UNIVERSITE D'AIX - MARSEILLE

THÈSE

présentée

A L'U.E.R DE LUMINY

pour obtenir

LE GRADE DE DOCTEUR DE SPÉCIALITÉ

EN MATHÉMATIQUES APPLIQUÉES

par

Philippe ROUSSEL

DÉFINITION ET TRAITEMENT DE L'ÉGALITÉ FORMELLE EN DEMONSTRATION AUTOMATIQUE

Soutenue le 30 Mai 1972, devant le Jury,

MM. A. ARAGNOL *Président*

A. COLMERAUER
R. KOWALSKI } *Examinateurs*

UNIVERSITE D'AIX - MARSEILLE

THÈSE

présentée

A L'U.E.R DE LUMINY

pour obtenir

LE GRADE DE DOCTEUR DE SPÉCIALITÉ

EN MATHÉMATIQUES APPLIQUÉES

par

Philippe ROUSSEL

DÉFINITION ET TRAITEMENT DE L'ÉGALITÉ FORMELLE EN DEMONSTRATION AUTOMATIQUE

Soutenue le 30 Mai 1972, devant le Jury,

MM.	A. ARAGNOL	*Président*
	A. COLMERAUER	*Examinateurs*
	R. KOWALSKI	

Que Monsieur le Professeur A.ARAGNOL, Président de l'U.E.R de Marseille-Luminy, trouve ici toute ma gratitude pour avoir bien voulu présider le jury de cette thèse malgré ses occupations absorbantes.

Je tiens à remercier chaleureusement Monsieur A.COLMERAUER, Maître de Conférences d'Informatique d'avoir dirigé cette thèse, et pour les nombreuses et fructueuses discussions que nous avons eues ensembles dans le cadre de son équipe.

Que Monsieur R. KOWALSKI, de l'Université d'Edimbourg soit assuré de ma reconnaissance pour les critiques et les conseils qu'il a accepté de formuler à propos de ce travail.

J'aimerais également remercier Monsieur J. TRUDEL de l'Université de Montréal pour m'avoir initié à la démonstration automatique.

Enfin je remercie vivement tous mes amis du Groupe d'Intelligence Artificielle de Luminy pour les nombreux et chaleureux entretiens que nous avons eus ensembles, en particulier B. JOUBERT pour la rapidité de sa rédaction et R. PASERO pour notre constant travail en commun.

à Jacqueline et Romain

TABLE DES MATIERES

ex. dif . . . 71

INTRODUCTION

Depuis l'article publié en 1965 par J.A ROBINSON (ROBINSON-1) sur une méthode nouvelle, dite "resolution" particulièrement efficace, la démonstration automatique n'a cessé de se développer. De nouveaux raffinements ont été créés puis implémentés sur ordinateur. Certains théorèmes ouverts ont même démontrés par des programmes de démonstration automatique (ALLEN et LUCKHAM-1).

Il faut noter cependant que parmi de nombreux problèmes qui restent encore posés dans ce domaine, figure celui du traitement de l'égalité. C'est en effet un outil particulièrement puissant mais qui n'a pas encore, malgré la paramodulation (Robinson et Wos-1), été résolu de manière satisfaisante.

Nous avons été amené à modifier l'axiomatique de ce prédicat de façon à rendre plus efficace son traitement et c'est en fait un prédicat différent: l'égalité formelle que nous étudierons.

La terminologie et les théorèmes de base de la démonstration automatique seront étudiés dans le premier chapitre. Nous parlerons entre autre des notions de clauses, d'interprétations et de substitution. Le théorème de Herbrand, clef de voûte de toute la démonstration automatique servira de base au théorème de resolution de Robinson. Nous parlerons enfin de la notion de stratègie introduite par Kowalski (KOWALSKI-2).

Le second chapitre sera consacré exclusivement à l'égalité formelle. Nous donnerons les axiomes de ce prédicat et la notion d'interprétation injective qui servira de base à tous les théorèmes qui suivront. Nous montrerons en particulier que pour toute méthode complète et "généralisable" (ce qui correspond au "lifting" de la littérature anglo-saxone) on peut trouver une méthode associée dans laquelle est inclu le traitement automatique des axiomes de l'égalité formelle. Plusieurs exemples d'utilisation de l'égalité formelle donneront une idée des avantages qu'il fournit aux démonstrateurs automatiques.

Le dernier chapitre enfin traitera plus spécialement de l'implémentation des méthodes décrites précedemment, sur ordinateur.

Le langage que nous utilisons pour décrire les algorithmes est Algol-W. Ce langage a été choisi pour les possibilités qu'il offre dans le maniement de données formelles comme les chaines, les listes ou les arbres et pour la simplicité et l'efficacité de ses procédures d'entrée-sortie. Le langage est à peu de choses près une extension d'Algol 60. Notons également que le compilateur dont nous disposions était particulièrement efficace tant à l'éxécution qu'à la compilation.

Nous décrirons la façon dont nous codons les clauses (formules) en mémoire ainsi que les algorithmes d'unification, de copie et d'utilisation d'un dictionnaire de liste d'arbres.

La syntaxe du langage dans lequel doit etre decrit un problème posé au programme OEDIPE où sont réunis ces différents algorihmes, les paramètres à fournir en ce qui concerne la stratégie ainsi que la syntaxe des réponses seront explicités.

Deux exemples détaillés seront donnés dans ce dernier chapitre.

Une bibliographie sera donnée en appendice.

CHAPITRE I

NOTIONS GENERALES SUR LA DEMONSTRATION AUTOMATIQUE

L'objet fondamental de la démonstration automatique est de trouver un algorithme général qui, pour un ensemble A quelconque de formules de la logique du 1er ordre, et pour une formule quelconque F donne comme résultat oui ou non suivant que B est déductible de A ou non.

Les logiciens ont montré que ce problème, pour les théories du 1er ordre, est indécidable, i-e que l'algorithme général n'existé pas. Grace au théorème de Herbrand, on montre cependant qu'il existe un algorithme dit de semi-décision qui résoud le problème suivant: pour tout ensemble de formule A, et pour toute formule B, si B est un théorème déductible de A, alors l'algorithme s'arrête au bout d'un nombre fini d'itérations et donne comme résultat une démonstration du théorème. Dans le cas contraire l'algorithme peut boucler et ne jamais s'arrêter.

Nous allons tout d'abord donner quelques rappels de logique sur les notions fondamentales à la démonstration automatique.

