Résolution au premier ordre

Jean Goubault-Larrecq

February 22, 1999

Nous avons déjà présenté la résolution comme une méthode de preuve automatique pour la logique propositionnelle classique. Bien qu'elle n'était pas l'une des méthodes les plus efficaces dans ce contexte, elle peut être optimisée de nombreuses façons. De plus, elle est utile pour présenter les outils fondamentaux dont nous avons besoin pour la recherche de preuve en logique classique du premier ordre sans que l'on ait à s'occuper de complications provenant de problèmes purement propositionnels.

1 Idées fondamentales

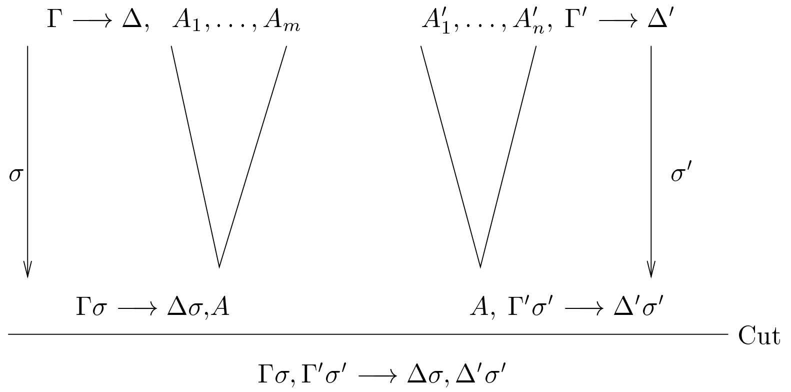
L'idée principale est due à J. Alan Robinson en 1965. Il y a deux ingrédients dans la méthode de résolution : la résolution propositionnelle, que nous avons déjà examinée, et l'unification, qui la rend utilisable lorsqu'on transpose la méthode au premier ordre.

Considérons une formule Φ du premier ordre, dont nous voulons savoir si elle est valide. Comme en résolution propositionnelle, nous la nions, et essayons d'en dériver une contradiction. Pour ce faire, nous skolémisons $\neg\Phi$ (rappelons que la forme de Skolem est insatisfiable si et seulement si $\neg\Phi$ l'est), et obtenons une formule de la forme $\forall x_1 \cdot \ldots \forall x_n \cdot \Psi$. Nous pouvons maintenant écrire Ψ en forme clausale, comme dans le cas propositionnel : Ψ est la conjonction de clauses C_1, …, C_m. Alors, par la construction de Herbrand, $\neg\Phi$ est insatisfiable si et seulement si la conjonction d'un nombre fini d'instances closes des clauses C_i, $1 \leq i \leq m$, est insatisfiable, c'est-à-dire *propositionnellement insatisfiable*, puisqu'il ne reste aucun quantificateur.

Ensuite, nous pouvons utiliser la résolution propositionnelle. En particulier, Φ est valide si et seulement s'il existe un ensemble fini S d'instances closes des clauses C_i tel que l'on peut dériver la clause vide $\longrightarrow$ à partir de S en n'utilisant que la règle Cut.

La vraie invention de Robinson a été de concevoir l'unification pour factoriser tous les aspects communs entre toutes les applications de Cut sur des instances closes différentes des mêmes clauses. Regarder la figure 1, où on a représenté deux clauses $C = \Gamma \longrightarrow \Delta, A_1, \ldots, A_m$ et $C' = A'_1, \ldots, A'_n, \Gamma' \longrightarrow \Delta'$, d'instances closes respectives $\Gamma\sigma \longrightarrow \Delta\sigma, A$ et $A, \Gamma'\sigma' \longrightarrow \Delta'\sigma'$, où $A = A_1\sigma = \ldots = A_m\sigma = A'_1\sigma' = \ldots = A'_n\sigma'$. Par la règle Cut propositionnelle, ces deux dernières clauses produisent la clause close $\Gamma\sigma, \Gamma'\sigma' \longrightarrow \Delta\sigma, \Delta'\sigma'$.

Maintenant supposons que A_1, …, A_m soient les seuls atomes à droite de C tels que $A_i\sigma = A$, et que A'_1, …, A'_n soient les seuls atomes à gauche tels que $A'_i\sigma' = A$. Toute coupure propositionnelle entre les clauses closes $C\sigma$ et $C'\sigma$ a la même structure que ci-dessus, quelles que soient les valeurs de σ et σ', du moment qu'elles identifient toutes $A_1\sigma$, …, $A_m\sigma$ à droite de C et $A'_1\sigma'$, …, $A'_n\sigma'$ à gauche de C'. Nous pouvons regrouper toutes ces déductions en une règle unique, la règle de résolution :



(where $A = A_1\sigma = \ldots = A_m\sigma = A'_1\sigma' = \ldots = A'_n\sigma'$)

Figure 1: Relèvement

- à partir des deux clauses $C = \Gamma \longrightarrow \Delta, A_1, \ldots, A_m$ et $C' = A'_1, \ldots, A'_n, \Gamma' \longrightarrow \Delta'$, déduire les clauses $\Gamma\sigma, \Gamma'\sigma' \longrightarrow \Delta\sigma, \Delta'\sigma'$, pour toutes substitutions σ et σ' qui rendent égaux exactement $A_1\sigma$, …, $A_m\sigma$ à droite de C et $A'_1\sigma'$, …, $A'_n\sigma'$ à gauche de C'.

Ceci ne semble pas meilleur que l'énumération de toutes les instances closes des clauses, suivie de la coupure. C'est en fait bien meilleur, parce que le processus d'unification, que nous rappelons en section 2, donne une représentation concise de l'ensemble des substitutions possibles σ et σ', comme l'ensemble des instances d'une substitution unique, appelée l'*unificateur le plus général* de A_1, …, A_m, et de versions renommées de A'_1, …, A'_n.

2 Unification

L'unification est le cas particulier où, étant donnés deux termes ou atomes s et t, nous voulons trouver l'ensemble des substitutions σ telles que $s\sigma = t\sigma$. (À partir de maintenant, nous ne considérerons que le cas des termes, car les atomes ont la même structure que les termes.)

Définition 1 (Unificateur) *Soient s et t deux termes. Une substitution σ est un* unificateur *de s et t si et seulement si $s\sigma = t\sigma$.*

Ce cas particulier est suffisant pour résoudre notre problème. Nous le montrons en deux lemmes, mais d'abord une définition :

Définition 2 (Renommage) *Une substitution ρ est un* renommage *si et seulement si elle envoie des variables vers des variables, et elle est bijective.*

Alors :

Lemme 3 (Unification avec renommage) *Soient s et t deux termes, et soit ρ un renommage envoyant les variables libres de t vers des variables non libres dans s.*

Alors les couples (σ, σ') de substitutions tels que $s\sigma = t\sigma'$ sont exactement les couples $(\sigma, \rho\sigma)$, où σ unifie s et $t\rho$.

Preuve : Exercice. □

Lemme 4 (Unification multiple) *Soient $t_1, \ldots, t_n$ n termes, $n \geq 2$. Il existe une substitution σ telle que $t_1\sigma = \ldots = t_n\sigma$ si et seulement s'il existe σ_1 telle que $t_1\sigma_1 = t_2\sigma_1$, et σ_2 telle que $t_2\sigma_1\sigma_2 = t_3\sigma_1\sigma_2$, et $\ldots$, et σ_{n-1} telle que $t_{n-1}\sigma_1 \ldots \sigma_{n-1} = t_n\sigma_1 \ldots \sigma_{n-1}$.*

Preuve : Exercice. □

Pour trouver deux substitutions σ et σ' telles que $A_1\sigma = \ldots = A_m\sigma = A'_1\sigma' = \ldots = A'_n\sigma'$, il suffit donc de renommer les A'_i's, $1 \leq i \leq n$, en $A'_i\rho$, où ρ envoie les variables de $\mathrm{fv}(A'_1) \cup \ldots \cup \mathrm{fv}(A'_n)$ vers $\mathcal{V} \setminus (\mathrm{fv}(A_1) \cup \ldots \cup \mathrm{fv}(A_m))$. Ensuite, par le lemme 3, il reste à unifier $A_1, \ldots, A_m, A'_1\rho, \ldots, A'_n\rho$. Par le lemme 4, nous pouvons le faire en unifiant les termes deux à deux, et en composant les substitutions au fur et à mesure.

Il reste à résoudre le problème de l'unification :

Étant donnés s et t, comment peut-on calculer l'ensemble des unificateurs de s et t?

Le résultat fondamental sur l'unification des termes du premier ordre est :

Théorème 5 (Unification) *Soient s et t deux termes. Alors soit il n'existe aucun unificateur de s et t, et nous disons alors que s et t ne sont pas unifiables; ou il existe un* unificateur le plus général *ou* mgu *("most general unifier" en anglais), c'est-à-dire une substitution σ telle que :*

- *σ est un unificateur de s et t, autrement dit, $s\sigma = t\sigma$;*
- *et tout unificateur σ' de s et t est une instance de σ, c'est-à-dire qu'il existe une substitution σ'' telle que $\sigma' = \sigma\sigma''$.*

Preuve : Omise (traité dans un autre cours du DEA). □

Le mgu, à strictement parler, peut ne pas être unique, car par exemple $[x/y]$ et $[y/x]$ sont toutes les deux des mgus des deux variables x et y. Mais on a une forme d'unicité à renommage près :

Lemme 6 (Unicité) *Soient s et t deux termes. Alors les mgus de s et t sont unique à renommage près, c'est-à-dire que, pour tout couple de mgus σ et σ' de s et t, il existe un renommage ρ tel que $\sigma = \sigma'\rho$ (et $\sigma' = \sigma\rho^{-1}$).*

Nous notons $mgu(s,t)$ le mgu (unique à renommage près) de s et t.

Preuve : Omise. □

Pour des raisons d'efficacité, nous représentons usuellement les unificateurs sous forme résolue comme :

Définition 7 (Forme triangulaire) *Une* forme triangulaire *est une suite finie $[t_1/x_1; t_2/x_2; \ldots; t_n/x_n]$ de liaisons terme/variable t_i/x_i. Nous disons qu'une forme triangulaire* représente *la substitution σ si et seulement si $\sigma = [t_1/x_1][t_2/x_2] \ldots [t_n/x_n]$*

Voir la figure 2 pour un algorithme d'unification retournant une forme triangulaire, dans le style de Martelli et Montanari [MM82]. L'algorithme fonctionne en transformant des couples (F, σ), où F est un ensemble ou un multi-ensemble de paires de termes (représentées comme des équations formelles non orientées entre termes) qui doivent être unifiées simultanément, et σ est la forme triangulaire du

(Delete)	$(F \cup \{t \doteq t'\}, \sigma)$	$\rightarrow (F, \sigma)$
	si $t = t'$	
(Decomp)	$(F \cup \{t \doteq t'\}, \sigma)$	$\rightarrow (F \cup \{t_1 \doteq t'_1, \ldots, t_m \doteq t'_m\}, \sigma)$
	si $t \neq t'$ et	
	$t = f(t_1, \ldots, t_m)$, $t' = f'(t'_1, \ldots, t'_{m'})$,	
	avec $f = f'$ et $m = m'$	
(Bind)	$(F \cup \{x \doteq t\}, [t_1/x_1; \ldots; t_n/x_n])$	$\rightarrow (F, [t/x; t_1/x_1; \ldots; t_n/x_n])$
	si $x \neq t$ et $x \notin \{x_1, \ldots, x_n\} \cup \mathrm{fv}(t[t_1/x_1] \ldots [t_n/x_n])$,	
(Merge)	$(F \cup \{x_i \doteq t\}, [t_1/x_1; \ldots; t_n/x_n])$	$\rightarrow (F \cup \{t_i \doteq t\}, [t_1/x_1; \ldots; t_n/x_n])$
	si $x_i \neq t$, $1 \leq i \leq n$	

Figure 2: Règles d'unification

mgu en cours de construction. En pratique, les multi-ensembles sont codés sous forme de listes, où l'ordre des éléments est jugé sans importance.

