de conditions de la forme $plus(x_i, x_j, x_{i+j})$, $deux - fois(x_i), x_{2i})$ utilisées.

problème	conditions	solutions	substitutions	algorithme 1	algorithme 2
n = 10	25	300	156	3.1	2.5
n = 11	30	186	108	5.0	6.8
n = 12	36	114	72	9.2	14.7
n = 13	42	18	18	14.6	35.4
n = 14	49	0	0	25.2	86.3

On remarque que plus la taille du problème d'unification est importante et plus le nombre de contraintes (c'est à dire le nombre de conditions par variable) est élevé, plus il est avantageux d'utiliser l'algorithme 1 qui détecte au plus tôt les cas d'échec. Par contre, pour des problèmes plus petits ou moins contraints, l'algorithme 2 est préférable.

8.3.6 Conclusion

La mise en œuvre de l'unification avec sortes ordonnées est un problème délicat car:

- il existe des problèmes d'unification possédant un nombre exponentiel de solutions;

- il est possible de traduire des problèmes NP-complets en problème d'unification avec sortes ordonnées. Par exemple, la signature:

  SORTS:

  $$\text{BOOL} \gg \text{FALSE, TRUE}$$

  FUNCTION-PROFILES:

  $$\begin{aligned} \vee : & \; BOOL \times BOOL \mapsto BOOL \\ \vee : & \; TRUE \times BOOL \mapsto TRUE \\ \vee : & \; BOOL \times TRUE \mapsto TRUE \\ \vee : & \; FALSE \times FALSE \mapsto FALSE \end{aligned}$$

$$
\begin{array}{lrlrl}
\wedge: & BOOL & \times & BOOL & \mapsto BOOL \\
\wedge: & FALSE & \times & BOOL & \mapsto FALSE \\
\wedge: & BOOL & \times & FALSE & \mapsto FALSE \\
\wedge: & TRUE & \times & TRUE & \mapsto TRUE \\
\neg: & BOOL & \mapsto & BOOL & \\
\neg: & FALSE & \mapsto & TRUE & \\
\neg: & TRUE & \mapsto & FALSE &
\end{array}
$$

permet d'exprimer sous la forme d'une équation $x : TRUE == t$ le problème de la satisfaisabilité d'une formule propositionnelle. Donc la résolution d'un problème d'unification avec sortes ordonnées peut être très complexe même en l'absence de solution.

Cependant, l'analyse suivante montre que l'algorithme que nous proposons est efficace:

- dans le cas où $(S, \leq)$ est un semi-treillis et quand pour tout $s \in S, f \in F$, $IP(f, s)$ contient au plus un élément, il permet de calculer en temps *linéaire* si un problème d'unification possède une solution;
- dans les autres cas,
  - il effectue en priorité la transformation d'un système en un système décomposé, une transformation qui peut être réalisée en temps (quasi-)linéaire et qui favorise la détection des cas d'échec simples;
  - le traitement des équations donnant lieu éventuellement à une forte combinatoire est effectué de manière à minimiser l'espace de recherche en limitant les possibilités d'application de la règle Affaiblissement2 tout en mettant en place une structure de données permettant de sélectionner efficacement les équations sur lesquelles cette règle est applicable.

De plus, il est possible d'intégrer dans notre algorithme un mécanisme permettant un retour-arrière intelligent. Mais ceci se fait au prix d'une plus grande consommation de mémoire.

8.4 Une représentation des clauses avec sortes ordonnées

Considérons l'ensemble de clauses suivant:

1. $p(x^{NZ}) \vee q(x^{NZ})$

2. $\neg q(x^{NZ} \times y^{NAT}) \vee q(y^{NAT})$

3. $\neg q(x^{Z} + y^{NAT}) \vee q(y^{NAT})$

4. $\neg q(a)$

5. $\neg p(((x_1^{NAT} + x_2^{NAT}) \times (\ldots \times (x_{2n-1}^{NAT} + x_{2n}^{NAT}))))$

Afin de montrer que cet ensemble de clauses est insatisfaisable, on applique la résolution avec la stratégie du support. Supposons que la clause (5) soit initialement dans le support.

- *Pas 1.* La clause (5) est résolue avec la clause (1). Le problème d'unification ayant 2^n solutions[2], 2^n clauses de la forme:

  $$q((z_1^{s_1} + z_2^{s_2}) \times (\ldots \times (z_{2n-1}^{s_{2n-1}} + z_{2n}^{s_{2n}})))$$

  sont ajoutées au support.