I.1. RAPPELS DE LOGIQUE

Démontrer qu'un ensemble de formules $\{F_1,.....,Fn\}$ du 1er ordre entraine logiquement une formule B est équivalent à montrer que la conjonction F_1 et F_2 et...Fn et non B est insatisfaisable c'est à dire qu'il n'existe aucune interprétation qui la satisfait.

Tout les algorithmes utilisés en démonstration automatique testent en fait l'insatisfabilité d'une formule. On a cependant besoin d'écrire cette formule sous formule "clausale" que nous allons expliciter.

Toute formule est équivalente à une formule normale prénexe conjonctive, c'est à dire une formule de la forme PF où P est une suite de quantificateurs et F est une conjonction de formules qui sont elles mêmes des disjonctions de littéraux. On dit alors que P est le préfixe et Q la matrice de la formule.
Exemple :

$\forall x \forall y \exists z \forall u$ $\quad P(x,y)$ et $(Q(y,z,u)$ ou $P(a,b))$ et non $R(z)$

ici

$\forall x \forall y \exists z \forall u$ est le préfixe $P(x,y)$ $Q(y,z,u)$ $P(a,b)$ $R(z)$ sont les littéraux.

On montre alors que l'on peut éliminer tous les quantificateurs existentiels d'une telle formule par l'introduction des nouveaux symboles fonctionnels dits "fonctions de Skölem".
L'algorithme est le suivant:

soit
(1) $\forall x_1 \ldots\ldots \forall x_n \exists z\ P_0\ F$ une formule sous forme normale prénexe conjonctive
où
$\forall x_1 \ldots\ldots \forall x_n \exists z P_0$ est le préfixe, F la matrice. Supposons en outre que

$\forall x_1 \ldots\ldots \forall x_n$ ne contienne aucune occurence du symbole $\exists$. Alors

la formule (1) est équivalente à la suivante:

(2) $\forall x_1 \ldots\ldots \forall x_n\ P_0\ F(z \mid f(x_1, \ldots\ldots x_n))$ où f est un nouveau

symbole fonctionnel d'ordre n n'apparaissant pas dans F et où $F(z \mid f(x_1,\ldots,x_n))$

désigne la formule obtenue à partir de F en remplaçant chaque occurence de z dans F par $f(x_1,\ldots,x_n)$.

En itérant le processus pour tous les symboles existentiels du préfixe on obtient donc une formule équivalente, où n'apparait plus de symbole existentiel.
Exemple :

soit
(1) $\forall x \exists y \forall z$ (P(x) <u>ou</u> Q(y)) <u>et</u> <u>non</u> R(y,z)

La "skolemisation" de cette formule donne:

(2) $\forall x \forall z$ (P(X) <u>ou</u> Q(f(x))) <u>et</u> <u>non</u> R(f(x),z)

Une fois obtenue cette formule peut être remplacée par une formule équivalente en distribuant les quantificateurs universels sur les <u>et</u> puisque $\forall x$(F et G) est équivalente à ($\forall x$ F) et ($\forall x$ G)

On arrive ainsi à une formule sous forme clausale c'est à dire une conjonction de formules qui sont elles-mêmes des dijonctions de littéraux, quantifiées universellement.

Voici un exemple de théorème à démontrer que l'on exprime sous forme clausale:

On suppose que les psychiatres ne psychanalysent que des malades et que quiconque ne se psychanalyse pas lui-même est malade. Montrer alors que tous les psychiatres sont malades.
Nous utiliserons 3 prédicats pour formaliser ce problème en une théorie du 1er ordre:

Ps(x) qu'on interprétera par x est psychiatre.
P(x,y) " " " x psychanalyse y.
Malade(x) " " " x est malade.

Le problème est de montrer que

(1) $\forall x$ (Ps(x)=> Malade(x))

est déductible de:

(2) $\forall x$ (Ps(x)=>($\forall y$ P(x,y)=>Malade(y)))

et

(3) $\forall x$ (non P(x,x))=>Malade(x))

Il faut donc montrer que la formule suivante est insatisfaisable:

(4) ($\forall x$ Ps(x)=>($\forall y$ P(x,y)=>Malade(y))) et ($\forall x$ non P(x,x)=>Malade(x))

et ($\exists x$ (Ps(x) et non Malade(x)))

En écrivant sous formule normale prénexe conjonctive on obtient:

(5) $\exists x \forall y \forall z$ (non Ps(y) ou non P(y,z) ou Malade(z))

et (P(y,y) ou Malade(y)) et (Ps(x)) et (non Malade(x))

Après introduction de la fonction de Skölem A et distribution des quantificateurs :

(6) $\forall y \forall z$ (non Ps(y) ou non P(y,z) ou Malade(z))
et
$\forall y$ (P(y,y)ou Malade(y))
et
Ps(A)
et
non Malade(A)

Il ne reste plus alors qu'à tester l'insatisfabilité de cette formule .

Chacune des disjonctions quantifiée universellement est appelée une clause.

Un ensemble de formules est donc logiquement équivalent à une conjonction de clauses. L'algorithme qui effectue une telle transformation ne figure pas dans notre programme et le soin est laissé à l'utilisateur de formuler le problème sous forme clausale. En fait la plupart des problèmes s'expriment facilement sous cette forme et la principale difficulté ne réside pas,loin de là, dans cette transformation mais plutôt dans la recherche de la preuve de l'insatisfabilité.

I.2. INTERPRETATIONS DE HERBRAND

Clauses

Nous allons donner une définition formelle des clauses sous forme d'une frammaire-context-free. L'alphabet dont on dispose est constitué d'un ensemble de variables V, d'un ensemble de symboles fonctionnels F et d'un ensemble de symboles relationnels (ou symboles de prédicats) R.
A chaque symbole fonctionnel ou relationnel est associé un nombre entier positif ou nul, son ordre (qui correspond au nombre d'arguments de la fonction ou du prédicat).

Les symboles fonctionnels d'ordre 0 sont aussi appelés constantes.

Nous définirons une clause en fait par sa matrice.

Voici les règles permettant de les définir:

<terme>::= <symbole fonctionnel n>(<liste n termes>)|
<symbole fonctionnel 0>|<variable>

<liste n termes >::= <liste n-1 termes> , <termes>

<liste 1 termes>::= <terme>

<formule atomique>::= <symbole relationnel n> (<liste n termes>)|
<symbole relationnel 0>

<littéral>::= <formule atomique>| non <formule atomique>

<clause>::= <conjonction littéraux>

<conjonction littéraux>::= <conjonction littéraux> ou <littéral>|<vide>

<symbole fonctionnel 0> ::= a|b|c|.....

<symbole fonctionnel 1> ::= f|g|h|.....

<variable> ::= x|y|z|.....

<symbole relationnel 0> ::= P|Q|Q|.....