Pour commencer et unifier s et t, on lance l'algorithme sur $(\{s \doteq t\}, [])$. Ensuite, on choisit une règle à appliquer selon une stratégie donnée jusqu'à ce que plus aucune règle ne soit applicable. Une stratégie possible est la suivante : si F est vide, alors σ est le mgu désiré; sinon, choisir une équation $t \doteq t'$ à résoudre dans F: si t égale t' (règle **(Delete)**), elle est déjà résolue, et on l'enlève simplement de F; si t et t' sont des applications du même symbole de fonction, alors seule la règle **(Decomp)** s'applique, et on remplace $t \doteq t'$ par les équations indiquées; si t ou t' est une variable, disons t, alors soit t n'est pas encore liée, c'est-à-dire qu'elle n'est pas dans le domaine de σ, et on doit appliquer la règle **(Bind)** pour lier t à t'; ou elle est liée, et on doit remplacer l'équation $t \doteq t'$ par une nouvelle équation où t a été remplacée par sa valeur par la règle **(Merge)**. Si aucune règle ne s'applique et F esta non vide, alors l'unification échoue.

L'algorithme qui en résulte prend toujours un temps polynomial en les tailles des termes de départ à unifier, et même un temps quasi-linéaire à condition de faire le *test d'occurrence* ($x \notin \{x_1, \ldots, x_n\} \cup \mathrm{fv}(t[t_1/x_1] \ldots [t_n/x_n])$; "occurs-check" en anglais) de la règle **(Bind)** de façon intelligente. Voir [JK90] pour un panorama sur l'unification, et plus.

3 Résolution

Revenons à la règle de résolution. Rappelons que nous l'avons énoncée comme suit :

- à partir des deux clauses $C = \Gamma \longrightarrow \Delta, A_1, \ldots, A_m$ et $C' = A'_1, \ldots, A'_n, \Gamma' \longrightarrow \Delta'$, déduire les clauses $\Gamma\sigma, \Gamma'\sigma' \longrightarrow \Delta\sigma, \Delta'\sigma'$, pour toutes substitutions σ et σ' qui rendent égaux exactement $A_1\sigma$, …, $A_m\sigma$ à droite de C et $A'_1\sigma'$, …, $A'_n\sigma'$ à gauche de C'.

Comme le mgu est unique, nous savons que toutes les clauses résultat $\Gamma\sigma, \Gamma'\sigma' \longrightarrow \Delta\sigma, \Delta'\sigma'$ sont en fait des instances d'une clause unique : $\Gamma\sigma, \Gamma'\rho\sigma \longrightarrow \Delta\sigma, \Delta'\rho\sigma$, où ρ renomme les variables libres de la seconde clause en de nouvelles variables, non libres dans la première clause, et σ est le mgu de A_1, …, A_m, $A'_1\rho$, …, $A'_n\rho$.

Ceci nous mène au principe de résolution, comme il est réellement utilisé en recherche de preuve par résolution :

Définition 8 (Résolution) *La* règle de résolution binaire *est :*

$$\frac{\Gamma \longrightarrow \Delta, A \qquad A', \Gamma' \longrightarrow \Delta'}{\Gamma\sigma, \Gamma'\sigma \longrightarrow \Delta\sigma, \Delta'\sigma} \sigma = mgu(A, A')$$

où nous supposons que les deux clauses $\Gamma \longrightarrow \Delta, A$ *et* $A', \Gamma' \longrightarrow \Delta'$ *sont renommées de sorte qu'elles n'ont aucune variable libre en commun. La conclusion de la règle est appelée le* résolvant binaire *des deux clauses, les premisses sont appelées les* clauses parentes*, et* A *(resp.* $\neg A'$*) est le littéral sur lequel on résout dans la première clause (resp. second clause).*

Nous disons que les littéraux A *et* $\neg A'$ *sont* complementaires *à condition que* A *s'unifie avec* A'*.*

La règle de factorisation *est la règle double :*

$$\frac{\Gamma \longrightarrow \Delta, A, B}{\Gamma\sigma \longrightarrow \Delta\sigma, A\sigma} \sigma = mgu(A, B) \qquad \frac{A, B, \Gamma \longrightarrow \Delta}{A\sigma, \Gamma\sigma \longrightarrow \Delta\sigma} \sigma = mgu(A, B)$$

La conclusion de la règle est appelée un facteur binaire *de la clause en prémisse.*

Un facteur *d'une clause est soit la clause elle-même, soit un facteur binaire d'un de ses facteurs. Un* facteur propre *d'une clause est un facteur d'un de ses facteurs binaires.*

Un résolvant *de deux clauses est un résolvant binaire de facteurs de chaque clause, non nécessairement propres.*

Une dérivation par résolution *est un arbre inversé dont les nœuds sont étiquetés par des clauses, les feuilles sont étiquetées par des clauses de l'ensemble initial de clauses, et les nœuds internes ont comme prédécesseurs leurs clauses parentes.*

Une réfutation par résolution *est une dérivation se terminant sur la clause vide, autrement dit une dérivation dont la racine est étiquetée par* $\longrightarrow$*.*

Remarquer que l'utilisation d'une règle séparée de factorisation code des unifications multiples.

Théorème 9 (Correction, complétude) *Soit* S_0 *une ensemble de clauses, et* S_n *la suite des ensembles de clauses définie par* S_{n+1} *égale l'ensemble de tous les résolvants de clauses dans* S_n *(*S_n *est appelé l'*ensemble de niveau n*).*

Alors la résolution est une procédure de recherche de réfutation correcte *et* complète*, autrement dit, l'ensemble* S_0*, vu comme une conjonction universellement quantifiée de clauses disjonctives, est insatisfiable si et seulement s'il existe* $n \in \mathbb{N}$ *tel que* S_n*,* $n \in \mathbb{N}$ *contienne la clause vide.*

Preuve : La correction vient, comme dans le cas propositionnel, du fait que la conjonction de deux clauses quelconques (ici, universellement quantifiées) implique leurs résolvants.

La complétude est le fait que si S_0 est insatisfiable, alors il existe un ensemble de niveau qui contient la clause vide. Par les observations de la section 1, si S_0 est insatisfiable, alors l'ensemble des instances closes de clauses dans $\bigcup_{n\in\mathbb{N}} S_n$ contient la clause vide. (Ceci vient de la complétude de la résolution propositionnelle et du théorème de Herbrand.) Maintenant, la seule clause qui a la clause vide comme instance est la clause vide elle-même. Donc la clause vide est dans $\bigcup_{n\in\mathbb{N}} S_n$, donc dans un des S_n. $\square$

On peut avoir l'impression que la factorisation est inutile, et que les facteurs seront fabriqués par unification lors de la résolution binaire. Ceci simplifierait en effet le codage de la résolution. Malheureusement, les facteurs sont indispensables. Par exemple, l'ensemble des deux clauses :

$$\longrightarrow P(x,a), P(a,x)$$
$$P(y,a), P(a,y) \longrightarrow$$

est insatisfiable, parce que $\longrightarrow P(a,a)$ est une facteur de la première clause, $P(a,a) \longrightarrow$ un facteur de la seconde, et la clause vide en est un résolvant. Cependant, les seules clauses que nous pouvons en extraire par résolution binaire uniquement sont $P(a,x) \longrightarrow P(a,x)$, $P(y,a) \longrightarrow P(y,a)$ et $P(a,a) \longrightarrow P(a,a)$, mais pas la clause vide : la résolution binaire seule est incomplète.

Voir [CL73] pour une autre preuve de la complétude de la résolution. Chang et Lee procèdent par des arguments purement sémantiques, en utilisant des *arbres de Herbrand*, que nous avons déjà utilisés dans la preuve du théorème de Herbrand.

4 Optimisations

La résolution est assez inefficace au niveau propositionnel. Si nous séparons la résolution au premier ordre en ses deux composantes, la résolution propositionnelle et le relèvement au premier ordre par unification, alors seule la première composante reste à améliorer. Heureusement, on peut raffiner la résolution de nombreuses façons pour la rendre plus efficace. Certaines des idées viennent de la procédure de Davis-Putnam, d'autres tentent de restreindre les dérivations par résolution à des dérivations presque canoniques pour éviter les redondances.

4.1 Stratégies de sélection de clauses

La première chose que nous devons examiner est le processus de sélection des clauses. Le seul que nous avons présenté pour l'instant, la recherche par *saturation de niveaux*, est complète, mais produit de nombreuses clauses inutiles.

Nous diminuerons fortement les chances de tomber sur les configurations les pires en utilisant diverses optimisations, voir les sections 4.2 ou 4.4. Il nous faudra y adjoindre des stratégies plus fines d'exploration de l'espace des dérivations par résolution.

La recherche par saturation de niveaux produit toutes les dérivations possibles en parallèle. D'abord toutes les dérivations de profondeur 1, qui fabriquent les clauses de S_1, ensuite les dérivations de profondeur 2, qui fabriquent les clauses de S_2, et ainsi de suite. Mais l'on n'est pas obligé de suivre une telle stratégie. En fait, tout ce que nous voulons, c'est explorer l'espace de toutes les dérivations de sorte qu'aucune étape de résolution ne soit remise à plus tard indéfiniment longtemps. Il s'agit d'une condition d'équité ("fairness" en anglais) :

Définition 10 (Équité) *Une stratégie de sélection de clauses est dite* équitable *si et seulement s'il n'existe pas de paire de clauses ignorées infiniment longtemps par la stratégie. (De façon équivalente, si l'ensemble des résolvants produits par la stratégie est exactement l'union de tous les ensembles de niveaux.)*

Il est alors clair que n'importe quelle stratégie équitable, et pas seulement la recherche par saturation de niveaux, préserve la complétude de la résolution. La recherche par saturation de niveaux est coûteuse en termes tant d'espace que de temps. La recherche en profondeur d'abord utilise la mémoire à meilleur escient : choisir une paire de clauses, résoudre, ajouter le résolvant à l'ensemble des clauses, et continuer; ensuite, rebrousser chemin ("backtrack" en anglais) et choisir une autre paire de clauses à résoudre si l'on réalise plus tard que cette paire ne mène pas à une réfutation. Le nombre de clauses engendrées que nous devons garder en mémoire au niveau k est alors $O(k)$, c'est-à-dire bien moins que dans la recherche par saturation de niveaux, où on doit en garder une exponentielle en k.

Le problème est que la recherche en profondeur d'abord n'est pas équitable. Mais on peut y remédier en utilisant l'astuce suivante, appelée *approfondissement itéré* ("iterative deepening" en anglais) : on se fixe une limite $k \in \mathbb{N}$, puis on effectue une recherche en profondeur comme ci-dessus, mais on rebrousse chemin si la clause vide n'a pas été produite en moins de k étapes de résolution; si aucune réfutation n'a pu être obtenue en moins de k étapes de résolution, nous le découvrirons en temps fini; si c'est la cas, augmenter k et recommencer. Ce faisant, nous reproduirons tous les calculs que nous avons déjà effectués, fabriquant des résolvants à des niveaux plus petits que l'ancienne valeur de k, mais ceci prend usuellement moins de temps que l'exploration de tous les nouveaux résolvants au niveau $k+1$. (Par exemple, supposons que nous passions un temps $O(2^k)$ pour explorer les clauses dans S_k, alors pour explorer celles de S_{k+1}, nous y passerons un temps $O(2^{k+1})$, dont seule la moitié est consacrée à la réexploration d'anciennes clauses.)

Nous allons maintenant chercher à réduire l'ensemble des clauses qui valent la peine d'être produites. Ceci réduira l'espace des dérivations par résolution qui nous intéresseront, mais pas suffisamment pour que cet espace devienne fini. Dans tous les cas, nous aurons besoin de stratégies équitables de sélection de clauses.

4.2 Stratégies d'effacement

Les *stratégies d'effacament* ("deletion strategies" en anglais) éliminent des clauses inutiles qui, si on les conservait, pollueraient le processus de recherche de preuve en augmentant artificiellement la taille de l'ensemble de clauses.

4.2.1 Élimination des clauses subsumées

Définition 11 (Subsomption) *Soit $C = \Gamma \longrightarrow \Delta$ et C' une autre clause. Nous disons que C* subsume *C', et nous notons $C \geq C'$, si et seulement si C' est de la forme $\Gamma', \Gamma\sigma \longrightarrow \Delta\sigma, \Delta'$ pour une certaine substitution σ.*

Autrement dit, C subsume toutes les clauses obtenues en affaiblissant une de ses instances. Considérons C et C' comme des disjonctions universellement quantifiées de littéraux : sémantiquement, si C subsume C', alors C implique C'.

Les clauses subsumées sont normalement inutiles, à cause de :

Lemme 12 *Soient C_1 et C'_1 deux clauses telles que C_1 subsume C'_1. Pour tout résolvant C' entre C'_1 et C_2, C' est subsumée par C_1 ou par un résolvant de C_1 et C_2.*

De plus, l'unique clause qui subsume la clause vide est la clause vide.