- *Pas 2.* Ces clauses sont ensuites résolue avec la clause (2), ce qui produit de nouveau 2^n clauses.

  $$q((z_3^{s_3} + z_3^{s_3}) \times (\ldots \times (z_{2n-1}^{s_{2n-1}} + z_{2n}^{s_{2n}})))$$

  Cependant, certaines de ces clauses sont identiques à un renommage (bien sorté) de leurs variables près : il n'existe en effet que 2^{n-1} clauses différentes.
  ...

- *Pas i.* Les 2^n clauses produites au pas $i-1$ génèrent en un pas de résolution avec la clause (2) 2^n nouvelles clauses,

  $$q((z_{2i-1}^{s_{2i-1}} + z_{2i}^{s_{2i}}) \times (\ldots \times (z_{2n-1}^{s_{2n-1}} + z_{2n}^{s_{2n}})))$$

  dont seulement 2^{n-i+1} sont en fait différentes: pour chaque clause il existe donc 2^{i-1} clauses de chaque type.
  ...

[2]Par la suite, nous tirerons parti du fait que ce problème d'unification à plusieurs solutions les plus générales; le fait que le nombre de telles solutions soit en fait exponentiel ne nous intéresse pas ici.

- *Pas n* On obtient alors 2^{n-1} clauses de l'un des deux types suivants:

$$q(x_{2n-1}^{NZ} + x_{2n}^{NAT})$$

$$q(x_{2n-1}^{NAT} + x_{2n}^{NZ})$$

- *Pas n + 1* Ces clauses sont résolues avec la clause (3) ce qui produit 2^{n-1} fois la clause $q(x_{2n}^{NZ})$.

- *Pas n+1* Une contradiction est finalement trouvée en résolvant $q(x_{2n}^{NZ})$ avec (4)

Durant cette preuve, plus de 2^n clauses ont été générées et plusieurs d'entre elles sont redondantes. La cause de ces redondances est la suivante.

Etant données deux clauses C_1, C_2 dont les ensembles respectifs des variables sont V_1, V_2, $V_1 \cap V_2 = \emptyset$, la production de l'ensemble de résolvantes binaires obtenues en résolvant le littéral L_1 de C_1 contre le littéral L_2 de C_2 nécessite la résolution d'un problème d'unification P tel que $Var(P) = V = V_1 \cup V_2$ et donc un ensemble minimal et complet de solutions hors de V est calculé. Supposons que l'ensemble de variables $V' = Var((C_1 \setminus L_1) \cup (C_2 \setminus L_2))$ soit *strictement* inclus dans V. Dans ce cas, il se peut qu'il y ait deux substitutions σ_1, σ_2 de l'ensemble minimal et complet de solutions de P telles que $\sigma_1 \not\approx_V \sigma_2$ mais $\sigma_1 \approx_{V'} \sigma_2$. Les résolvantes $\sigma_1((C_1 \setminus L_1) \cup (C_2 \setminus L_2))$ et $\sigma_2((C_1 \setminus L_1) \cup (C_2 \setminus L_2))$ sont alors identiques à un renommage bien sorté de leurs variables près. Il est donc nécessaire de tenir compte de ce phénomène avant de produire un ensemble de résolvantes obtenues par résolution, paramodulation ou factorisation.

Remarquons enfin que pour toutes substitutions σ, σ' appartenant à un ensemble complet de solutions d'un problème d'unification $t_1 = t_2$, $\sigma(t)$ et $\sigma(t')$ ont la même *structure*, c'est-à-dire sont identiques au nom de leur variable près. Donc tout ensemble complet de solutions pour un tel problème d'unification peut être représenté sous la forme d'une paire $< \sigma, (\delta_i)_{1 \leq i \leq p} >$ où $(\delta_i)_{1 \leq i \leq p}$ désigne une famille d'assignations de sorte aux variables de $VCOD(\sigma)$ telle que,si δ est une assignation de sorte aux variables de $V = (Var(t_1) \cup Var(t_2))$ alors

$$\forall 1 \leq i \leq n,\ \sigma \in WSS(V \cup VCOD(\sigma), \delta \circ \delta_i)$$

Ceci nous amène naturellement à représenter les clauses avec sortes ordonnées ayant même structure syntaxique mais dont la sorte de leurs variables diffèrent par une paire:

$$< clause_{sans-sorte}[x_1, \ldots, x_n], (\delta_i)_{1 \leq i \leq p} >$$

où

- $clause_{sans-sorte}[x_1, \ldots, x_n]$ est une représentation d'une clause dont les variables sont $x_1 \ldots x_n$
- $(\delta_i)_{1 \leq i \leq p}$ désigne une famille d'assignations de sorte aux variables $x_1 \ldots x_n$.