<symbole relationnel 1> ::= S|T|.....

etc.....

Voici un exemple de clause:

P ou non R ou non S(f(g(a))) ou T(h(x))

En fait l'opération ou étant commutative, associative et idempotente il est plus commode de définir une clause non pas comme une disjonction de littéraux mais comme un ensemble de littéraux.

DEFINITION I.2.1 : clause
On appele clause tout ensemble de littéraux.

De plus par commodité d'écriture nous noterons la négation d'une formule atomique A non pas par non A mais par $\overline{A}$.
En outre si L est un littéral $\overline{L}$ désignera le littéral suivant, A étant une formule atomique:

si L = A alors $\overline{L} = \overline{A}$
si L = $\overline{A}$ alors $\overline{L}$ = A

Exemple de clause: { $\overline{P}$, Q(x,y), R(f(x),a)}

Nous donnerons au chapitre III la façon dont une clause est codée en mémoire dans le programme OEDIPE ainsi que l'algorithme de copie d'une clause.

En extrapolant la définition précédente nous considérerons toute conjonction de clauses comme un ensemble de clauses et nous définirons alors les interprétations en fonction de ces conventions. L'avantage de telles représentations est de pouvoir utiliser le langage de la théorie des ensembles pour établir des théorèmes généraux sur l'insatisfabilité des théorie du 1er ordre, plutôt que de manier des interprétations au sens habituel en logique.

REMARQUE
La clause égale à l'ensemble vide sera notée □ et appelée clause v

DEFINITION I.2.2
Une expression est un ensemble de clauses un terme, un littéral ou u clause. Une expression est dite terminale si elle ne contient aucune occurence de variables.

DEFINITION I.2.3 : Univers de Herbrand.

Soit S un ensemble de clauses. L'Univers de Herbrand de S, noté H(est l'ensemble des termes terminaux construits à partir de l'ensemble des symboles fonctionnels de S (ensemble auquel on ajoute la constante A s'il est vide).

Dans l'exemple du paragraphe I la formule (6) pouvait être considérée comm l'ensemble S de clauses suivant:

S={{$\overline{P}$s(y), $\overline{P}$(y,z), Malade(z)},{ P(y,y),Malade(y)},{Ps(A) , $\overline{\text{Malade}}$(

L'univers de Herbrand de S est alors $H(S)=\{A\}$

Si $S= \{\{\overline{P}(f(x),a)\},\{Q(y),R(f(b,y))\}\}$ alors

$H(S)=\{a,b,f(a),f(b),g(a,a),g(b,a),gb,b),f(f(a)),.....\}$.

H(S) est toujours fini ou dénombrable si S est fini ou dénombrable.

Substitutions

Intuitivement une clause C étant quantifiée universellement, si elle est satisfaite par une interprétation, toute clause obtenue en substituant les variables de C par d'autres termes est encore satisfaite par cette interprétation. C'est ce qui nécessite l'introduction de la notion de substitution.

DEFINITION I.2.4 : Substitution

On appele substitution tout ensemble fini, éventuellement vide, σ de couples de la forme x_i/t_i :

$\sigma=\{x_1/t_1, x_2/t_2,....., x_n/t_n\}$ tels que:

(1) $\forall i \in \{1,....n\}$, x_i est une variable et t_i un terme avec $t_i \neq x_i$

(2) $\forall i,j \ \{1,....n\}$ $\quad x_i \neq x_j$

L'ensemble $\{x_1,....,x_n\}$ est l'ensemble des variables de la substitution et sera noté $V(\sigma)$.
Chaque couple x_i/t_i est appelé composante de la substitution

REMARQUE

Dans le cas où σ est vide nous noterons par ε . Dans ce cas $V(\varepsilon)= \emptyset$.

DEFINITION I.2.5

Soit E une expression et $\sigma= \{x_1/t_1,...., x_n/t_n\}$ une substitution.

On définit alors E σ comme étant l'expression obtenue en remplaçant simultanément chaque occurence de x_i par t_i dans l'expression E pour chaque composant x_i/t_i de σ .

Dans le cas où σ est vide E ε=E.

Exemples:

(1) $E= \{P(x,f(x)), P(f(a),y)\}$ $\quad \sigma= \{x/a,y/f(x),z/x\}$

$E\sigma=\{ P(a,f(a)),P(f(a),f(x))\}$

(2) $E= \{P(x,y),P(f(y),a)\}$ $\quad \sigma =\{ x/f(a),y/a\}$

$E\sigma= \{P(f(a),a)\}$

(3) $E= f(x)$ $\quad \sigma =\{y/z\}$ $\quad E\sigma =f(x)$

DEFINITION I.2.6 : Instance

Soient E et E' deux expressions. On dit que E' est une instance de E ssiilexiste une substitution σ telle que

$$E'=E\sigma$$

Exemple:

P(a,f(a)) est une instance de P(x,f(y))

Nous verrons au paragraphe 3 les principales propriétés des substitutions, dont celles qui serviront pour le chapitre II. Les quelques définitions que nous venons de donner nous permettent cependant d'introduire la notion d'interprétation de Herbrand.

Iterprétations de Herbrand

DEFINITION I.2.7 base de Herbrand

Soit S un ensemble de clauses. On appelle base de Herbrand de S, notée $\hat{H}$ (s), l'ensemble des instances terminales des atomes ayant une occurence dans S.

NOTATION

L étant un littéral quelconque nous noterons $|L|$ l'atome tel que $|L| =L$ ou $|\overline{L}|=L$. On a alors $|\overline{L}|=|L|$ et $|L\sigma|=|L|\sigma$ pour toute substitution σ .