Preuve : Notons C_1 $\Gamma_1 \longrightarrow \Delta_1$, et C'_1 $\Gamma'_1, \Gamma_1\sigma_1 \longrightarrow \Delta_1\sigma_1, \Delta'_1$. Si C' est un facteur de C'_1, il est clairement subsumé par C_1. Nous vérifions maintenant le lemme sur les résolvants binaires. Supposons que nous ayons obtenu C' par résolution binaire sur un atome A'_1 à droite de C'_1 et un atome A_2 à gauche de C_2. (Le cas symétrique est similaire.)

Premier cas : $A'_1 \in \Delta'_1$. Soit $\Delta''_1 = \Delta'_1 \setminus \{A'_1\}$, $C_2 = A_2, \Gamma_2 \longrightarrow \Delta_2$, et $\sigma' = mgu(A'_1, A_2)$. Alors $C'_1 = \Gamma'_1, \Gamma_1\sigma_1 \longrightarrow \Delta_1\sigma_1, \Delta''_1, A'_1$, $C' = \Gamma_2\sigma', \Gamma'_1\sigma', \Gamma_1\sigma_1\sigma' \longrightarrow \Delta_1\sigma_1\sigma', \Delta''_1\sigma', \Delta_2\sigma'$. Donc $C' \leq C_1$.

Second cas : $A'_1 \in \Delta_1\sigma_1$. Soit A'_1 égale $A''_1\sigma_1$, $\Delta''_1 = \Delta_1 \setminus \{A''_1\}$, $C_2 = A_2, \Gamma_2 \longrightarrow \Delta_2$, et $\sigma' = mgu(A''_1\sigma_1, A_2)$. Comme $A''_1\sigma_1\sigma' = A_2\sigma'$, et comme $\mathrm{fv}(A''_1) \cap \mathrm{fv}(A_2) = \emptyset$, la substitution $\sigma_1\sigma' \cup \sigma'$ est un unificateur de A''_1 et A_2, disons $\sigma\theta$, où $\sigma = mgu(A''_1, A_2)$. Alors nous pouvons résoudre C_1 avec C_2, sur

A_1'' et A_2 respectivement, ce qui produit $C = \Gamma_1\sigma, \Gamma_2\sigma \longrightarrow \Delta_1''\sigma, \Delta_2\sigma$. Puisque $C' = \Gamma_2\sigma\theta, \Gamma_1'\sigma\theta, \Gamma_1\sigma\theta \longrightarrow \Delta_1''\sigma\theta, \Delta_1'\sigma\theta, \Delta_2\sigma\theta$, C subsume C'.

Finalement, si $\Gamma \longrightarrow \Delta$ subsume la clause vide, alors Γ et Δ doivent être vides, autrement dit seule la clause vide subsume la clause vide. $\square$

Corollaire 13 *Soit S et S' deux ensembles de clauses. Supposons que toute clause C de S est dans S' ou est subsumée par une clause de S'.*

Si la clause vide est dérivable à partir de S en k étapes de résolution, alors elle est dérivable à partir de S' en au plus k étapes de résolution.

Preuve : Par récurrence sur k. Si $k = 0$, alors la clause vide est dans S, donc par hypothèse elle est dans S' ou elle est subsumée par une clause de S'; dans chaque cas, la clause vide doit déjà être dans S'. Sinon, supposons que le résultat soit valide pour $k - 1$ étapes, et que nous puissions dériver la clause vide de S en k étapes, $k > 0$. Examinons la première étape de résolution dans la dérivation : nous résolvons deux clauses C_1 et C_2 pour produire le résolvant C.

Par hypothèse, C_1 est subsumée par une clause C_1' de S' (éventuellement C_1 elle-même), et C_2 est subsumée par une clause C_2' de S'. Par le lemme 12, C est subsumée par une clause C', qui peut être C_1', C_2' ou un résolvant de C_1' et C_2'. Alors, $S \cup \{C\}$ et $S' \cup \{C'\}$ sont deux ensembles de clauses vérifiant les conditions du théorème, et nous pouvons dériver la clause vide en $k - 1$ étapes de $S \cup \{C\}$. Par hypothèse de récurrence, nous pouvons dériver la clause vide en au plus $k - 1$ étapes de $S' \cup \{C'\}$, donc en au plus k étapes de S'. $\square$

Nous pouvons donc éliminer les clauses subsumées comme suit. Si S est l'ensemble courant de clauses, nous enlevons de S toute clause subsumée par une autre clause de S tant que c'est possible. Ceci fournit un ensemble minimal S' comme dans le corollaire. Puis nous produisons un nouveau résolvant, et effaçons les clauses subsumées de l'ensemble résultant, et ainsi de suite. C'est complet parce que si nous sommes à une distance de k résolvants de la clause vide, le fait d'effacer des clauses subsumées nous laisse à au plus k résolvants de la clause vide, et si nous calculons le bon résolvant, nous serons alors à au plus $k - 1$ résolvants de la clause vide, et ainsi de suite.

Une façon erronée d'effacer des clauses subsumées est de repérer toutes les clauses subsumées dans S et de les effacer en masse. En effet, le préordre de subsomption peut avoir des cycles lorsque différentes versions renommées de la même clause sont présentes dans S. Considérons l'exemple suivant : $S = \{\longrightarrow P(x), \longrightarrow P(x'), P(y) \longrightarrow\}$. S est insatisfiable, mais les deux premières clauses se subsument l'une l'autre; alors que $\{\longrightarrow P(x'), P(y) \longrightarrow\}$ et $\{\longrightarrow P(x), P(y) \longrightarrow\}$ sont tous les deux des sous-ensembles insatisfiables de S (des sous-ensembles S' comme dans le corollaire 13), l'élimination de toutes les clauses subsumées de S ne nous laisse que la clause $P(y) \longrightarrow$, qui est satisfiable.

On utilise la subsomption essentiellement de deux façons. La *subsomption avant* ("forward subsumption") consiste en : chaque fois que nous fabriquons des résolvants à partir de clauses dans l'ensemble courant S, nous ne gardons que les résolvants qui ne sont pas subsumés par aucune clause de S. D'un autre côté, la *subsomption arrière* ("backward subsumption") consiste à faire que quand nous produisons des résolvants C à partir de clauses dans l'ensemble courant S, nous effaçons de S toutes les clauses subsumées par C. On peut utiliser une combinaison des deux, par exemple la subsomption avant, puis arrière si le résolvant a survécu à la subsomption avant.

Remarquons aussi que la subsomption avant ne doit être utilisée que sur les résolvants, pas sur toutes les clauses, et en particulier pas sur les facteurs. En effet, un facteur d'une clause C est toujours subsumé par C : si la subsomption avant sur les facteurs préservait la complétude, alors on n'aurait pas besoin de la factorisation pour assurer la complétude de la méthode.

Calculer si une clause C est subsumée par une autre clause C' est difficile. En fait, le problème est NP-complet. En effet, bien que déterminer si un atome B est de la forme $A\sigma$ soit facile (c'est du *filtrage* ("matching"), et ça peut se tester en temps polynomial par une modification simple de l'algorithme d'unification, où le test d'occurrence n'est plus nécessaire), au contraire pour décider la subsomption, nous devons d'abord deviner, ou trouver en rebroussant éventuellement chemin, une fonction des littéraux A' de C' vers le littéraux A de C, telle que $A = A'\sigma$ pour une certaine substitution commune σ. On peut améliorer cette procédure en remarquant que A' ne peut s'instancier en A que s'ils ont le même symbole de prédicat au sommet, de sorte que nous n'avons besoin d'explorer que les atomes A qui ont le même symbole de prédicat que A', pour tout A'. On peut encore faire mieux en considérant les symboles de fonction apparaissant aux diverses positions dans chaque atome, mais c'est plus compliqué, et nous ne le décrirons pas ici.

En pratique, le problème est encore plus difficile parce que ce qui nous intéresse n'est pas de savoir si C est subsumée par une clause donnée C', mais par une clause inconnue dans l'ensemble, éventuellement très grand, de clauses S. Il existe des méthodes efficaces pour résoudre ce problème, qui fonctionnent en général très rapidement. Mentionnons une solution, analogue aux *réseaux de discrimination* ("discrimination nets") du prouveur Otter de Larry Wos, et utilisée par Andrei Voronkov dans son prouveur Vampire. On compile chaque atome, puis chaque clause, sous forme d'une suite d'instructions d'une machine abstraite (qui a des instructions qui empilent et dépilent des termes sur une pile, qui testent l'égalité de termes, qui lient des variables à des termes, et ainsi de suite), dont le rôle est de tester si un atome ou une clause donné subsume un atome ou une clause donné en argument. Comme ces programmes sont des chaînes de byte-codes, on peut les partager via leurs préfixes communs, en en créant des nœuds de choix, ce qui fait de l'ensemble de programmes un arbre. Le partage des préfixes de ces programmes fait que le programme arborescent qui en résulte effectue les tests de subsomption sur de nombreuses clauses en parallèle en général [Vor94].

4.2.2 Élimination de tautologies

La stratégie d'effacement la plus simple est l'*élimination de tautologies*. Une clause tautologique est une clause de la forme $\Gamma, A \longrightarrow A, \Delta$ pour un certain atome A. (Si nous voyons les clauses comme des disjonctions atomiques, avoir à la fois A et $\neg A$ est en effet la seule façon d'en faire une tautologie.) Éliminer les tautologies signifie que chaque fois que nous produisons un résolvant, nous ne l'ajoutons à l'ensemble courant de clauses que s'il n'est pas une tautologie (et qu'il n'est pas subsumé par les clauses précédentes).

Intuitivement, les tautologies n'apportent rien à la découverte d'une preuve, parce qu'elles sont toujours vraies. Ce n'est pas une justification, cependant. Mais l'élimination des tautologies est justifiée par :

Lemme 14 *Soit C une clause tautologique. Alors tout résolvant de C avec n'importe quelle clause C' est une tautologie ou est subsumée par C'.*

Preuve : Tout facteur de C' est subsumé par C. Nous examinons le cas des résolvants binaires entre C et C'. Soit C égale $A, \Gamma \longrightarrow \Delta, A$. Si nous ne résolvons pas sur A, alors le résultat est encore une tautologie. Sinon, posons C' égale $A, \Gamma' \longrightarrow \Delta'$ (le cas symétrique est similaire), alors le résolvant est $A, \Gamma, \Gamma' \longrightarrow \Delta', \Delta$, qui est subsumé par C. □

De plus, aucune tautologie ne peut être la clause vide. Donc la génération de nouvelles tautologies est inutile, ainsi que celle des clauses subsumées.

L'élimination des tautologies en résolution au premier ordre est similaire à l'élimination de tautologies dans la méthode de Davis-Putnam.

4.2.3 Élimination de clauses pures

Une autre simplification que nous pouvons adapter de la méthode de Davis-Putnam au cas de la résolution au premier ordre est l'élimination des clauses pures.

Définition 15 *Soit S un ensemble de clauses.*

Une clause C est dite pure *dans S si et seulement si elle contient un littéral pur, autrement dit elle est de la forme $\Gamma \longrightarrow \Delta, A$, (resp. $A, \Gamma \longrightarrow \Delta$) où A ne s'unifie avec aucun atome à gauche (resp. à droite) d'aucune clause dans S. Si c'est le cas, nous disons que A (resp. $\neg A$) est un littéral pur dans la clause C.*

Nous pouvons toujours éliminer les clauses pures, qu'elles viennent d'être produites ou qu'elles soient déjà présentes dans l'ensemble courant de clauses S, parce que :

Lemme 16 *Tout résolvant d'une clause pure avec n'importe quelle clause est pure.*

Preuve : Tout facteur d'une clause pure est clairement encore pure. Ensuite, toute résolution binaire entre une clause où A (resp. $\neg A$) est pur avec une autre clause doit opérer sur un autre littéral B. Le résolvant binaire produit doit donc contenir le littéral A (resp. $\neg A$), et est donc pur. $\square$

Comme aucune clause pure ne peut être vide, nous concluons que les clauses pures représentent des impasses dans la recherche de la clause vide. Nous pouvons donc éliminer les clauses pures sans sacrifier la complétude.