Cette représentation possède les avantages suivants:

- elle évite la duplication de la structure syntaxique commune à ces clauses et donc réalise une économie en place mémoire;
- elle permet de définir les opérations de résolution, paramodulation et factorisation sur des *ensembles de clauses.* Par exemple, étant données deux paires $< C_i[x_1 \ldots x_{n_i}, (\delta^i_j)_{1 \leq j \leq p_i} >$, $i = 1, 2$, l'ensemble de résolvantes binaires obtenues par résolution entre les littéraux L_1 de C_1 et L_2 de C_2 est la réunion pour toute paire $(j_1, j_2) \in (\{1 \ldots p_1\} \times \{1 \ldots p_2\})$ de l'ensemble des résolvantes binaires obtenues par résolution entre les littéraux L_1 de $C^1_{j_1} = C_1[x_1 : \delta^1 j_1(x_1) \ldots x_{n_1} : \delta^1 j_1(x_{n_1})]$ et L_2 de $C^2_{j_2} = C_2[x_1 : \delta^2 j_2(x_1) \ldots x_{n_2} : \delta^2 j_2(x_{n_2})]$.
  Cependant, il est possible de tirer parti du partage de structure de la représentation ci-dessus lors du calcul de cet ensemble de résolvantes.
- elle permet d'éliminer facilement les redondances obtenues par le phénomène décrit plus haut.
- elle est compatible avec les arbres de sélection présentés au chapitre précédent.
- elle permet de réutiliser les techniques d'implémentation utilisées pour la mise en œuvre de la Résolution sans sortes.

Chapitre 9

Conclusion

Dans cette thèse, nous avons étudié comment l'utilisation d'une mise sous forme clausale non standard permet d'établir une correspondance naturelle entre les déductions d'un calcul par séquents et les dérivations obtenues en utilisant le principe de Résolution. En particulier, nous avons montré comment traduire les preuves obtenues dans l'un des ces calculs en des déductions de l'autre calcul.

La possibilité de simuler polynomialement le calcul des séquents par la Résolution devrait permettre d'étudier plus finement comment des heuristiques et des méthodes de raisonnement issues de l'expérience humaine, pouvant être facilement reproduites par des démonstrateurs utilisant des calculs en déduction naturelle [Ble77] [OS88], peuvent être interprétées dans le cadre de la Résolution et dans quelle mesure ces schémas de déduction sont compatibles avec les stratégies de Résolution existantes.

Nous avons montré aussi comment traduire des preuves par Résolution en preuves par Déduction Naturelle. Ceci doit permettre de fournir à un utilisateur utilisant un démonstrateur fondé sur la Résolution une justification lisible des déductions réalisées sous forme clausale. En pratique, il est sans doute souhaitable de présenter ces déductions dans un autre système de déduction que le calcul des

séquents. En particulier, l'utilisation d'un système proche du calcul NK de Gentzen constituerait une nette amélioration. En fait, étant donnée la proximité des calculs LK et NK [Pra65] pp 88-93 la traduction directe des preuves par Résolution en preuves du calcul NK ne pose aucune difficulté majeure. La suite logique de ce travail constituera alors à produire une justification formulée en langue naturelle d'une preuve formulée dans un calcul en déduction naturelle. Outre les problèmes posés par la génération de texte, cette traduction présente de nombreuses difficultés dont les suivantes:

- la distinction qualitative entre séquences d'inférences triviales qui devront êtres regroupées et présentées de manière succinte et étapes cruciales du raisonnement qui devront faire l'objet d'une présentation détaillée. Cette distinction est d'autant plus difficile qu'elle dépend du contexte. Par exemple, on n'accordera pas la même importance dans la traduction à la preuve de la commutativité de l'intersection de deux ensembles suivant que cette déduction constitue l'objet principal de la démonstration ou un simple argument de symétrie permettant de conclure la preuve d'un théorème complexe sans avoir à dupliquer un même schéma de preuve.