Donc on peut dire que $A\in \hat{H}(S)$ si et seulement si il existe un littéral L ayant une occurence dans S,et une substitution σ telles que:

$\sigma =\{x_1/t_1,\ldots., x_n/t_n\}$ avec $t_i \in H$ $\quad i\in \{1,\ldots.,n\}$

$L\sigma$ est terminale et $A=|L\sigma|$

Soit

$$S=\{\{ P(n,f(x))\},\{ \overline{Q}(a),Q(f(a))\},\{(y)\}\}$$

$$\hat{H}(s)= \{Q(a),R(a), Q(f(a)),R(f(a)),P(f(a)))),.....\}$$

DEFINITION I.2.8: interprétation de Herbrand

Soit S un ensemble de clauses. On appelle <u>interprétation de Herbrand</u> de S, tout ensemble I de littéraux terminaux tels que

(1) $L \in I \Rightarrow |L| \in \hat{H}(S)$

(2) $L \in I \Rightarrow \overline{L} \notin I$

Une telle interprétation I détermine en fait une interprétation unique φ dans le sens usuel, de la manière suivante.
L'univers de φ est l'univers de Herbrand H (considéré comme un ensemble de mots)

Pour tout symbole fonctionnel f d'ordre n, de S, $\varphi(f)$ est une application de H^2 dans H définie par :

$$\forall t_1,....,t_n \in H \qquad \varphi(f)(t_1,....,t_n) = f(t_1,....., t_n)$$

Pour tout symbole relationnel P d'ordre n de S, $\varphi(P)$ est une application de H^n dans {0,1} défini par

$$\forall t_1,....,t_n \in H \qquad \varphi(P)\ (t_1,...t_n)=1 \quad \text{ssi} \quad P(t_1,....,t_n) \in \hat{H}$$

En tenant compte de ces remarques on peut donc définir la notion de satisfabilité pour les interprétations de Herbrand. Habituellement une formule quantifiée universellement est satisfaite par une interprétation φ ssi toutes ses instances terminales sont satisfaites par φ. Une conjonction de formule est satisfaite par φ ssi toutes les formules le sont. On peut donc définir en conséquence la satisfabilité au sens des interprétations de Herbrand.

DEFINITION I.2.9 : satisfabilité

Soit S un ensemble de clauses et I une interprétation de Herbrand de S.

On dit qu'une clause $C \in S$ est <u>satisfaite</u> par I ssi toute instance terminale $C\sigma$ de C est telle que $C\sigma \cap I \neq \phi$.

L'ensemble de clause S est <u>satisfait</u> par I ssi toute clause de C est satisfait par I.

Un ensemble de clauses S est satisfaisable ssi il existe une interprétation de Herbrand qui le satisfait et il est <u>insatisfaisable</u> dans le cas contraire.

Notons que cette définition est bien cohérente avec la définition habituelle.

Soit $C'=C\sigma$ une instance terminale d'une clause $C \in S$, I une interprétation de Herbrand, φ l'interprétation dans H(S) (l'interprétation habituelle) associée à I.

Si $C'=\{L_1,\ldots,L_n\}$ où $L_1,\ldots L_n$ sont des littéraux terminaux, alors par définition de φ (A ou B) on a :

$$\varphi(C')= \text{Max}\{ \varphi(L_1),\ldots,\ \varphi(L_n)\} \quad \text{donc} \quad \varphi(C')=1$$

ssi $\exists i \in \{1,\ldots,n\}$ tel que $\varphi(L_i)=1$

donc

$\varphi(C')=1$ ssi $\exists L \in C'$ tel que $\varphi(L)=1$ (ie-$L \in I$)

donc $\varphi(C')=1$ ssi $C' \cap I \neq \phi$

On étend la définition au cas où $C= \square$ auquel cas $C \cap I=\phi$ pour toute interprétation de Herbrand I (donc $\square$ est toujours insatisfaisable).

En ce qui concerne un ensemble de clauses S qui représente une conjonction de formule, la définition s'accorde également à la définition habituelle.

En ce qui concerne la notion de satisfabilité il faut noter que pour toute interprétation φ il existe une interprétation de Herbrand I qui satisfait les mêmes clauses que φ.

REMARQUE

Un ensemble de clauses S est insatisfaisable ssi il existe un ensemble d'instances terminales de clauses de S insatisfaisable.

En effet si S est insatisfaisable alors l'ensemble de toutes les instances terminales de clauses de S est lui aussi insatisfaisable.

Si S' est un ensemble de clauses instances terminales de clauses de S et si S' est insatisfaisable alors $\forall$ I interprétation de Herbrand, $\exists$ C' $\in$ S' telle que C' n'est pas satisfaisable par I. Donc si $C'= C\sigma$ avec $C \in S$, C n'est pas satisfaisable par I donc S non plus.
Donc S est insatisfaisable.

Le théorème de Herbrand est une restriction à cette remarque.

THEOREME I.2.1: Herbrand

Un ensemble de clause S est insatisfaisable ssi il existe un ensemble fini S' d'instances terminales de clauses de S, qui est insatisfaisable.

Le détail de la démonstration figure dans ROBINSON-1, HAYES et KOWALSKI-1

Ce théorème permet déjà d'imaginer un algorithme pour tester l'insatisfabilité d'un ensemble fini de clauses S. L'ensemble des instances terminales de clauses de S étant dénombrable récursivement, on peut générer cet ensemble par un algorithme. Pour chaque sous-ensemble fini ainsi généré on peut tester son inconsistence par un algorithme puisque il est constitué de clauses terminales en nombre fini.

En fait une telle méthode est impraticable et c'est la notion de plus grand unifieur (P.G.U) introduite par Robinson qui va permettre de trouver un algorithme plus efficace.

I.3 PROPRIETES DES SUBSTITUTIONS

Définitions et propriétés générales

DEFINITION I.3.1: variables

Soit E une expression quelconque. L'ensemble des variables de E, noté V(E), est l'ensemble des variables ayant une occurence dans E.

NOTATION I.3.1: Union dijointe d'ensembles.

Soient E, F et G trois ensembles. On définit l'union disjointe de 2 ensembles, que l'on note $\dot{\cup}$ par:

$$E = F \dot{\cup} G \text{ ssi } F \cap G = \phi \text{ et } E = F \cup G$$

DEFINITION I.3.2: Composition de substitutions

Soient σ et ρ deux substitutions. On définit la composée de σ et de ρ, que l'on note par $\sigma\rho$, de la façon suivante:

$$\sigma.\rho = \{x/t.\rho \text{ tels que } x/t \in \sigma \text{ et } x \neq t.\rho \} \dot{\cup} \{x/t \in \rho \text{ tels que } x \notin V(\sigma)\}$$

Donnons des exemples:

(1) $\sigma = \{x/a, y/f(x)\}$ $\rho = \{x/y, z/b\}$ $\sigma.\rho = \{x/a, y/f(y), z/b\}$

(2) $\sigma = \{x/y, y/x\}$ $\rho = \{y/x, x/y\}$ $\sigma.\rho = \varepsilon$ (substitutions vide)

Notons que $\sigma\rho$ est bien une substitution puisque réunion disjointe de deux substitution qui n'ont pas de variables en commun.