En fait, nous pouvons combiner toutes les stratégies d'effacement que nous avons mentionnées (la subsomption, l'élimination de tautologies, l'élimination de clauses pures) tout en conservant la complétude de la résolution. Regardons maintenant ce que nous avons au niveau propositionnel (où la factorisation n'est pas nécessaire) : la résolution binaire est similaire à une étape de la règle de séparation ("splitting") de Davis et Putnam (si l'on fait toutes les résolutions possibles sur un atome donné A, on produit en fait toutes les clauses de la règle de séparation — nous éclaircirons ce point en section 4.3); l'élimination de tautologies, resp. de littéraux purs, correspondent directement aux opérations des mêmes noms dans la méthode de Davis-Putnam. Les seules différences sont : en résolution propositionnelle, la subsomption avant revient à ne pas fabriquer d'affaiblissements de clauses précédentes, ce qui généralise la mémoïsation des clauses précédemment engendrées dont nous avions besoin pour faire terminer la résolution propositionnelle; le fait de produire des résolvants est en général moins efficace que d'utiliser la règle de séparation de la méthode de Davis-Putnam; et nous n'avons pas l'équivalent de la règle des clauses unitaires en résolution.

4.2.4 Résolution unitaire

Pour faire un parallèle avec la règle des clauses unitaires de la procédure de Davis-Putnam procedure, définissons un cas particulier de la règle de résolution, la *résolution unitaire* :

Définition 17 *Une* clause unitaire *est une clause de la forme $\longrightarrow A$ ou $A \longrightarrow$, où A est un atome.*

La résolution unitaire *est la variante de la résolution où la résolution binaire est contrainte de sorte qu'au moins une des clauses parentes soit unitaire.*

En d'autres termes, dans la résolution unitaire, pour appliquer la résolution binaire sur des clauses non unitaires, nous devons d'abord factoriser au moins une des clauses pour en faire une clause unitaire.

Malheureusement, la résolution unitaire n'est *pas* complète. Par exemple, l'ensemble de clauses propositionnelles :

$$\begin{array}{l} \longrightarrow A, B \\ A \longrightarrow B \\ A, B \longrightarrow \\ B \longrightarrow A \end{array}$$

est insatisfiable, mais on ne peut pas le montrer en n'utilisant que la résolution unitaire, parce qu'il ne contient pas de clause unitaire.

La résolution unitaire demande bien plus que l'application de la règle des clauses unitaires dans la méthode de Davis-Putnam : cette dernière règle était appliquée dès que possible, mais pas à l'exclusion des autres règles. Nous pourrions donc vouloir assouplir la contrainte sur la résolution unitaire comme suit : utiliser la résolution unitaire dès que possible, sinon utiliser la résolution générale. Mais ceci aussi est incomplet, autrement dit cette stratégie n'est pas équitable : considérons l'ensemble des clauses ci-dessus, plus :

$$\begin{array}{l} \longrightarrow P(a) \\ P(x) \longrightarrow P(f(x)) \end{array}$$

L'ensemble total est encore insatisfiable, mais on pourra appliquer la résolution unitaire indéfiniment, et on n'obtiendra que des clauses de la forme $\longrightarrow P(f^k(a))$, $k \geq 1$, mais jamais la clause vide.

Un cas particulier important est cependant :

Définition 18 (Horn) *Une* clause de Horn *est une clause qui a au plus un atome sur le côté droit (c'est-à-dire un séquent intuitionniste atomique).*

Un ensemble de Horn *est un ensemble de clauses de Horn.*

Lemme 19 *La résolution unitaire est complète pour les ensembles de Horn.*

Preuve : Exercice. (Le montrer d'abord dans le cas propositionnel; observer que la règle d'élimination de clauses unitaires par la méthode de Davis-Putnam est complète à elle seule dans le cas présent, et traduire toute dérivation de la clause vide ainsi obtenue en une preuve par résolution unitaire.) □

4.3 Stratégies ordonnées

Dans la méthode de Davis-Putnam, la séparation élimine un atome A en réécrivant l'ensemble S de clauses en deux ensembles $S[\mathbf{T}/A]$ et $S[\mathbf{F}/A]$, où A n'apparaît plus. Ce processus élimine les atomes un à un, jusqu'à ce qu'il n'en reste plus. La règle de résolution élimine aussi un atome, mais seulement entre deux clauses. Un problème majeur, alors, est la nature hautement non déterministe de ce processus d'élimination.

Considérons pas exemple les quatre clauses :

$$\begin{array}{l} \longrightarrow A, B \\ A \longrightarrow B \\ B \longrightarrow A \\ A, B \longrightarrow \end{array}$$

Nous pouvons choisir de résoudre la plupart des paires de clauses soit sur A soit sur B. Si nous choisissons de résoudre sur A, les seuls choix qui ne produisent pas

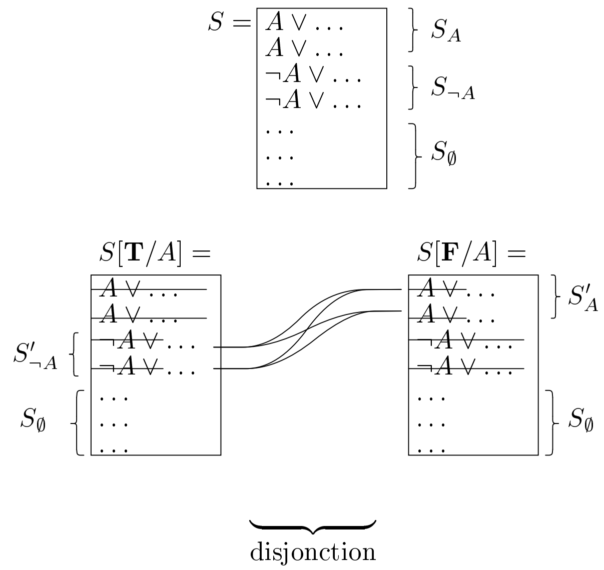


Figure 3: Séparation simulée par la résolution

de tautologies sont entre la première et la deuxième clause, ce qui donne $\longrightarrow B$, ou entre la troisième et la quatrième, ce qui donne $B \longrightarrow$. Alors, si on choisit de résoudre sur B entre ces deux dernières clauses, nous obtenons $\longrightarrow$. Nous aurions pu résoudre sur B d'abord, puis sur A, et nous aurions obtenu le même résultat : c'est parce que, pour obtenir la clause vide, nous devons éliminer tous les atomes des clauses par résolution, et nous pouvons le faire dans n'importe quel ordre.

En bref, il n'y a pas vraiment de non-déterminisme dans l'ordre dans lequel nous éliminons les atomes par la règle de résolution, comme tous les ordres doivent donner le même résultat — la dérivabilité ou l'indérivabilité de la clause vide. (Il s'agit d'un argument de *confluence*.) Pour éliminer cette redondance, une solution est de fixer un ordre sur les atomes à l'avance, et de ne permettre à la résolution de ne porter que sur les atomes maximaux dans l'ordre.

La raison pour laquelle cela fonctionne est le fait qu'au niveau propositionnel, la séparation sur A (l'atome maximal) consiste à calculer tous les résolvants de clauses sur A. Regarder la figure 3 : écrivons S comme l'union de trois ensembles : l'ensemble S_A des clauses de la forme $A \vee C$, autrement dit les séquents avec A à droite; l'ensemble $S_{\neg A}$ des clauses de la forme $\neg A \vee C$, autrement dit les séquents avec A à gauche; et l'ensemble $S_\emptyset$ de toutes les clauses où A et $\neg A$ n'apparaissent pas. Nous supposons que les tautologies ont été éliminées, et donc ces trois ensembles sont disjoints. Soit S'_A l'ensemble S_A d'où A a été effacé, et de même pour $S'_{\neg A}$ et $S_{\neg A}$.

Observer que $S[\mathbf{T}/A]$ est précisément l'ensemble des clauses de $S'_{\neg A}$ et $S_\emptyset$. Intuitivement, $S[\mathbf{T}/A] = S'_{\neg A} \wedge S_\emptyset$; et de même, $S[\mathbf{F}/A] = S'_A \wedge S_\emptyset$. La méthode de Davis-Putnam cherche une interprétation satisfaisant $S[\mathbf{T}/A]$ ou $S[\mathbf{F}/A]$. C'est une façon de chercher une interprétation satisfaisant $S[\mathbf{T}/A] \vee S[\mathbf{F}/A]$. Ce dernier est, intuitivement, $S_\emptyset \wedge (S_A \vee S_{\neg A})$. Et $S_A \vee S_{\neg A}$ est juste, par distributivité, la même chose que l'ensemble des résolvants entre clauses de S sur A et $\neg A$.

Au niveau propositionnel, nous pouvons éliminer les atomes dans n'importe quel ordre. Autrement dit, nous pouvons nous donner un ordre sur les atomes, et toujours éliminer le plus grand. Tous les ordres ne fonctionnent pas au premier ordre, cependant, c'est pourquoi nous définissons :

Définition 20 (Stabilité) *Un ordre strict $>$ sur les atomes est dit* stable *si et seulement si pour tous atomes A, B, et pour toute substitution σ, $A > B$ implique $A\sigma > B\sigma$.*

Rappelons qu'un ordre strict est une relation irréflexive et transitive. Les ordres stricts stables sont aussi appelés *A-ordres* ("A-orderings") dans la littérature.

Un ordre strict stable ne peut pas être total en général. En effet, si $P(x) > P(y)$, par exemple, alors on doit aussi avoir $P(y) > P(x)$, par la substitution $[x/y, y/x]$. Comme nous voulons restreindre les atomes sur lesquels nous résolvons aux atomes maximaux pour $>$, il est intéressant d'avoir le moins de choix possible, c'est-à-dire d'avoir aussi peu d'atomes incomparables que possible. Résoudre ce problème est encore impossible en général, mais nous pouvons améliorer$>$ en le forçant à avoir la propriété suivante :

Définition 21 (Totalité sur les termes clos) *Un ordre strict $>$ est dit* total sur les termes clos *si et seulement si pour tous atomes clos A et B, on a $A < B$, ou $A = B$, ou $A > B$.*

Les ordres totaux sur les termes clos existent, par exemple les ordres récursifs sur les chemins ("recursive path ordering" ou rpo) [DJ90] fondés sur une précédence totale. Nous définissons alors le raffinement suivant de la résolution :

Définition 22 (Stratégie ordonnée) *Soit $>$ un ordre strict stable sur les atomes (possiblement, et préférablement, total sur les termes clos). Soit $>^\#$ la relation définie par $s >^\# t$ si et seulement si $s > t$ ou s et t sont incomparables pour $>$, ou de façon équivalente si non $s < t$. Nous disons qu'un atome A est* maximal *dans un ensemble S d'atomes si et seulement si $A >^\# B$ pour tout $B \in S \setminus \{A\}$, c'est-à-dire si et seulement si A est plus grand que tout atome qui lui est comparable dans S, c'est-à-dire qu'aucun autre atome n'est plus grand que A dans S.*

La stratégie ordonnée *est le raffinement de la résolution où les facteurs de clauses C sont calculés en n'unifiant que les atomes maximaux de C, et où les résolvants binaires de clauses C et C' sont calculés en unifiant des littéraux complémentaires maximaux dans leurs clauses respectives.*

Théorème 23 (Complétude) *La stratégie ordonnée est correcte et complète.*

Preuve : Elle est clairement correcte, en tant que restriction de la résolution, qui est correcte.

Elle est complète au niveau des atomes clos (au niveau propositionnel). En effet, nous invoquons l'argument ci-dessus pour simuler Davis-Putnam par la résolution. Mais il y a encore un léger problème : lorsque A ou $\neg A$ est pur, c'est-à-dire quand $S_{\neg A}$ ou S_A est vide, il n'y a aucun résolvant sur A. Par exemple, A est pur dans l'ensemble des trois clauses $A \longrightarrow B$, $\longrightarrow B$ et $B \longrightarrow$, donc il n'y a pas de résolvant sur A; mais cet ensemble de clauses est quand même insatisfiable : il suffit de résoudre sur B à la place. Nous devons donc ignorer les clauses pures dans la preuve.

Soit S un ensemble insatisfiable fini de clauses closes. Nous simulons la règle de séparation de Davis et Putnam sur l'atome non pur A en calculant l'ensemble de tous les résolvants de clauses dans S sur A; soit $S(A)$ ce dernier ensemble. Alors $S(A)$ est une forme normale conjonctive de la disjonction de $S[\mathbf{T}/A]$ et $S[\mathbf{F}/A]$, en conjonction avec $S_\emptyset$, comme illustré sur la figure 3.