- la restructuration de preuves suivant le contexte. Suivant le domaine mathématique auquel appartient le théorème dont la preuve est traduite, il sera préférable d'utiliser certains types de preuves ou certaines figures de style. Par exemple si le théorème est un jeu ou une énigme (à la manière du problème du Steamroller), il apparaitra naturel d'utiliser des preuves par contradiction et la présence d'hypothèses superflues sera considérée comme un artifice destiné à augmenter la difficulté de trouver la preuve du théorème en créant un phénomène d'explosion combinatoire. Par contre, s'il s'agit d'un théorème – disons – de topologie, on essaiera autant que possible d'obtenir une preuve directe et la présence d'hypothèses non utilisées prendra alors une toute autre signification. Remarquons que, dans cette thèse, nous n'avons pas chercher à restructurer les preuves. Au contraire, nous avons chercher à donner des traductions qui préservent la structure de celles-ci. Les problèmes liés à la restructuration des preuves sont partiellement abordés dans [Mil87] [Pfe84], [PN90] [Lin90].

En fait, il serait certainement profitable de mener en parallèle à l'étude de la traduction en langue naturelle de preuves de théorèmes obtenues par un système de

démonstration automatique l'étude inverse de l'analyse de preuves formulées en langue naturelle en vue d'en synthétiser une preuve formelle pouvant faire l'objet d'une vérification automatique.

Un autre axe de recherche consistera à étudier comment les résultats établis dans cette thèse peuvent être étendus à d'autres logiques que la logique classique. En effet, on dispose de calcul par séquents pour de nombreuses logiques modales [Fit83] et pour la logique intuitionniste. Or nous avons montré que, dans le cadre de la logique classique, l'utilisation du renommage permettait d'établir une correspondance entre séquents $\Gamma \Longrightarrow \Delta$ d'une dérivation du calcul LK' et clause $C_{\Gamma \Longrightarrow \Delta}$ obtenue par le principe de Résolution. Si une correspondance similaire pouvait être établie dans le cadre de logiques non classiques, il serait alors possible d'envisager une mécanisation efficace de ces logiques dans des calculs proches de la Résolution. En particulier, il existe diverses méthodes de Résolution pour les logiques modales [FndC82] [Ohl88]. Afin de pouvoir comparer ces méthodes entre elles ou avec le calcul des séquents, il serait intéressant de pouvoir obtenir des résultats quant à l'existence de simulations polynomiales entre ces procédures de preuves ainsi que de pouvoir comparer leurs espaces de recherche respectifs.

D'autre part, nous avons montré au chapitre 6 que l'on pouvait augmenter l'efficacité de la Résolution en ne procédant qu'à un renommage partiel de la formule à réfuter. Les résultats concernant l'identification des renommages inutiles doivent être étendus. De plus, il serait intéressant de pouvoir comparer plus précisément les renommages minimisant respectivement le nombre de clauses ou le nombre de littéraux en terme de longueur des réfutations et en terme de taille de l'espace de recherche. Ce dernier problème est un point difficile: les expérimentations réalisées sur le problème d'Andrews ont montré par exemple qu'il est parfois utile de ne pas renommer une formule afin de ne pas créer de définitions comportant trop de variables libres. On conçoit facilement la difficulté qu'il y a à prendre en compte un tel critère dans une analyse de complexité! Sans doute, les expérimentations resteront l'outil privilégié d'investigation dans ce domaine.

Enfin, en ce qui concerne les techniques d'implémenation de démonstrateurs de théorèmes, il serait souhaitable de pouvoir étudier les problèmes suivants:

- dans quelle mesure est-il souhaitable et/ou nécessaire de modifier les structures de données et les algorithmes utilisés pour implémenter des démonstrateurs fondés sur la Résolution afin de tirer un meilleur parti de l'utilisation du renommage?

- la possibilité de simuler pas à pas un calcul des séquents par la Résolution permet-elle d'envisager de manière réaliste que des démonstrateurs par Résolution soient utilisés comme des interpréteurs efficaces de démonstrateurs utilisant la Déduction Naturelle? (La mise sous forme clausale non standard serait alors assimilée à une sorte de compilation dans un langage de plus bas niveau.)