PROPOSITION I.3.1

Pour toutes substitutions σ, ρ, μ et pour toutes expressions E et E' on a:

(1) $\sigma.(\rho.\mu)=(\sigma.\rho).\mu$ associativité

(2) $\sigma\varepsilon = \varepsilon.\sigma = \sigma$ ε élément neutre

(3) $(E.\sigma).\mu = E.(\sigma.\mu)$

(4) si $E \subset E'$ alors $E\sigma \subset E'\sigma$

(5) $(E \cup E')\sigma = E\sigma \cup E'\sigma$

Nous ne donnerons pas les démonstrations de ces quelques propriétés qui se trouvent dans l'article de Robinson (ROBINSON-1)

Pour les besoins du chapitre II nous aurons besoin de propriétés supplémentaires que voici:

PROPOSITION I.3.2

Soient ρ et σ 2 substitutions et E une expression. Alors

$$(\forall x \in V(E) \quad x.\sigma = x.\rho) \Longleftrightarrow E\sigma = E\rho$$

En effet soit $x \in V(E)$. Alors x est substitué par un même terme dans E par σ et ρ.

PROPOSITION I.3.3: Substitution minimale.

Soient E une expression et D une instance de E. Alors il existe une substitution unique σ_0 (dite substitution minimale) telle que :

(1) $\forall \sigma \subset E.\sigma=D \Longleftrightarrow (\sigma_0 \cup \sigma$ et $V(\sigma_0) = V(E) \cap V(\sigma))$

(2) $E\sigma_0=D$

(3) $V(\sigma_0) \subset V(E)$

DEMONSTRATION

- D étant une instance de E il existe σ_1 telle que $E\sigma_1=D$.

- Soit $\sigma_0=\{x/t \in \sigma_1 \ / \ x \in V(E)\}$. On a donc (3) $V(\sigma_0) \subset V(E)$

De plus d'après I.3.2 puisque $\forall x \in V(E)$ $x.\sigma_1=x.\sigma_0$ on a

$$E\sigma_0=E\sigma_1=D \quad (2)$$

- Soit σ telle que $E\sigma=D$. On a alors puisque $E\sigma=E\sigma_0$ toujours d'après I.3.2 : $\forall x \in V(E) \quad x.\sigma=x.\sigma_0$

donc puisque $V(\sigma_0) \subset V(E) \quad \forall x/t \in \sigma_0 \quad x.\sigma_0=t=x.\sigma_1$

donc $x/t \in \sigma_1$ donc $\sigma_0 = \sigma$

De plus $V(\sigma) \cap V(E)=V(\sigma_0) \cap V(E)=V(\sigma_0)$

PROPOSITION I.3.4

Soit $\sigma=\sigma_1 \dot{\cup} \sigma_2$ une substitution, E une expression telle que $V(\sigma_1) \cap V(E)= \phi$.
Alors

$$E\sigma=E\sigma_2$$

DEMONSTRATION

En effet soit σ_0 la substitution minimale telle que $E\sigma_0=E\sigma$
On a alors d'après I.3.3

$$V(\sigma_0)=V(E) \cap V(\sigma)=V(E) \cap (V(\sigma_1) \dot{\cup} V(\sigma_2))=V(E) \cap V(\sigma_2)$$

donc

$\sigma_0=\{x/t \in \sigma_2 / x \in V(E)\}$ donc d'après I.3.2 $E\sigma_0=E\sigma_2$

PROPOSITION I.3.5

Soient $\sigma=\sigma_1 \dot{\cup} \sigma_2$ et ρ deux substitutions telle que σ_2 soit une substitution terminale (c'est à dire que $x/t \in \sigma_2 \Rightarrow$ t terminal)
Alors

$$\sigma_2 \subset \sigma.\rho$$

DEMONSTRATION

D'après la définition de la composition on a:

$\sigma.\rho=\{ x/t.\rho \quad t.\rho \text{ tel que } (t.\rho \neq x) \text{ et } x/t \in \sigma_1\} \cup$
$\{x/t.\rho \quad t.\rho \neq x \quad \text{et} \quad x/t \in \sigma_2\} \cup$
$\{x/t \in \rho / x \notin V(\sigma_1 \dot{\cup} \sigma_2)\}$

Or $\{x/t.\rho \text{ tels que } x/t \in \sigma_2 \text{ et } t.\rho \neq x\}=\{x/t \text{ tels que } x/t \in \sigma_2\}=\sigma_2$
donc

$\sigma_2 \subset \sigma.\rho$

PROPOSITION I.3.6

Soit $C\sigma=D$ une instance terminale de C. Alors:

$\forall \rho \qquad \sigma \subset \rho \Rightarrow C\rho=D$

DEMONSTRATION

En effet puisque $C\sigma$ est une instance terminale de C toutes les variables de C ont été substituées par des termes terminaux donc $V(\sigma) \supset V(C)$ donc d'après I.3.2 comme $\sigma \subset \rho$ on a

$$C\rho = C\sigma$$

PROPOSITION I.3.7

Soit $E \subseteq F$ deux expression, E' une instance terminale de E.
Alors:

$\exists\ \sigma$ telle que $F\sigma$ est une instance terminale de F

et

$$E\sigma = E'$$

DEMONSTRATION

Soit σ la substitution minimale telle que $E' = E\sigma_0$.
Soit V_1 l'ensemble des variables de F n'apparaissant pas dans E

$$V_1 = V(F) \cap (V(E))$$

Soit

$\sigma_1 = \{x/a$ avec $x \in V_1\}$ et a un symbole constant quelconque

Soit

$\sigma = \sigma_1 \cup \sigma_0$ $\quad E\sigma_1 = E\sigma_0 = E'$ d'après I.3.6 puisque $\sigma_0 \subset \sigma$

De plus

$V(\sigma) = V(\sigma_1) \cup V(\sigma_0) = (V(F) \cap (V(E)) \cup V(\sigma_0)$.

or

$V(\sigma_0) = V(E)$ puisque E' est terminale .

Donc

$V(\sigma) = V(F)$ et comme σ est une substitution terminale $F\sigma$ est terminale.

Plus grands unifieurs

Un des principaux outils de la resolution est l'utilisation des unifieurs. Il permet de minimiser le nombre d'instance de clauses utilisées dans les algorithmes. Il a été introduit par Robinson (ROBINSON-1).

DEFINITION I.3.3: Unifieurs

Soit A un ensemble d'expressions, et σ une substitution.
On dit que σ est un unifieur de A ssi $A\sigma$ est un singleton.
Dans le cas ou A admet un unifieur on dit que A est unifiable.

Exemple:

$A = \{P(x,e,x),\ P(e,x,x)\}$ alors $\sigma = \{x/e\}$ est un unifieur de A

car

$A\sigma = \{P(e,e,e)\}$ donc A est unifiable .

Mais $B=\{x,f(x)\}$ n'est pas unifiable.