Comme S est insatisfiable, soit S doit contenir la clause vide, et nous avons fini; soit il existe au moins un atome non pur A dans S, sinon S serait satisfiable. Choisissons A maximal parmi tous les atomes non purs dans S, et calculons $S(A)$: c'est un ensemble de résolvants ordonnés de S sur A, union le sous-ensemble $S_\emptyset$ de

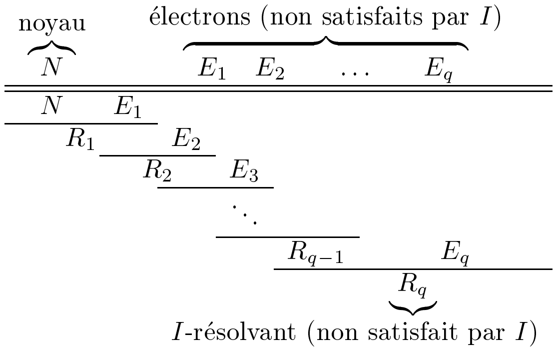


Figure 4: I-conflit en résolution sémantique

S. De plus, $S(A)$ contient strictement moins de littéraux purs que S, puisqu'aucun nouveau littéral n'a été créé, et les littéraux purs dans S restent purs dans $S(A)$; et $S(A)$ est clairement insatisfiable.

Cet argument mène à un argument facile de complétude par récurrence sur le nombre d'atomes non purs dans S.

Pour relever le résultat au premier ordre, nous appliquons le théorème de Herbrand : un ensemble S de clauses du premier ordre est insatisfiable si et seulement s'il existe un ensemble fini propositionnellement insatisfiable S' d'instances de clauses de S. Alors, nous pouvons utiliser la résolution ordonnée dans le cas clos pour dériver la clause vide de S', comme nous venons de le montrer. Finalement, cette résolution ordonnée close peut être simulée par des étapes de résolution au premier ordre comme au théorème 9, et ces étapes de résolution sont des étapes de résolution ordonnée, parce que $>$ est stable. $\square$

La résolution ordonnée est compatible avec l'élimination de tautologies, de clauses subsumées et de clauses pures. En d'autres termes, la combinaison de ces techniques fournit encore une procédure complète de réfutation. En fait, la preuve ci-dessus montre que l'élimination des clauses pures est compatible avec la stratégie ordonnée. En ce qui concerne la subsomption et l'élimination de tautologies, la clef est que le lemme 12 continue de s'appliquer dans le cas ordonné, parce que par stabilité de l'ordre, tout littéral maximal dans une clause $\Gamma', \Gamma\sigma \longrightarrow \Delta\sigma, \Delta'$ subsumée par $\Gamma \longrightarrow \Delta$ est dans Γ', Δ' ou est une instance d'un littéral maximal de $\Gamma \longrightarrow \Delta$.

4.4 Résolution sémantique et stratégies d'ensemble de support

Les *résolutions sémantiques* guident les règles de résolution au moyen d'une interprétation I. Considérons la procédure de Davis-Putnam de nouveau. Dans le cas clos (propositionnel), la stratégie ordonnée revient à fixer un ordre sur les atomes clos A_1, A_2, ..., A_n; nous séparons alors sur A_1, puis sur A_2, et ainsi de suite. Les résolutions sémantiques contrôlent quelle branche nous explorons d'abord à chaque séparation. Chaque fois que nous souhaitons séparer sur A_i, nous pouvons essayer d'explorer le cas A_i vrai d'abord, puis le cas A_i faux; ou bien le contraire. Nous pouvons coder ce choix par une interprétation I : si I rend A_i vrai, alors nous choisissons d'explorer le cas A_i vrai d'abord, sinon nous explorons le cas A_i faux d'abord.

Disons que I *satisfait* une clause C si et seulement si elle satisfait C pour toutes les affectations possibles de ses variables libres. Autrement dit, si x_1, ..., x_n sont les variables libres de C, si elle satisfait la clôture universelle $\forall x_1 \cdot \ldots \cdot \forall x_n \cdot C$ de C. Un autre point de vue sur la résolution sémantique est le suivant.

Supposons par exemple que notre ensemble initial de clauses S consiste en, tout d'abord, un ensemble S' de clauses exprimant un certain nombre d'axiomes de l'arithmétique, et ensuite, un ensemble S'' de clauses représentant la version skolémisée de la négation d'une proposition à prouver dans le système S'. Comme l'arithmétique est cohérente, elle a un modèle I, à savoir l'ensemble $\mathbb{N}$ des entiers naturels muni des opérations usuelles. Nous pouvons alors utiliser ce modèle I pour nous guider au travers de la preuve de notre proposition. L'idée de la résolution sémantique est d'essayer de reconnaître quelles clauses sont déjà satisfaites par I, et d'interdire toutes les étapes de résolution entre deux de ces clauses. Intuitivement, cela ne nous aiderait pas à progresser vers la dérivation de la clause vide, car nous ne pouvons pas la dériver à partir d'un ensemble satisfiable de clauses.

N'importe quelle interprétation auxiliaire convient pour la résolution sémantique. Une des interprétations les plus simples est l'*interprétation de Herbrand positive* dans laquelle tous les atomes clos sont supposés vrais. Alors I satisfait une clause C si et seulement si son côté droit est non vide. Nous appelons une telle clause $\Gamma \longrightarrow$, qui n'est pas satisfaite par l'interprétation positive, une *clause négative*.

De façon symétrique, dans l'*interprétation de Herbrand négative*, tous les atomes clos sont supposés faux. Dans cette interprétation, une clause est satisfaite si et seulement son côté gauche est non vide, autrement dit si elle n'est pas une *clause positive* $\longrightarrow \Delta$.

Tout ceci se résume dans le raffinement suivant de la résolution :

Définition 24 (Résolution sémantique) *Soit I une interprétation. Soit A_I un algorithme (terminant) tel que, si A_I retourne "oui" sur l'entrée C, alors I satisfait C.*

*Un I-*conflit *est un ensemble fini de clauses $\{N, E_1, \ldots, E_q\}$, $q \leq 1$, telles que :*

- *A_I retourne "non" sur chacune de $E_1, \ldots, E_q$;*
- *il existe un résolvant R_1 de N et E_1, un résolvant R_2 de R_1 et E_2, ..., un résolvant R_q de R_{q-1} et E_q;*
- *A_I retourne "oui" sur $R_1, \ldots, R_{q-1}$, et "non" sur R_q.*

Les E_i sont appelées les électrons*, N est appelée le* noyau *du I-conflit, et R_q est appelé un I-*résolvant *du I-conflit.*

L'algorithme A_I est la partie bizarre de la définition. Dans l'interprétation positive (resp. négative), il suffit que A_I retourne "oui" sur les clauses négatives (resp. positives), et "non" sur les autres, et ceci caractérise exactement les clauses qui sont satisfaites par l'interprétation. Mais dans des utilisations plus élaborées de la I-résolution, la satisfaction par I peut ne pas être décidable : dans ce cas, la contrainte que nous imposons à A_I en est une de *sécurité*. Dans le cas trivial où A_I retourne toujours "non" — un cas vraiment très sûr — la résolution sémantique n'est rien d'autre que la résolution usuelle, non contrainte, et nous n'avons pas gagné grand chose. Nous souhaiterons donc que A_I retourne "oui" sur le plus de clauses possibles.

Voir la figure 4 pour une illustration de ce qu'est un I-conflit. Remarquer que le noyau peut avoir été détecté ou non comme étant satisfait par I. D'une façon plus informelle, la résolution sémantique fonctionne comme suit. Prenons un ensemble S de clauses, et découpons-le en l'ensemble S' des clauses sur lesquelles A_I retourne "oui" (l'ensemble des clauses dont on est sûr qu'elles sont satisfaites par I), et en l'ensemble S'' des clauses sur lesquelles A_I retourne "non" (l'ensemble des clauses qui ne sont peut-être pas satisfaites par I). Ensuite, choisissons une clause N dans $S' \cup S''$, et une clause E_1 dans S'', et résolvons pour produire une nouvelle clause R_1. (Si nous avions choisi N et E_1 venant toutes les deux de S', nous saurions que I satisfaisait R_1, donc ce serait un lemme inutile.) Si A_I retourne "non" sur R_1, alors R_1 est possiblement non satisfaite par I, et ceci nous apporte une information nouvelle, donc nous ajoutons R_1 à S'', et ceci termine l'étape de I-résolution. Sinon, nous résolvons entre R_1 et une clause E_2 de S'', ce qui produit R_2, et ainsi de suite jusqu'à ce que nous produisions une clause R_q sur laquelle A_I retourne "non", et alors nous ajoutons R_q à S'', ce qui termine l'étape de I-résolution.

En général, cependant, nous ne pouvons pas procéder comme cela, car nous ne connaissons aucune borne a priori sur q, et nous pourrions nous retrouver embarqués dans une boucle infinie, à chercher un I-résolvant sur un mauvais noyau. Nous devrons donc effectuer les recherches sur tous les noyaux possibles en parallèle. En pratique, on n'utilise que de très simples cas de résolution sémantique, comme l'hyper-résolution ou la stratégie d'ensemble de support, où ce problème ne se pose pas. En fait, la résolution sémantique est essentiellement un outil théorique pour explorer des raffinements plus pratiques de la résolution.

Nous montrons le théorème de complétude attendu avant de décrire les examples important sus-mentionnés de résolution sémantique.

Théorème 25 (Complétude) *Étant donnée une interprétation arbitraire I, la I-résolution est correcte et complète.*

Preuve : La I-résolution est correcte en tant que restriction de la résolution. Nous affirmons qu'elle est complète, ce que nous montrons dans le cas clos d'abord. Nous relevons ensuite le résultat au premier ordre par le théorème de Herbrand.

Soit S un ensemble fini insatisfiable de clauses closes. Nous montrons par récurrence sur le nombre d'atomes apparaissant dans S que l'on peut dériver la clause vide à partir de S par une réfutation par I-résolution. Si ce nombre est 0, alors S doit être $\{\longrightarrow\}$, et l'affirmation est prouvée. Sinon, soit A un atome apparaissant dans S, et supposons sans perte de généralité que A est vrai dans I. (Il est sensé de dire que A est vrai ou faux dans I, parce que A est un atome clos.)

Comme S est insatisfiable, $S[\mathbf{T}/A]$ est insatisfiable aussi. Par hypothèse de récurrence, il existe une dérivation D_1 de $\longrightarrow$ par I-résolution à partir de $S[\mathbf{T}/A]$. Nous la traduisons en une dérivation D'_1 par I-resolution de $A \longrightarrow$ comme suit.

En premier, on peut voir D_1 comme un arbre inversé, où les nœuds sont des clauses, les feuilles sont des clauses dans $S[\mathbf{T}/A]$, et la racine est la clause vide $\longrightarrow$. Construisons D'_1 en rajoutant A à gauche des clauses $\Gamma \longrightarrow \Delta$ dans $S[\mathbf{T}/A]$ aux feuilles qui venaient de clauses $A, \Gamma \longrightarrow \Delta$ dans S, et en ajoutant récursivement A à gauche des clauses des nœuds intérieurs si nous l'avions ajouté à gauche d'un de ses parents. (Cette transformation n'est pas nécessairement unique.) D'_1 est encore une dérivation par I-résolution, parce que les électrons augmentés par A à gauche restent faux dans I. La conclusion de D'_1 peut être $\longrightarrow$ de nouveau, auquel cas nous avons terminé; ou bien $A \longrightarrow$, qui est fausse dans I. Supposons ce dernier cas.

Maintenant, comme S est insatisfiable, $S[\mathbf{F}/A]$ l'est aussi, donc par hypothèse de récurrence, il existe une réfutation D_2 par I-résolution de $S[\mathbf{F}/A]$. Nous transformons D_2 en une réfutation D'_2 par I-résolution de $S \cup \{A \longrightarrow\}$ comme suit. L'idée est de réinsérer A à droite des clauses de D_2, mais ça ne fonctionne pas : certains électrons et certains I-résolvants peuvent devenir vrais dans I.