Bibliographie

[AB70] R. Anderson and W.W. Bledsoe. A linear format for resolution with merging and a new technique for establishing completeness. *Journal of the Association for Computing Machinery*, 17(3):525–534, 1970.

[And80] P.B. Andrews. Transforming matings into natural deduction proofs. In W. Bibel and R. Kowalski, editors, *5th Conference on Automated Deduction, Les Arcs, France*, pages 281–292. Springer-Verlag, Lecture Notes in Computer Science 87, 1980.

[And81] P.B. Andrews. Theorem proving via general matings. *Journal of the Association for Computing Machinery*, 28:193–214, 1981.

[And86] P.B. Andrews. *An Introduction to Mathematical Logic and Type Theory: to Truth trough Proofs.* Academic Press, 1986.

[Bac86] L. Bachmair. Proof normalization for resolution and paramodulation. In *LICS*, 1986.

[BB74] W.W. Bledsoe and P. Bruel. A man-machine theorem proving system. *Artificial Intelligence*, 5, 1974.

[BdlT89] T. Boy de la Tour. A locally optimal transformation into clause form using partial formula renaming. Rr 765-i-imag - 90 lifia, institut IMAG, BP 68, 38402 Saint Martin d'Hères cedex, January 1989.

[BdlT90] T. Boy de la Tour. Minimizing the number of clauses by renaming. In M.E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, pages 558–572. Springer Lecture Notes in Artificial Intelligence 449, July 1990.

[BdlT91] T. Boy de la Tour. *Optimisations par Renommage dans la Méthode de Résolution.* PhD thesis, Institut National Polytechnique de Grenoble, 1991.

[BdlTC90a] T. Boy de la Tour and G. Chaminade. Renommage et forme clausale. In *Actes des 3e Journées nationales PRC-GDR Intelligence artificielle*, pages 183–192. éditions Hermès, March 1990.

[BdlTC90b] T. Boy de la Tour and G. Chaminade. The use of renaming to improve the efficiency of clausal theorem proving. In *to appear in the Proceedings of AIMSA '90*, 1990.

[BdlTCC88] T. Boy de la Tour, R. Caferra, and G. Chaminade. Some tools for an inference laboratory (atinf). In E. Lusk and R. Overbeek, editors, *Proccedings of the 9th International Conference on Automated Deduction*, pages 744–745. Springer Lecture Notes in Computer Science 310, 1988.

[BES+81] K. Blãsius, N. Eisinger, J. Siekman, G. Smolka, A. Herold, and C. Walther. The markgraf karl refutation procedure. In *Proceeding of the 7th International Joint Conference on Artificial Intelligence. Vancouver Canada*, 1981.

[BG90] L. Bachmair and H. Ganziger. On restrictions of ordered paramodulation with simplifications. In M.E Stickel, editor, *10th International conference on Automated Deduction, Kaiserslautern, FRG*, pages 427–441. Lecture Notes in Artificial Intelligence, 1990.

[Bib81] W. Bibel. Matrices with connections. *Journal of the Association for Computing Machinery*, 28:633–645, 1981.

[Ble71] W.W. Bledsoe. Splitting and reduction heuristics in automatic theorem proving. *Artificial Intelligence*, 2, 1971.

[Ble77] W.W. Bledsoe. Non-resolution theorem proving. *Artificial Intelligence*, 9, 1977.

[BLMO86] R. Butler, E.L. Lusk, W.W. McCune, and R.A. Overbeek. Path to high-performance automated theorem proving. In J.H. Siekmann, editor, *Proceedings of the 8th Conference on Automated Deduction*, pages 588–597. Springer Lecture Notes in Computer Science 230, 1986.

[BM72] R.S. Boyer and J.S. Moore. The sharing of structure in theorem proving programs. *Machine Intelligence*, 7:101–116, 1972.

[Bru82] M. Bruynooghe. The memory management of prolog implementations. *Logic Programming*, pages 83–98, 1982.

[Cha88] G. Chaminade. Some computational aspects of an order-sorted calculus: Order-sorted unification using compact representation of clauses. In Yves Kodratoff, editor, *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 625–630. Pitman Publishing, August 1988.

[Cha89] G. Chaminade. An implementation oriented view of order-sorted resolution. *Revue d'Intelligence Artificielle*, 3(1):7–30, August 1989.

[CL73] C. Chang and R.C. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, New York, 1973.