THEOREME I.3.1 Plus grand unifieur

Soit A un ensemble non vide fini d'expression. Si A est unifiable alors:

il existe un unifieur σ_A (appelé plus grand unifieur de A et noté p.g.u) de A tel que :

Pour tout unifieur Θ de A il existe σ tel que $\Theta = \sigma_A.\sigma$

Nous allons donner en fait l'algorithme qui permet de construire σ_0 La preuve que la substitution trouvée répond au théorème se trouve dans ROBINSON-1.

Soit $A=\{E_1,.....,E_n\}$ un ensemble fini d'expressions (termes ou littéraux)

L'ensemble de desagrement de A est l'ensemble des expressions $e_1,.....,e_n$ qui sont les sous expressions de $E_1,.....E_n$ commençant au premier symbole où $E_1,....E_n$ différent.

Exemple:

$A= \{P(x,e,x), P(x,f(x,y),z)\} P(x,g(a),b)\}$

L'ensemble de désagrément de A est $\{e,f(x,y),g(a)\}$

Si Θ est un unifieur de A,A n'étant pas un singleton, Θ unifie son ensemble de désagrément.

L'agorithme permettant de calculer σ_A est le suivant:

Etat1: $\sigma_0 \leftarrow \varepsilon$ $K \leftarrow 0$ Aller à l'état 2 .

Etat2: Si $A\sigma_k$ n'est pas singleton aller à état 3, sinon $\sigma_A \leftarrow \sigma_K$ aller à FIN

Etat3: Supposons l'ensemble de désagrément de $A\sigma_K$ (non vide) orconné lexico graphiquement, les variables ayant l'ordre le plus petit parmi les symboles.

Soient t_1 et t_2 les deux premier termes de l'ensemble de désagrément de $A\sigma_K$

a) Si t_1 est une variable n'ayant pas d'occurences dans t_2 alors:

$\sigma_{K+1} \leftarrow \sigma_K . \{t_1/t_2\}$

$K \leftarrow K+1$

Aller à 2

b) Sinon aller à FIN

On montre alors que l'algorithme s'arrête en un nombre fini d'étapes et que si l'arrêt est produit à l'état 3 alors A n'est pas unifiable.
Par contre si l'arrêt se produit en 2 alors σ_A est le plus grand unifieur cherché.

Nous donnerons l'algorithme écrit en Algol-W qui permet de calculer le p.g.u d'un ensemble de deux expressions (Chapitre III)

Très souvent nous parlerons d'unifieur de t_1 et t_2 au lieu de parler d'unifieur de l'ensemble $\{t_1, t_2\}$.

PROPOSITION I.3.8
Soit σ^0 un p.g.u d'un ensemble d'expressions A.
Alors

$$\forall x/t \in \sigma_0 \qquad V(t) \cap V(\sigma_0) = \phi$$

c'est à dire que tous les termes intervenant dans la substitution n'ont pas de variables communes avec la substitution.

DEMONSTRATION
D'après la constitution d'un p.g.u de A on a

$$\sigma = \sigma_1 . \sigma_2 ..... \sigma_N$$

En outre

$$\forall K \in \{1,.....N\} \quad \sigma_k \text{ est de la forme} \{x_k/t_k\} \text{ où } x_k \notin V(t_k)$$

Montrons la proposition I.3.8 par récurrence sur le nombre N :

$N=1 \rightarrow \quad \sigma_1 = \{x_1/t_1\}$ avec $V(t_1) \not\ni x_1$ donc $V(t_1) \cap V(\sigma) = \phi$

$K+1 \rightarrow \quad \sigma_{k+1} = \sigma_k . \{x_{K+1}/t_{K+1}\}$ c'est à dire $\sigma_{K+1} = \sigma_K . \sigma$ avec $\sigma = \{x_{K+1}/t_{K+1}\}$

Or l'ensemble de désagrément de A_{σ_k} ne contient plus les variables $x_1, ... x_k$ puisque $\forall x/t \in \sigma_k \quad V(t) \cap V(\sigma_k) = \phi$.
donc

$$V(t_{K+1}) \cap V(\sigma_K) = \phi$$

et

σ_{K+1}= {x/t.σ tels que x/t $\in \sigma_K$} $\cup$ {x_{K+1}/t_{K+1}}

$V(t) \cap V(\sigma_k) = \phi$ donc la seule variable substituée dans t est éventuellement x_{K+1} qui est remplacée par t_{K+1}. Or $V(t_{K+1}) \cap V(\sigma_{K+1}) = \phi$ donc $V(t.\sigma) \cap V(\sigma_{K+1}) = \phi$

Voici en fait une des raisons de la puissance de cet outil qu'est le p.g.u dans la démonstration automatique.

Soient deux clauses: $C=D \cup \{L\}$ et $C'=D' \cup \{\overline{L'}\}$. Ces clauses impliquent logiquement toutes leurs instances.
Supposons que L et L' soient unifiables, σ un unifieur: $L\sigma=L'\sigma$
on a donc

$$C\sigma = D\sigma \cup \{L\sigma\} \qquad C'\sigma = D'\sigma \cup \{\overline{L'}\sigma\}$$

Par la règle de coupure de la logique on peut déduire logiquement de C et C' la clause

$$E=D\sigma \cup D'\sigma$$

Si l'on prend pour unifieur σ, le p.g.u de L et L' la clause E obtenue est la clause la plus générale que l'on peut obtenir de C et C' par coupure et instantiation.

Notons qu'un p.g.u d'un ensemble A n'est pas unique puisqu'il est défini à un ordre lexicographique près.

De plus si σ est un p.g.u de A, toute instance de A qui est un singleton est lui-même une instance de Aσ .

Donnons enfin un exemple de p.g.U:

A= {P(x,y,u), P(z,x,x)}

σ_A={u/x, y/x, z/x} $\qquad A\sigma_A$= {P(x,x,x)}

Un autre p.g.u possible est:

σ'_A={x/z, y/z,u/z} $\qquad A\sigma'_A$={P(z,z,z)} variante de $A\sigma_A$

I.4 RESOLUTION

La méthode de resolution est une méthode de déduction qui permet, étant donné un système de clauses inconsistant de générer la clause vide à partir de ces clauses.

Une méthode de déduction est consistente si les clauses déduites d'un ensemble de clauses par cette méthode sont logiquement déductible de l'ensemble de départ.

Elle est complète si à partir d'un ensemble inconsistant en peut générer la clause vide.