Mais nous pouvons réobtenir une dérivation par I-résolution en résolvant avec l'électron $A \longrightarrow$. Soit N, E_1, …, E_q un I-conflit arbitraire dans D_2. En ajoutant A à droite de certaines de ces dernières clauses, nous obtenons une suite N', E'_1, …, E'_q de clauses, où N' est N ou N plus A à droite, et pour tout i, $1 \leq i \leq q$, E'_i est soit E_i soit E_i plus A à droite. Pour tout i, si $E'_i = E_i$, alors E'_i est un électron, et nous ne le modifions pas; sinon, nous remplaçons E'_i par le I-conflit E'_i (noyau), $A \longrightarrow$ (électron) pour déduire E_i (qui est effectivement fausse dans I). Ceci nous permet d'obtenir une suite d'étapes de résolution entre N', $E_1, \ldots, E_q$. Si $N' = N$, alors nous avons reconstruit le I-conflit de D_2. Sinon, nous produisons le I-conflit N', $A \longrightarrow$, $E_1, \ldots, E_q$: comme la résolution entre N' et $A \longrightarrow$ produit N, cet I-conflit a exactement la même clause finale que $N, E_1, \ldots, E_q$. Par récurrence sur la longueur de D_2, nous produisons ainsi une réfutation par I-résolution à partir de $S \cup \{A \longrightarrow\}$.

Pour relever le résultat au premier ordre, nous appliquons le théorème de Herbrand : un ensemble S de clauses du premier ordre est insatisfiable si et seulement s'il existe un ensemble fini propositionnellement insatisfiable S' d'instances de clauses de S. Alors nous pouvons utiliser la I-résolution dans le cas clos pour dériver la clause vide de S', comme nous venons de le montrer. Finalement, cette résolution close peut être simulée par des étapes de résolution comme dans le théorème 23, et ces étapes de résolution sont en fait des étapes de I-résolution. En effet, lorsqu'une instance close E' d'une clause E est fausse dans I, alors A_I doit retourner "non" sur E : par notre hypothèse de sécurité, si A_I retournait "oui", alors E serait satisfaite par I. □

Non seulement la résolution sémantique est complète, pour n'importe quelle interprétation I, mais les stratégies d'effacement (tautologies, subsomption, clauses pures) préservent la complétude. Refaisons en effet la preuve du lemme 12 : le point-clé est que si un électron est subsumé par une autre clause, alors cette dernière est aussi un électron, autrement dit elle est falsifiée par I.

Cependant, la résolution sémantique *ordonnée* n'est *pas* complète en général (par exemple, l'hyper-résolution positive ordonnée ne peut pas dériver $\longrightarrow$ à partir de l'ensemble insatisfiable de clauses $\longrightarrow A, B$; $B \longrightarrow C$; $A \longrightarrow$; $C \longrightarrow$ dans l'ordre $A < B < C$; laissé en exercice). Nous pouvons cependant utiliser le raffinement suivant :

Définition 26 *La résolution sémantique* semi-ordonnée *est le raffinement de la résolution sémantique où nous exigeons que le littéral sur lequel on résout à chaque étape soit maximal dans son électron.*

Autrement dit, nous exigeons qu'il soit maximal dans chaque E_i, mais pas dans R_i, $1 \leq i \leq q$. Alors :

Théorème 27 *Étant donnée une interprétation arbitraire I, et n'importe quel ordre strict stable $>$, la I-résolution semi-ordonnée est correcte et complète.*

Preuve : Analogue au théorème 25, à part que nous choisissons A minimal. Lorsque nous ajoutons A à un électron E, le littéral sur lequel on résout dans E reste maximal dans E augmenté par A. De plus, dans $A \longrightarrow$, A est nécessairement maximal. Donc toutes les dérivations qui sont produites dans la preuve sont encore semi-ordonnées. □

Il est clair que la résolution semi-ordonnée reste complète lorsqu'on utilise les stratégies d'effacement.

4.4.1 Hyper-résolution

Le premier cas de résolution sémantique est l'*hyper-résolution*:

Définition 28 (Hyper-résolution)
*L'*hyper-résolution positive *est la I-résolution, où les électrons et les résolvants sont contraints à être des clauses positives. (Autrement dit, l'interprétation I est l'interprétation de Herbrand négative.)*

*L'*hyper-résolution négative *est la I-résolution, où les électrons et les résolvants sont contraints à être des clauses négatives. (Autrement dit, l'interprétation I est l'interprétation de Herbrand positive.)*

Par le théorème 27, les hyper-résolutions positive et negative, et même leurs variantes semi-ordonnées, sont complètes. Pour coder, disons, l'hyper-résolution positive, nous assouplissons la formalisation par noyau et électrons : prenons une clause positive, résolvons-la avec une autre clause, et ajoutons le résolvant à l'ensemble courant de clauses. L'hyper-résolution positive semi-ordonnée consiste alors à ne résoudre que sur les littéraux maximaux dans la clause positive choisie.

Comme Chang et Lee le font remarquer, l'hyper-résolution positive est similaire à la pensée en avant, c'est-à-dire à utiliser des faits disjonctifs (les clauses positives) comme électrons, pour déduire davantage de faits, jusqu'à ce qu'on obtienne la clause vide, c'est-à-dire un ensemble de faits contradictoires. Si nous voulons prouver $A \Rightarrow B$, nous fabriquons une clause $\longrightarrow A$ qui dénote un fait, et une clause $B \longrightarrow$ qui représente un but : l'hyper-résolution positive déduira alors des faits à partir de $\longrightarrow A$, et les annulera en utilisant l'information information négative sur B pour produire la clause vide.

Réciproquement, l'hyper-résolution négative est similaire à la pensée en arrière, c'est-à-dire à réduire le but à prouver en sous-buts, jusqu'à ce qu'il ne reste plus de sous-but. Dans l'exemple ci-dessus, l'hyper-résolution négative fabriquera des clauses négatives à partir de $B \longrightarrow$, jusqu'à ce que tous les sous-buts (les atomes à gauche des clauses négatives) soient prouvés par des faits dérivés de A.

4.4.2 Ensemble de support

L'autre cas important de résolution sémantique est la *stratégie d'ensemble de support*. L'idée est la suivante : nous ne voulons pas en général prouver des formules isolées Φ, mais plutôt Φ, étant donné un ensemble d'axiomes $\mathcal{A}$. Autrement dit, et en supposant que $\mathcal{A}$ est fini, nous voulons prouver $\mathcal{A} \Rightarrow \Phi$, où nous avons noté $\mathcal{A}$ la conjonction de tous les axiomes dans l'ensemble $\mathcal{A}$. Nous pouvons usuellement prouver ou au moins supposer que $\mathcal{A}$ est cohérent : les axiomes ne sont pas des hypothèses temporaires, ils sont censés décrire un modèle. Le problème alors, si l'on cherche à prouver $\mathcal{A} \Rightarrow \Phi$, est que la recherche de preuve passera une portion considérable de son temps à réfuter $\mathcal{A}$, et on sait à l'avance — ou on croit — que c'est tout à fait inutile. Ceci est particulièrement clair dans le cas de la résolution, où pour prouver $\mathcal{A} \Rightarrow \Phi$, nous produirions des clauses pour $\mathcal{A}$, et quelques clauses pour $\neg\Phi$, mais toutes les résolution entre clauses de $\mathcal{A}$ sont inutiles !

On peut y remédier en utilisant la résolution sémantique, où l'interprétation guide I est un modèle arbitraire des axiomes. Comme nous ne savons rien à propos de I à l'avance, nous choisissons comme algorithme A_I un des plus simples qui soient sûrs : répondre "oui" sur l'entrée C si et seulement si C est une des clauses axiomes.

Puisque A_I est si simple, la stratégie d'ensemble de support peut être réalisée comme suit : diviser l'ensemble initial S en deux sous-ensembles disjoints S' (l'ensemble des clauses venant des axiomes dans $\mathcal{A}$) et S'' (les autres clauses, correspondant à la négation du but Φ à prouver). L'ensemble S'' est alors l'*ensemble de support* pour le processus de résolution. Maintenant, nous appliquons la résolution répétitivement, en sélectionnant des clauses parentes de sorte qu'au moins une des clauses parentes vienne de l'ensemble de support S'', et nous ajoutons les résolvants à l'ensemble de support S''. (Remarquer que l'on peut aussi coder l'hyper-résolution comme ceci, et en fait n'importe quelle résolution sémantique.)

En tant que cas particulier de résolution sémantique, la stratégie d'ensemble de support est complète, pourvu que les axiomes aient au moins un modèle. Nous pou-

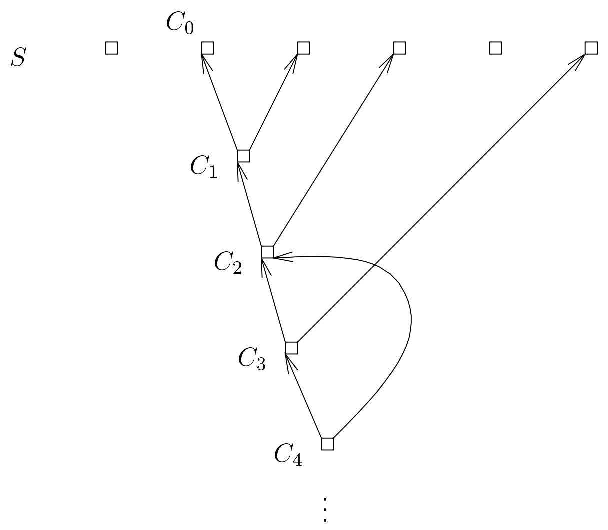


Figure 5: Résolution linéaire

vons aussi utiliser la restriction semi-ordonnée sans perdre la complétude, ainsi que toutes les stratégies d'effacement. En pratique, bien que la résolution sémantique semble prometteuse, elle est considérablement moins efficace que ce à quoi on s'attendrait. Une source possible d'inefficacité est que les résolvants sont toujours mis dans l'ensemble de support S'', et ne sont jamais reconnus comme satisfaits par I, même s'ils le sont : l'algorithme A_I ci-dessus est trop simpliste, et nous aurions réellement besoin d'un algorithme plus spécialisé.

4.5 La stratégie linéaire

La quantité de non-déterminisme dans le choix des clauses à résoudre, même dans les cas de la résolution ordonné ou de la I-résolution semi-ordonnée, peut être terrifiant. Un raffinement de la résolution où ce niveau de non-déterminisme est réduit est la *résolution linéaire*, où nous choisissons une clause C_0 de l'ensemble de clauses initiales S, nous la résolvons avec une clause pour produire un résolvant C_1, que nous ajoutons à S; nous résolvons ensuite C_1 avec une autre clause pour produire C_2, que nous ajoutons à S, et ainsi de suite, jusqu'à ce que nous obtenions la clause vide.

Le non-déterminisme est réduit, parce qu'un des clauses à résoudre est toujours fixée : c'est la dernière clause que nous avons produite par résolution. Le terme "résolution linéaire", qui ne se réfère pas à la logique linéaire, provient de la forme spéciale des dérivations par résolution dans ce cas : voir figure 5. Définissons :

Définition 29 (Résolution linéaire) *Une* dérivation par résolution linéaire *est une dérivation à partir d'un ensemble S de clauses C_0, C_1, …, C_i, telles que $C_0 \in S$, et pour tout $i \geq 0$, C_{i+1} est un résolvant de C_i avec une clause de $S \cup \{C_1, \ldots, C_i\}$.*

Si C_{i+1} est un résolvant de C_i avec une clause C, nous disons que C_i est la clause centrale *("center clause" en anglais) de l'étape de résolution, C est la* clause latérale *("side clause"), et C_{i+1} est la nouvelle* clause au sommet *("top clause"). Les clauses C_0, …, C_i sont appelées les* clauses ancêtres *("ancestor clauses"). Les clauses de S sont appelées les* clauses d'entrée *("input clauses").*

Une dérivation par résolution linéaire est dite semi-ordonnée *par rapport à un ordre strict stable $>$ si et seulement si le littéral sur lequel on résout dans la clause centrale est maximal pour $>$.*

Théorème 30 (Complétude) *La résolution linéaire (resp. semi-ordonnée) est correcte et complète.*

Preuve : Elle est correcte en tant que raffinment de la résolution.

L'idée principale est de rejouer la preuve du théorème 25, en utilisant le fait suivant : étant donné un ensemble insatisfiable fini S de clauses, il existe un ensemble S' de clauses de S qui est *minimalement insatisfiable*, c'est-à-dire un dont tout sous-ensemble propre est satisfiable. Cet ensemble minimal peut ne pas être unique, mais cela n'a pas d'importance. Sans perte de généralité, supposons que S lui-même est minimalement insatisfiable.

Remarquons qu'un ensemble minimalement insatisfiable S de clauses ne peut pas contenir de clauses pures ou de tautologies. Nous en concluons que pour tout atome A apparaissant dans S, A apparaît à gauche d'une clause et à droite d'une autre clause dans S; et que A ne peut pas apparaître à la fois à gauche et à droite de la même clause.