[Coh87] A. Cohn. A more expressive foundation of many sorted logic. *Journal of Automated Reasoning*, 3-2:113–200, June 1987.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.

[DJ90] N. Dershowitz and J.P. Jouannaud. Rewrite systems. In Van Leuven, editor, *Handbook of Theoretical Computer Science.* North Holland, 1990.

[Ede84] E. Eder. An implementation of a theorem prover based on the connection method. In W. Bibel and B. Petkoff, editors, *AIMSA'84, Artificial Intelligence—Methodology Systems Application*, pages 121–128. North-Holland, September 1984.

[Ede90] E. Eder. *Relative Complexities of First Order Calculi.* Vieweg, 1990.

[Fel90] A. Felty. A logic program for transforming sequent proofs to natural deduction proofs. Rapport de recherche inria no 1301., INRIA, Sophia Antipolis, October 1990.

[Fit83] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* Synthese Library, Reidel, 1983.

[FLSD74] S. Fleisig, D. Loveland, A.K. Smiley, and Yarmush D.L. An implementation of the model elimination proof procedure. *Journal of the Association for Computing Machinery*, 21:124–139, January 1974.

[FndC82] L. Fariñas del Cerro. A simple deduction method for modal logic. *Information Porcessing Letters*, 14(2), 1982.

[Gal86] J.H. Gallier. *Logic for Computer Science.* Harper & Row, 1986.

[Gen69] G. Gentzen. Investigations into logical deductions. In M.E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Co, 1969.

[GL85] G. Gottlob and A. Leitsch. On the efficiency of subsumption algorithms. *Journal of the Association for Computing Machinery*, 32-2:280–295, 1985.

[GNOP82] S. Greenbaum, A. Nagasaka, P. O'Rorke, and D.A. Plaisted. Comparison of natural deduction and locking resolution implementations. In D. Loveland, editor, *Proceedings of the 6th Conference on Automated Deduction*, pages 159–171. Springer Lecture Notes in Computer Science 138, 1982.

[Gre86] S. Greenbaum. *Input Transformation and Resolution Implementation Techniques for Theorem Proving in First-Order Logic.* PhD thesis, University of Illinois at Urbana-Champaign, September 1986.

[HK69] P. Hayes and R.A. Kowalski. Semantic trees in automatic theorem proving. *Machine Intelligece 5*, pages 181–201, 1969.

[HLO+80] L. Henshen, E. Lusk, R. Overbeek, B.T. Smith, R. Veroff, S. Winker, and L. Wos. challenge problem 1. *SIGART newsletter*, 72:30–31, July 1980.

[Hue76] G Huet. *Résolution d'équations dans les langages d'ordre 1,2,...,ω.* PhD thesis, Université Paris VII, 1976. Thèse d'état.

[JK90] J.P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. Rapport de recherche 561, L.R.I, Bât. 490. Université D'Orsay. 91405 Orsay, March 1990.

[JOR88] S. Jeannicot, L. Oxusoff, and A. Rauzy. Evaluation sémantique: une propriété de coupure pour rendre efficace la procédure de davis et putnam. *Revue d'Intelligence Artificielle*, 2(1):41–60, 1988.

[Kir88] C. Kirchner. Order-sorted equational unification. In *Proceedings of the 5th International Conference on Logic Programming*, August 1988.

[Kow75] R. Kowaslki. A proof procedure using connection graph. *Journal of the Association for Computing Machinery*, 22:572–595, 1975.

[KZ90] T. Kaufl and N. Zabel. The theorem prover of the program verifier tatzelwurm. In M.E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, pages 657–658. Lecture Notes in Artificial Intelligence 449, 1990.

[Lin90] C. Lingenfelder. *Transforming ans Structuring of computer Generated Proof.* PhD thesis, Fachbereich Informatik des Universität Kaiserslautern, 1990.

[LMO82] E.L. Lusk, W.W. McCune, and R.A Overbeek. Logic machine architecture: Kernel functions. logic machine architecture: Inference mechanisms. In D. Loveland, editor, *Proceedings of the 6th Conference on Automated Deduction*, pages 70–108. Springer Lecture Notes in Computer Science 138, 1982.

[LMO84] E.L. Lusk, W.W. McCune, and S.A. Overbeek. A portable environment for research in automated theorem proving. In R.E. Shostak, editor, *Proceedings of the 7th Conference on Automated Deduction.* Springer. Lecture Notes in Computer Science 170, 1984.