Résolution d'un ensemble de clauses terminales

DEFINITION I.4.1

Soient C_1 et C_2 deux clauses terminales. On dit que C est une resolvante de C_1 et C_2 ssi:

il existe un littéral L tel que $L \in C_1$ et $\overline{L} \in C_2$ et

$$C=(C_1-\{L\}) \cup (C_2-\{\overline{L}\})$$

Dans le cas de clauses terminales la resolution n'est autre que la règle de "coupure" des logiciens.
Donnons un exemple:

$C_1=\{L,\overline{P},Q(a),\overline{R}(b,c)\}$ $\quad C_2=\{L,R(a,b),\overline{Q}(a)\}$

C_1 et C_2 admettent comme resolvante $C=\{L,\overline{P},\overline{R}(b,c),R(a,b)\}$

DEFINITION I.4.2

Une dérivation terminale $\mathcal{D}$ d'une clause C, à partir d'un ensemble de clauses terminales S est une suite de Clauses

$$\mathcal{D}=(C_1,.....,C_n)$$

telle que $\forall i \in \{1,...,n\}$. $C_i \in$ S ou C_i est une resolvante de C_j et C_k avec j et k < i , et $C_n=C$.

exemple:

Soit $S=\{C_1,C_2,C_3,C_4,C_5\}$ avec $C_1=\{\overline{L}_1,L_2\}$ $C_2=\{\overline{L}_2,L_3\}$ $C_3=\{\overline{L}_3,L_1\}$

$C_4=\{L_1,L_2,L_3\}$ $C_5=\{\overline{L}_1,\overline{L}_2,\overline{L}_3\}$

Soit $\underset{\sim}{D}=(C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9,)$ avec

$C_6=\{\overline{L}_1, L_3\}$ resolvante de C_1 et C_2

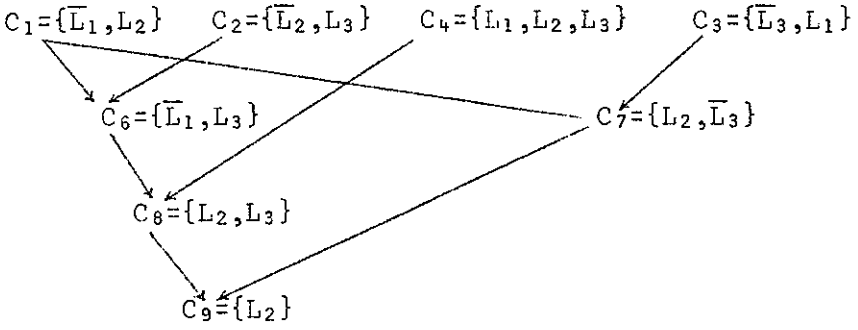
$C_7=\{L_2, \overline{L}_3\}$ resolvante de C_3 et C_1

$C_8=\{L_2, L_3\}$ resolvante de C_6 et C_4

$C_9=\{L_2\}$ resolvante de C_8 et C_7

Alors $\underset{\sim}{D}$ est une dérivation de la clause $\{L_2\}$ à partir de S

On peut representer cette dérivation par le graphe suivant:



Ici chaque noeud est étiquetté par une clause.

DEFINITION I.4.3

On appelle refutation toute dérivation de la clause vide.

Le théorème de Robinson établit qu'en fait la resolution est une méthode complète et consistente.

THEROEME I.4.1 (Robinson)

Soit S un ensemble fini de clauses terminales. Alors:

S est instisfaisable ssi il existe une refutation de S

Ce théorème permet de construire un algorithme "relativament" efficace pour tester l'inconsistence d'un ensemble de clauses S. Il suffit de générer toutes les déductions de S (qui sont en nombre dénombrable) jusqu'à trouver la clause vide.

En fait l'algorithme (pour qu'il soit complet) doit faire intervenir la notion de stratégie, que nous verrons au paragraphe I.6, introduite par Kowalski (HAYES et KOWALSKI-1)

La stratégie la plus simple consiste à générer les resolvantes de premier "niveau", puis de deuxième "niveau" etc.....(ceci pour un ensemble fini de clauses).

Nous montrerons que ce théorème est encore vrai si S est infini.

Resolution dans le cas général

Le théorème de Herbrand nous assure que si en ensemble de clauses S est insatisfaisable, il existe un ensemble S' fini d'instances terminales de clauses de S qui est lui aussi insatisfaisable. Donc d'après Robinson il existe une refutation à partir de cet ensemble.
On définit alors la resolution dans le cas général de façon à "généraliser" le cas terminal c'est à dire que toute resolvante de 2 instances terminales de 2 clauses C_1 et C_2 est une instance terminale d'une resolvante (générale) de C_1 et C_2 .
On sait alors étant donné S et S' construire une refutation qui "généralise" la refutation à partir de S'. Cette refutation est une refutation à partir de S.

On est obligé dans la définition des dérivations générales d'introduire la notion de facteur et la notion de resolvante, c'est à dire deux règles d'inférence au lieu d'une dans le cas terminal.

DEFINITION I.4.4

On appelle variante d'une clause C toute clause D telle que il existe deux substitutions σ_1 et σ_2 avec

$$D=C\sigma_1 \quad \text{et} \quad C=D\sigma_2$$

Deux variantes ne différent en fait que par un changement des noms de variables (deux variables distinctes étant substituées par des variables distinctes).

DEFINITION I.4.5

On appelle facteur de base d'une clause C toute clause $C\sigma$, où σ est un p.g.u de 2 littéraux de C, ou toute variante de C.
On appelle facteur de C tout facteur de base d'un facteur de C, ou tout facteur de base C.

Exemples:
$C=\{P(x,y,z),\ P(a,x,x),\ P(u,a,a)\}$ admet comme facteurs lui-même et:

$C_1=\{P(a,a,a),\ P(u,a,a)\}$ $C_2=\{P(x,a,a),P(a,x,x)\}$

$C_3=\{P(a,y,z),\ P(a,a,a)\}$

$C_4=\{P(a,a,a)\}$ ainsi que toutes leurs variantes .

DEFINITION I.4.6

Soient deux clauses C_1 et C_2 avec $F_1 \subset C_1$ et $\overline{F}_2 \subset C_2$, C_1 et C_2 n'ayant pas de variables communes.
Soient $F_1 \cup F_2$ unifiables, σ un de ses P.g.u. Alors la clause

$C=(C_1-\{F_1\})\sigma \cup (C_2-\{\overline{F}_2\})\sigma$ est une resolvante de C .

C_1 et C_2 sont les parents de C, F_1 et $\overline{F}_2$ les littéraux résolus.