Nous montrons qu'il existe une réfutation par résolution linéaire à partir de S par récurrence sur le nombre d'atomes apparaissant dans S. Si ce nombre est 0, alors S doit être $\{\longrightarrow\}$, et nous avons fini. Sinon, le résultat est une conséquence du fait plus précis suivant, que nous montrons par récurrence : si S' est un ensemble satisfiable de clauses, et $S' \cup \{C_0\}$ est insatisfiable, alors il existe une réfutation par résolution linéaire de $S' \cup \{C_0\}$.

Si C_0 est la clause vide, nous avons fini. Sinon, soit A un atome quelconque apparaissant dans C_0. Sans perte de généralité, supposons que A apparaisse à gauche, autrement dit C_0 est de la forme $A, \Gamma_0 \longrightarrow \Delta_0$.

Comme S est insatisfiable, $S[\mathbf{T}/A]$ est insatisfiable aussi, donc par hypothèse de récurrence il existe une réfutation linéaire D_1 de $\longrightarrow$ à partir de $S[\mathbf{T}/A]$. Comme dans la preuve du théorème 25, nous transformons D_1 en une dérivation D'_1 en rajoutant A à gauche de toutes les clauses appropriées dans D_1. D'_1 est une dérivation linéaire, et elle conclut soit $\longrightarrow$ soit $A \longrightarrow$. Dans le premier cas, nous avons fini, donc supposons que nous sommes dans le deuxième cas.

Maintenant $S' \cup \{A \longrightarrow\}$ est aussi insatisfiable, parce que toute interprétation qui satisfait à la fois S' et $A \longrightarrow$ satisfait aussi S' et $A, \Gamma_0 \longrightarrow \Delta_0$, c'est-à-dire S. En particulier, $(S' \cup \{A \longrightarrow\})[\mathbf{F}/A]$ est insatisfiable, autrement dit $S'[\mathbf{F}/A]$ est insatisfiable. Soit S'' un sous-ensemble minimalement insatisfiable de $S'[\mathbf{F}/A]$. Comme S'' est insatisfiable, S'' n'est pas un sous-ensemble de S', qui est satisfiable. Donc il existe une clause $C_1 = \Gamma_1 \longrightarrow \Delta_1$ dans S'' qui n'est pas dans S'. Cette clause doit provenir d'une clause dans S' qui n'est pas C_1, donc de $\Gamma_1 \longrightarrow \Delta_1, A$.

Par hypothèse de récurrence, il existe une réfutation linéaire D_2 de $S'[\mathbf{F}/A]$ de clause au sommet initiale C_1. Transformons D_2 en une réfutation linéaire D'_2 de $S' \cup \{A \longrightarrow\}$ de clause au sommet initiale $A \longrightarrow$ comme suit. D'abord, D'_2 résout la clause au sommet $A \longrightarrow$ avec $\Gamma_1 \longrightarrow \Delta_1, A$ (qui est dans S'), ce qui produit C_1 comme nouvelle clause au sommet. Si C_1 est la clause vide, nous avons fini. Sinon, la première étape de D_2 était une résolution entre la clause au sommet C_1 et une autre clause C'_1 dans $S'[\mathbf{F}/A]$, et produisait une clause C_2 comme nouvelle clause au sommet. Si C'_1 est dans S', alors D'_2 se contente de résdoure entre les deux mêmes clauses pour obtenir la même clause au sommet C_2. Sinon, il existe une clause $\Gamma_1 \longrightarrow \Delta_1, A$ dans S' telle que $\Gamma_1 \longrightarrow \Delta_1 = C'_1$. Alors D'_2 résout d'abord entre C_1 et $\Gamma_1 \longrightarrow \Delta_1, A$ pour obtenir C_2 avec un A en plus à droite comme nouvelle clause au sommet, et alors nous résolvons cette dernière clause avec la clause ancêtre $A \longrightarrow$ pour réobtenir C_2. Par une récurrence sur la longueur de D_2 (que nous ne formalisons pas ici), nous obtenons ainsi une réfutation linéaire D'_2 de $S' \cup \{A \longrightarrow\}$ de clause au sommet initiale $A \longrightarrow$.

La concaténation de D'_1 et de D'_2 fournit alors une réfutation par résolution linéaire de S de clause au sommet initiale C_0, comme annoncé. Nous relevons ensuite le résultat au premier ordre comme d'habitude.

La résolution linéaire semi-ordonnée est complète par un raisonnement analogue : il suffit de choisir pour A un atome minimal, et pour C_0 une clause le contenant, comme dans le théorème 27. □

La résolution linéaire n'est pas complète pour n'importe quel choix de C_0; au contraire, il existe un choix de la clause initiale C_0 qui mènera à la clause vide, si S est insatisfiable. Nous pouvons combiner cette stratégie avec celle d'ensemble de support, et choisir C_0 en-dehors de l'ensemble des clauses axiomes. En particulier, si le but est une conjonction universellement quantifiée de formules atomiques, il n'y a qu'une clause non-axiome, et le choix est alors fixé.

Il est aussi possible que la clause au sommet C_i doive être résolue non seulement avec des clauses d'entrée (celles de S), mais aussi avec des clauses ancêtres C_j, $1 \leq j < i$. La restriction de la résolution linéaire où les clauses latérales sont des clauses d'entrée est appelée l'*input-résolution*. Mais l'input-résolution n'est pas complète. Par exemple, il n'y a pas de réfutation par input-résolution de $\longrightarrow A, B$, $A \longrightarrow B$, $B \longrightarrow A$, $A, B \longrightarrow$. En fait, il y a une réfutation par input-résolution d'un ensemble S de clauses si et seulement s'il y a une réfutation par résolution unitaire de S (cf. [CL73], pages 134–135), et nous avons déjà vu qu'il n'y avait pas de réfutation par résolution unitaire de ce dernier ensemble de clauses. L'input-résolution est cependant complète pour les ensembles de Horn (les programmes Prolog, en gros).

Finalement, remarquons qu'*aucune* stratégie d'effacement n'est compatible avec la stratégie linéaire : effacer la clause courante n'aurait pas de sens.

4.6 Autres raffinements de la résolution

Il existe de nombreux autres raffinements de la résolution. Consulter [CL73] pour un panorama, ainsi que pour les raisons détaillées pour lesquelles ils fonctionnent et comment.

La résolution ordonnée peut être définie au moyen de différentes notions d'ordre. La *lock-résolution* de Robert Boyer, par exemple, (cf. [CL73]), affecte des rangs arbitraires à chaque *occurrence* de littéraux dans les clauses.

Une variante intéressante de la résolution linéaire appelée *OL-résolution* (pour ordonnée linéaire, mais ce n'est pas exactement ce que nous avons appelé résolution ordonnée linéaire en section 4.5), est la résolution linéaire où les clauses sont codées comme des *listes* de littéraux sans répétition; l'ordre dans lequel les littéraux apparaissent joue le rôle de l'ordre sur les atomes que nous avons utilisé pour restreindre les dérivations par résolution. La méthode est complète. De plus, des techniques de codages intelligentes peuvent être utilisées pour ne représenter que la clause au sommet C_i, qui codent les clauses centrales $C_1, \ldots, C_{i-1}$ à l'intérieur de C_i. (Voir [CL73], section 7.4.) Nous verrons au prochain chapitre comment cette technique peut en fait être vue comme une technique de tableaux.

La *résolution à graphe de connexions* ("connection graph resolution") de Robert Kowalski est plus élaborée, car elle est à la fois une réalisation particulière et une méthode pour empêcher la génération de nombreuses clauses inutiles. Dans cette méthode, les paires unifiables de littéraux complémentaires sont représentées par des *liens*, c'est-à-dire des arêtes dans un graphe de clauses; les résolvants héritent implicitement les liens de leurs clauses parentes, de sorte que le nombre d'unifications à effectuer est minimisé. Une clause possédant un littéral sans lien est pure, et est immédiatement éliminée. Ceci peut ensuite couper d'autres liens, ce qui crée de nouvelles clauses pures, et ainsi de suite. Pour éviter les redondances, une fois qu'un lien a été utilisé pour produire un résolvant, il est coupé pour qu'il ne puisse pas être réutilisé. C'est une réalisation très efficace de la résolution. Cependant, le fait de couper les liens utilisés peut faire perdre la complétude en présence de certaines stratégies; et lorsque la résolution à graphe de connexions est complète, la preuve de complétude est en général complexe.

Ces problèmes sont résolus dans la *soustraction d'instances* de Jean-Paul Billon, mais la soustraction d'instances ne fonctionne pas avec la résolution. À la place, Billon code la soustraction d'instances au-dessus du calcul de Lee et Plaisted [LP92], où ce sont des instances de clauses, pas des résolvants, qui sont produits au moyen de la règle :

$$\frac{C \wedge A \qquad \neg A' \wedge C'}{(C \wedge A)\sigma \quad (\neg A' \wedge C')\sigma}$$

où les clauses sous la ligne sont ajoutées à l'ensemble courant de clauses, et σ est le mgu de A et A'. Cette idée de Lee et Plaisted permet au prouveur de découpler la tâche de recherche d'unificateurs (par unification de littéraux complémentaires) de la tâche consistant à montrer que l'ensemble de clauses close sous-jacent est propositionnellement insatisfiable. Cette second tâche peut être laissée à n'importe quelle méthode propositionnelle, incluant la résolution, mais pas uniquement la résolution.

Au-dessus de ce calcul, l'idée de Billon est d'attacher à chaque clause un ensemble de substitutions qui ont été utilisée pour produire des instances, et qui ne doivent pas être réutilisées. Ceci évite d'engendrer la même clause plusieurs fois dans la plupart des cas. Au lieu de couper des liens, elle interdit les unifications le long des liens précédents en comparant les substitutions. C'est plus général que la technique de Kowalski, puisque cela ne se limite pas à couper le lien sur lequel on a résolu, et il est plus facile de montrer que c'est complet. Billon a depuis considérablement amélioré l'idée au point que l'on n'a même plus besoin d'enregistrer ces substitutions, en utilisant intelligemment une notion de chemins à travers l'ensemble des clauses [Bil96].

5 La résolution comme preuve par coupure seulement

Nous terminons ce chapitre par une analyse de ce que nous avons fait d'un point de vue de théorie de la preuve. Nous avons en effet développé la plus grande partie de la résolution, comme il est traditionnel, d'un point de vue essentiellement sémantique. Cependant, la règle de résolution ressemble tellement à la règle Cut de coupure dans le système de Gentzen **LK** qu'il doit exister une correspondance entre les deux (cf. ici dans le cas propositionnel). Cette correspondance, comme on peut s'y attendre, provient d'une variante du théorème d'élimination des coupures. Ceci apportera aussi une réponse à l'énigme : comme nous pouvons éliminer toutes les coupures, comment ceci justifie-t-il la résolution, où toutes les règles *sauf* la coupure sont éliminées ?

En premier, généralisons un peu le théorème d'élimination des coupures au premier ordre :

Théorème 31 (Élimination des coupures généralisée) *Soit S un ensemble de séquents atomiques. Soit $\mathbf{LK} + S$ le calcul des séquents obtenu en ajoutant aux règles de* **LK** *toutes les règles de la forme :*

$$\overline{\Gamma \longrightarrow \Delta}$$

où $\Gamma \longrightarrow \Delta$ *est un séquent de* S*. De telles règles sont appelées des* axiomes non logiques.

Soit S_∞ *le plus petit ensemble de séquents contenant* S *et stable par l'application de la règle Cut. (Autrement dit, si* $\Gamma \longrightarrow \Delta, \Phi$ *et* $\Phi, \Gamma' \longrightarrow \Delta'$ *sont dans* S_∞*, alors* $\Gamma, \Gamma' \longrightarrow \Delta, \Delta'$ *aussi.) Nous disons qu'une coupure est* non logique *si et seulement si ses prémisses sont toutes les deux dans* S_∞*. (Alors, sa conclusion est aussi dans* S_∞*.)*

Alors un séquent atomique est prouvable dans $\mathbf{LK} + S$ *si et seulement s'il est un affaiblissement d'un séquent prouvable dans* $\mathbf{LK} + S$ *par une preuve où toutes les coupures sont non logiques.*

Preuve : Remarquons que les étapes d'affaiblissement peuvent être codées sous forme de coupures avec Ax, pourvu que le séquent affaibli ne soit pas vide. De plus, toute suite d'affaiblissements est équivalent à un affaiblissement unique : nous supposerons donc implicitement que la conclusion d'un séquent affaibli n'est jamais coupée avec Ax.