[Lov78] D.W. Loveland. *Automatic Theorem Proving: a logical basis.* North Holland, 1978.

[McC88] B. McCune. An indexing mechanism for finding more general formulas. *AAR newsletter*, 9:7–8, January 1988.

[MF86] D.A. Miller and A. Felty. An integration of resolution and natural deduction theorem proving. In *5th National Conference on Artificial Intelligence, Philadelphia USA*, pages 198–202, August 1986.

[Mil84] D.A. Miller. Expansion tree proofs and their conversion to natural deduction proofs. In R.E. Shostak, editor, *8th Conference on Automated Deduction, Napa, California*, pages 375–393. Springer-Verlag, Lecture Notes in Computer Science 170, 1984.

[Mil87] D.A. Miller. A compact representation of proofs. *Studia Logica*, XLVI(4):347–370, 1987.

[MM76] A. Martelli and H. Montanari. Unification in linear time and space: A structured presentation. Technical report b76-16, university of pisa, July 1976.

[MM82] A. Martelli and H. Montanari. An efficient unification algorithm. *ACM Transactions On Programming Languages and Systems*, 4-2:258–282, 1982.

[Mur82] M.V. Murray. Completely non clausal theorem proving. *Artificial Intelligence*, 18(1):67–85, January 1982.

[NM88] G. Nadathur and D. Miller. An overview of λprolog. In K. Bowen and R. Kowalski, editors, *Fifth International Conference and Symposium on Logic Programming.* MIT Press, 1988.

[Nol80] H Noll. A note on resolution: how to get rid of factoring without loosing completeness. In W. Bibel and R. Kowalski, editors, *5th Conference on Automated Deduction, Les Arcs, France*, pages 250–263. Springer-Verlag, Lecture Notes in Computer Science 87, 1980.

[Ohl88] H.J. Ohlbach. A resolution calculus for modal logics. In E. Lusk and R. Overbeek, editors, *9th Conference on Automated Deduction*, pages 500–516. Springer-Verlag, Lecture Notes in Computer Science 310, 1988.

[OS88] F. Oppacher and E. Suen. Harp: A tableau based theorem prover. *Journal of Automated Reasoning*, 4:69–100, 1988.

[Pfe84] F. Pfenning. Analytic and non-analytic proofs. In R.E. Shostak, editor, *8th Conference on Automated Deduction, Napa, California*, pages 394–413. Springer-Verlag, Lecture Notes in Computer Science 1970, 1984.

[PG86] D.A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[PN90] F. Pfenning and D. Nesmith. Presenting intuitive deductions via symmetric simplification. In M.E. Stickel, editor, *10th Conference on Automated Deduction, Kaiserslautern, RFA*, pages 336–350. Springer-Verlag, Lecture Notes in Artificial Intelligence 449, 1990.

[Pra65] D. Prawitz. *Natural Deduction.* Almqvist-Wiskell, 1965.

[PW78] M.S. Paterson and M.N Wegman. Linear unification. *Journal of Computer Systems and Science*, 16:158–167, 1978.

[Rob65] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery*, 32:23–41, 1965.

[Rus87] M. Rusinowitch. *Démonstration automatique par des techniques de réécriture.* PhD thesis, Nancy-1 France, November 1987. Thèse d'Etat — also available as textbook (Inter Editions, Paris 1989).

[Sie84] J. Siekmann. Universal unification. In R.E. Shostak, editor, *Proceedings of the 7th Conference on Automated Deduction. Napa. USA*, pages 1–42. Springer. Lecture Notes in Computer Science 170, 1984.

[Sko67] T. Skolem. Logico-combinatorial investigation in the satisfiability or provability of mathematical propositions: A simplified proof of a theorem by L Lowenheim and generalizations of the theorem. In J.V. Heijenoort, editor, *From Frege to Godel, a source book in mathematical logic, 1879-1931.* Harvard University Press, 1967.

[Sla72] J.R. Slagle. Automated theorem proving for theories with built-in theories of equality, partial order and sets. *Journal of the Association for Computing Machinery*, 19:120–135, 1972.

[Sla74] J.R. Slagle. Automated theorem proving for theories with simplifiers, commutativity and associativity. *Journal of the Association for Computing Machinery*, 21:622–642, 1974.

[Smu68] R.M. Smullyan. *First-Order Logic.* Springer-Verlag, New-York, 1968.

[SS85a] M. Schmidt-Schauss. A many sorted calculus with polymorphic functions based on resolution and paramodulation. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 84–87. W. Kaufmann, 1985.

[SS85b] M. Schmidt-Schauss. A many sorted calculus with polymorphic functions based on resolution and paramodulation. Interner bericht, Universität Kaiserslautern., West Germany., 1985.

[SS86] M. Schmidt-Schauss. Unification in many-sorted equational theories. In J.H. Siekmann, editor, *Proceedings of the 8th Conference on Automated Deduction. Oxford England*, pages 538–552. Springer. Lecture Notes in Computer Science 230, 1986.

[SS87] M. Schmidt-Schauss. *Computational Aspects of an Order Sorted Logic with Term Declarations.* PhD thesis, Universität Kaiserslautern. West Gemany, 1987.

[Sti82] M.E. Stickel. A nonclausal connection-graph resolution theorem-proving program. In *Proceedings of the AAAI-82 National Conference on Artificial Intelligence. Pittburgh, Pennsylvania*, pages 229–233, 1982.

[Sti85] M.E. Stickel. Automated deduction by theory resolution. *Journal of Automated Reasoning*, 1:333–355, 1985.

[Sti86a] M.E. Stickel. A prolog technology theorem prover: Implementation by an extended prolog compiler. In J.H. Siekmann, editor, *Proceedings of the 8th Conference on Automated Deduction*, pages 588–597. Springer Lecture Notes in Computer Science 230, 1986.

[Sti86b] M.E. Stickel. Schubert's steamroller problem: Formulation and solutions. *Journal of Automated Reasoning*, 2:89–101, 1986.

[Tar75] R.E. Tarjan. Efficiency of a good but not linear disjoint set union algorithm. *Journal of the Association for Computing Machinery*, 22:215–225, 1975.

[Tse68] G.S. Tseitin. On the complexity of derivation in propositional calculus. In A.O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, 1968.

[Wal83] C. Walther. A many sorted calculus based on resolution and paramodulation. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 882–891. W. Kaufmann, 1983.

[Win85] S. Winker. An evaluation of an implementation of qualified hyperresolution. *IEEE transaction on Computers*, C-25:835–843, 1985.

[Wos88] L. Wos. *Automated Reasoning: 33 Basic Research Problems.* Prentice-Hall, 1988.

[WR67] L. Wos and G. Robinson. The concept of demodulation in theorem proving. *Journal of the Association for Computing Machinery*, pages 698–709, 1967.

[WRC67] L. Wos, G. Robinson, and D. Carson. Efficiency and completeness of set of support strategy in theorem proving. *Journal of the Association for Computing Machinery*, 12:536–541, 1967.

[WRCS70] L. Wos, G. Robinson, D. Carson, and L. Shalla. Paramodulation and set of support. In *Symposium on Automatic Demonstration, Versailles France.*, pages 276–310. Lecture Notes in Mathematics 125, Springer, 1970.

Sommaire

Résumé

Dans une première partie, nous montrons qu'il est possible d'établir une correspondance "naturelle" entre les preuves en Déduction Natutrelle de la validité d'une formule et les réfutations par Résolution d'un ensemble de clauses obtenues en appliquant à la négation de cette formule une mise sous forme clausale non standard utilisant une technique de renommage.

En particulier, nous montrons qu'il est possible de simuler le fonctionnement d'un calcul des séquents proche de celui de Gentzen par la Résolution et nous montrons comment traduire des réfutations par Résolution en preuve en Déduction Naturelle. De plus, nous proposons plusieurs améliorations de cette mise sous forme clausale avec renommage permettant de faciliter la recherche d'une réfutation par Résolution.

Dans une deuxième partie, nous décrivons en détail des techniques permettant une mise en œuvre efficace de la Résolution avec sortes ordonnées ainsi qu'un principe d'indexation des clauses permettant de résoudre efficacement de nombreux problèmes-clés (tels ceux posés par la subsomption, l'utilisation de systèmes de réécriture ...). Ces algorithmes ont été utilisés dans l'implémentation d'un démonstrateur par Résolution que nous avons réalisé dans le cadre d'ATINF.

mots clés:

Déduction automatique , Déduction Naturelle, Résolution, Forme clausale, Renommage, Logiques avec sortes ordonnées.