Nous montrons qu'un séquent est prouvable dans $\mathbf{LK} + S$ si et seulement s'il est dérivable par une preuve où les coupures sont soit non logiques, soit sont des coupures entre une version affaiblie d'un séquent de S_∞ et Ax (fournissant encore une version affaiblie du même séquent en conclusion).

La preuve est la même que pour le théorème d'élimination des coupures. Nous disons qu'une coupure est *essentielle* si elle est non logique et qu'elle n'est pas une coupure avec Ax. Nous allons éliminer toutes les coupures non essentielles.

Redéfinissons le rang de coupure $r(\pi)$ d'une preuve π dans $\mathbf{LK}+S$, comme valant 0 si π ne contient que des coupures essentielles (au lieu que π soit sans coupure dans le cas de $\mathbf{LK}$), et sinon comme le maximum des $d(\pi') + d(\Phi)$, où π' parcourt toutes les sous-preuves de π se terminant sur une coupure logique entre deux preuves ne contenant que des coupures essentielles, et de formule de coupure Φ. Ensuite nous devons montrer que, pour toute preuve π de $\mathbf{LK} + S$ de rang de coupure non nul r, nous pouvons la transformer en une preuve de $\mathbf{LK} + S$ de rang au plus $r - 1$ du même séquent, où aucune coupure non logique ne se trouve sous une coupure avec Ax.

Pour être sûr qu'aucune coupure non logique ne se trouve sous une coupure avec Ax, nous pouvons toujours pousser les coupures avec Ax (les affaiblissements) vers le bas de la dérivation : si $\Gamma \longrightarrow \Delta$ produit $\Gamma, \Gamma' \longrightarrow \Delta, \Delta'$ par affaiblissement, et si nous inférons un séquent $\Gamma'' \longrightarrow \Delta''$ à partir de ce dernier par une règle R, alors soit R agit sur une formule qui était déjà dans $\Gamma \cup \Delta$, et les règles permutent, soit R agit sur une autre formule, et alors $\Gamma'' \longrightarrow \Delta''$ est une version affaiblie de $\Gamma \longrightarrow \Delta$, comme on peut le vérifier sur les règles de $\mathbf{LK}$. Nous supposerons tacitement que cette normalisation est systématiquement effectuée.

Comme dans la preuve de l'élimination des coupures, choisissons une sous-preuve π' de π la plus haute possible (maximale) se terminant sur une coupure non essentielle sur Φ, avec $d(\pi') + d(\Phi)$ maximal. Alors π' est de la forme :

$$\dfrac{\begin{array}{c}\pi_1\\ \vdots\\ \Gamma \longrightarrow \Delta, \Phi\end{array} \qquad \begin{array}{c}\pi_2\\ \vdots\\ \Gamma', \Phi \longrightarrow \Delta'\end{array}}{\Gamma, \Gamma' \longrightarrow \Delta, \Delta'}\ \text{Cut}$$

où π_1 et π_2 ne contiennent aucune coupure non essentielle. Comme nous avons supposé que tous les affaiblissements seraient systématiquement poussés en-dessous des autres coupures, π_1 et π_2 ne contiennent en fait aucune coupure logique (non essentielle ou affaiblissement).

Par construction, la dernière règle de π_1 n'est pas une coupure non logique, ou la dernière règle de π_2 n'est pas une coupure non logique. En effet, le contraire signifierait que la coupure ci-dessus est non logique, donc essentielle. Supposons sans perte de généralité que π_1 ne se termine pas sur une coupure non logique. Alors elle ne se termine pas par une coupure, parce que π_1 ne contient aucune coupure logique. Si π_2 ne se termine pas par une coupure, alors nous continuons comme dans le théorème d'élimination des coupures.

Sinon, π_2 se termine par une coupure non logique, et π_1 ne se termine pas par une coupure. Comme S_∞ ne contient que des séquents atomiques, la formule de coupure Φ doit être atomique. Alors, soit Φ n'est pas la formule principale dans la dernière règle de π_1, et nous faisons remonter la coupure le long de π_1; soit Φ est la formule principale, mais comme Φ est atomique, π_1 doit se terminer par une instance de Ax, de sorte que $\Gamma, \Gamma' \longrightarrow \Delta, \Delta'$ doit être un affaiblissement de $\Gamma', \Phi \longrightarrow \Delta'$; mais ceci contredit notre hypothèse que la coupure considérée était non essentielle.

Finalement, le processus termine comme avant, produisant une preuve où les seules coupures sont des coupures non logiques et des coupures avec Ax, et où les coupures avec Ax sont repoussées tout en bas de la preuve. Ces affaiblissements peuvent se résumer en au plus une coupure avec Ax, ce qui montre le théorème. □

Nous en déduisons la notion correspondante de propriété de sous-formule :

Corollaire 32 (Propriété de sous-formule étendue) *Soit S un ensemble de séquents atomiques. Si $\Gamma \longrightarrow \Delta$ est prouvable par une dérivation de* $\mathbf{LK} + S$ *où toutes les coupures sont non logiques, alors toutes les formules apparaissant dans la preuve sont des sous-formules de formules de Γ, Δ, ou apparaissent dans S.*

Preuve : Par récurrence structurelle sur la preuve. Le seul point intéressant est quand nous arrivons sur la conclusion d'une règle de coupure non logique; mais alors toutes les formules, y compris la formule de coupure, apparaissent dans S_∞. Nous montrons que si elles apparaissent dans S_∞, alors elles apparaissent dans S. En effet, soit S_0 égale S, S_1 égale l'ensemble de toutes les conclusions de coupures sur des séquents de S_0, et en général soit S_n l'ensemble de niveau n; nous avons $S_\infty = \bigcup_{n \in \mathbb{N}} S_n$, et une récurrence facile sur n montre que l'ensemble des formules apparaissant dans S_n est exactement l'ensemble des formules apparaissant dans S. □

En particulier, si le séquent à déduire est le séquent vide, nous réobtenons la version de théorie de la preuve du théorème de complétude pour la résolution propositionnelle, d'une façon purement syntaxique :

Théorème 33 (Résolution) *Soit Φ une formule propositionnelle en forme normale conjonctive. Considérons Φ comme un ensemble S de séquents atomiques. Alors Φ est incohérente si et seulement si le séquent vide $\longrightarrow$ est déductible de S et de la règle de coupure Cut uniquement.*

Preuve : Par le théorème 31, si Φ est incohérente, alors nous pouvons dériver le séquent vide de S, en utilisant des coupures non logiques et les règles de **LK** autres que Cut, et en terminant éventuellement par une étape d'affaiblissement. Mais le seul séquent qui s'affaiblisse en le séquent vide est le séquent vide. Par le corollaire 32, cette preuve ne peut utiliser que des formules apparaissant dans S.

Mais ceci implique qu'aucune autre règle que la coupure n'a pu être utilisée, d'où le résultat. □

Ensuite, pour relever ce résultat au cas des clauses du premier ordre, nous utilisons la version syntaxique du théorème de Herbrand et concluons qu'un ensemble de clauses S du premier ordre est incohérent si et seulement s'il existe un ensemble fini d'instances de ces clauses qui est incohérent.

L'unification est nécessaire parce que nous ne savons pas à l'avance de quelles instances nous aurons besoin. Autrement dit, nous remplaçons la règle de coupure habituelle :

$$\frac{\Gamma \longrightarrow \Delta, \Phi \qquad \Gamma', \Phi \longrightarrow \Delta'}{\Gamma, \Gamma' \longrightarrow \Delta, \Delta'}$$

la règle suivante de “coupure paresseuse” :

$$\frac{\Gamma \longrightarrow \Delta, \Phi_1, \ldots, \Phi_n \qquad \Gamma', \Phi'_1, \ldots, \Phi'_{n'} \longrightarrow \Delta'}{\Gamma\sigma, \Gamma'\sigma' \longrightarrow \Delta\sigma, \Delta'\sigma'}$$

où n et n' sont non nuls, et $\Phi_1\sigma = \ldots = \Phi_n\sigma = \Phi'_1\sigma' = \ldots = \Phi'_{n'}\sigma'$. Autrement dit, au lieu de produire toutes les instances des prémisses, et d'en trouver une sur laquelle nous pouvons appliquer la coupure, nous incluons le processus de calcul de ces instances dans la règle elle-même. Nous disons qu'il s'agit d'une règle “paresseuse” parce que nous ne sommes pas obligés de connaître toutes les liaisons de variables à des termes pour pouvoir l'utiliser.

Les considérations sur l'unification de la section 2 s'appliquent toujours, et nous pouvons encore coder cette règle comme la combinaison d'une règle de factorisation et d'une règle de résolution binaire.

La raison pour laquelle nous avons besoin de la factorisation, qui est une règle coûteuse en termes de non-déterminisme dans la recherche de la preuve, acquiert un nouveau sens : elle est l'expression de la règle implicite de contraction de la logique classique. En effet, lorsque nous appliquons la substitution σ à un séquent, et si σ unifie plusieurs formules dans le séquent, alors une étape automatique de contraction est effectuée, qu'il nous faut deviner par factorisation.

D'autres techniques ont aussi une justification en théorie de la preuve. Par exemple, l'effacement des tautologies revient à ignorer les instances de Ax : comme le théorème 31 le montre, nous ne pouvons pas ignorer Ax en général, puisque nous pourrions en avoir besoin lors d'une coupure à l'étape finale de la preuve d'un séquent $\Gamma \longrightarrow \Delta$ de $\mathbf{LK} + S$; mais si Γ et Δ sont vides, nous pouvons l'ignorer, parce que le seul séquent qui s'affaiblisse pour donner le séquent vide est le séquent vide lui-même. La raison pour laquelle l'élimination de tautologies fonctionne, alors, et que nous sommes capable de pousser toutes les coupures avec Ax en bas de la preuve, comme nous l'avons montré dans la preuve du théorème 31.

L'élimination des clauses subsumées est alors en partie justifiée par le même argument. Rappelons qu'une clause $C = \Gamma \longrightarrow \Delta$ subsume C' si et seulement si C' est de la forme $\Gamma', \Gamma\sigma \longrightarrow \Delta\sigma, \Delta'$ pour une certaine substitution σ, c'est-à-dire si C' est une version affaiblie d'une instance de C. L'affaiblissement n'est que la coupure avec Ax, et peut ainsi être poussée en bas de la preuve. Pour expliquer pourquoi les instances de clauses sont inutiles (sauf en tant que facteurs), nous devons recourir au lemme 12, que nous en fait déjà prouvé complètement syntaxiquement. (Autrement dit, contrairement au cas de la complétude de la résolution ordonnée, par exemple, nous n'avons pas eu à recourir à des considérations sémantiques, nous n'avons fait que déplacer des substitutions.)

Finalement, observons que les diverses stratégies, la résolution ordonnée, la résolution sémantique, la résolution linéaire, et ainsi de suite, sont des façons d'exprimer que la coupure sur une formule permute avec la coupure sur n'importe quelle autre formule, ce qui étend le théorème de permutabilité. Ces restrictions de la résolution peuvent alors être considérées comme des restrictions supplémentaires sur la forme des preuves du séquent vide n'utilisant que la coupure, dont le but est de les rendre plus canoniques. Cependant les preuves montrant que nous pouvons permuter les coupures sont compliquées par le fait que nous devons le faire modulo l'application de substitutions. C'est pourquoi les preuves de complétude de la résolution sémantique et de la stratégie ordonnée procédaient par raisonnement sur le cas pleinement instancié (non paresseux, ou clos), et relevaient ensuite le résultat au cas du premier ordre (paresseux).

References

[Bil96] Jean-Paul Billon. The disconnection method — a confluent integration of unification in the analytic framework. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *5th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 110–126, Terrasini, Palermo, Italy, May 1996. Springer Verlag Lecture Notes in Artifical Intelligence 1071.

[CL73] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving.* Computer Science Classics. Academic Press, 1973.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 243–320. Elsevier Science Publishers b.v., 1990.

[JK90] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. Technical report, LRI, CNRS UA 410: Al Khowarizmi, March 1990.

[LP92] Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning*, 9(1):25–42, August 1992.

[MM82] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[Vor94] Andrei Voronkov. Implementing bottom-up procedures with code trees: a case study of forward subsumption. UPMAIL Technical Report 88, Uppsala University, Computing Science Department, 1994. Available at `ftp://ftp.csd.uu.se/pub/papers/reports/0088.ps.gz`.