Exemple:

$C_1=\{P(x_1,x_2,a),\ Q(x_2,z,b)\}$ et $C_2=\{\overline{P}(a,b,z'),\ \overline{P}(b,c,z')\}$

admettent comme résolvantes:

$C=\{Q(b,z,b),\ \overline{P}(b,c,a)\}$ $\sigma=\{x_1/a,x_2/b,z'/a\}$

et

$C'=\{Q(c,z,b),\ P(a,b,a)\}$ $\sigma=\{x_1/b,x_2/c,z'/a\}$

Cette définition est une extension du cas terminal puisqu'alors

$$\sigma=\varepsilon\ (F_1 \overline{F}_2)$$

Puisque

$C_1 \models C_1\sigma$ et $C_2 \models C_2\sigma$ et que C est déduite de $C_1\sigma$ et $C_2\sigma$ par coupure C_1 et C_2 impliquent logiquement C.

La resolution est une règle d'inférence consistante.

DEFINITION I.4.7

On appelle dérivation $\underset{\sim}{D}$ d'une clause D à partir d'un ensemble de clause S, toute suite $\underset{\sim}{D}=(C_1,.....,C_n)$ telle que:

(i) $C_n=D$

(ii) $\forall\ i\ \in\{1,....n\}$ $\begin{cases} -C_i \text{ est un facteur d'une clause } C \quad S \in \text{ ou} \\ -C_i \text{ est un facteur d'une resolvante de 2 clauses } C_j \text{ et } C_k \text{ avec } j,k \prec i \end{cases}$

Une réfutation est une dérivation de la clause vide.

Donnons un exemple de refutation (problème des psychiatres du chapitre I).

$S=\{C_1,C_2,C_3,C_4\}$

$C_1=\{\overline{Ps(y)},\ \overline{P(y,z)},\ Malade(z)\}$ $C_2=\{P(y,y),\ Malade(y)\}$ $C_3=\{Ps(A)\}$

$C_4=\{\overline{Malade}(A)\}$

La suite $\underset{\sim}{D}=(C_1,C_2,C_3,C_4,C_5,C_6,C_7,C_8)$ est une réfutation à partir de S

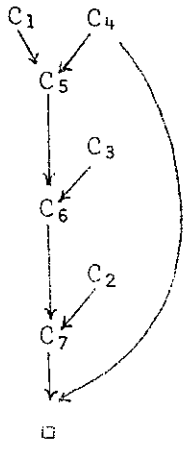
$C_5=\{\overline{Ps(x)},\ \overline{P}(x,A)\}$ resolvante de variantes de C_1 et C_4

$C_6=\{\overline{P}(A,A)\}$ resolvante de C_5 et C_3

$C_7=\{Malade(A)\}$ resolavante de C_2 et C_6

$C_8=\square$ resolvante de C_7 et C_4

Cette refutation peut être représentée par le graphe



Le second théorème de Robinson est une extension du premier au cas général:

THEOREME I.4.2 (Robinson)
Soit S un ensemble de clauses. Alors
S est insatisfaisable ssi il existe une réfutation à partir de S

De même que dans le cas terminal ce théorème permet de construire des algorithmes permettant de générer la clause vide à partir d'un ensemble de clauses inconsistant.
Notons deux règles d'inférence sont utilisées: resolution et factorisation et que l'ensemble S peut être infini. En fait pour que l'algorithme puisse être complet il faudrait générer toutes les déductions à partir de S jusqu'à trouver une réfutation.
Entre autres toutes les variantes d'une clause (puisque facteurs de cette clause) devraient être produites.
Il est évident que celà n'est pas nécessaire et qu'en fait une clause peut être considérée comme l'ensemble de ses instances.
C'est ceci qui motive l'introduction de la notion de classe de clause

Classes de clauses

PROPOSITION I.4.1
La relation suivante: C_1 et C_2 sont variantes l'une de l'autre, est une relation d'équivalence sur l'ensemble des clauses d'une théorie . Nous la noterons $\sim$.

DEFINITION I.4.8
La classe d'une clause est l'ensemble des variantes de cette clause. On a alors $\square = \{\square\}$ et $\dot{C}=\{C\}$ pour toute clause C terminale.

PROPOSITION I.4.2
Soient C'_1 et C'_2 deux variantes de C_1 et C_2,C_1 et C_2(resp (resp C'_1 et C'_2) n'ayant pas de variables en commun.Alors toute resolvante de C_1 et C_2est une variante d'une resolvante de C'_1 et C'_2 .

DEMONSTRATION

En effet soit $C_1=D_1 \dot{U} \{L\}$ et $C_2=D_2 \dot{U} \{\overline{L}\}$ avec L_1 et L_2 ayant σ comme p.g.u . C'_1 et C'_2 étant des instances de C_1 et C_2 $C'_1=C_1\sigma_1$ et $C'_2=C_2\sigma_2$. σ_1 et σ_2 les substitutions minimum, n'ayant pas de variables communes soit $\sigma=\sigma_1 \dot{U} \sigma_2$.on a alors $C'_1=C_1\sigma$ et $C'_2=C_2\sigma$.

soit

$C=(D_1 \cup D_2)\sigma_0$ une resolvante de C_1 et C_2 où σ_0 est un p.g.u de L_1 et L_2 .

Soient $L'_1=L_1\sigma$ et $L'_2=L_2\sigma$, σ étant une substitution qui ne fait que changer le nom des variables (en associant des noms différents à des variables distinctes) L'_1 et L'_2 sont eux aussi unifiables avec σ'_0 comme p.g.u. On alors

$L'_1\sigma'_0=L'_2\sigma'_0$ donc $L_1\sigma\sigma'_0=L_2\sigma\sigma'_0$ donc $\sigma\sigma'_0=\sigma_0.K$

puisque σ_0 est un p.g.u de L_1 et L_2.

Soient $D'_1=D_1\sigma$ et $D'_2=D_2\sigma$, on a $C'_1=D'_1 \dot{U} \{L'_1\}$ et $C'_2=D'_2 \cup \{L'_2\}$ donc $C'=(D'_1 \cup D'_2)\sigma'_0$ est une resolvante de C'_1 et C'_2 . De plus $C'=(D_1\sigma \cup D_2\sigma)\sigma'_0=(D_1 \cup D_2)\sigma\sigma'_0$

donc $C'=(D_1 \cup D_2)\sigma_0.K=C.K$

On montrerait de même que C est une instance de C' donc que C et C' sont variantes.

On peut alors définir la resolvante de 2 classes:

DEFINITION I.4.9

Soient $\dot{C}_1$ et $\dot{C}_2$ deux classes de clauses. On appelle classe resolvante de $\dot{C}_1$ et $\dot{C}_2$ toute classe d'une resolvante de 2 clauses de $\dot{C}_1$ et $\dot{C}_2$.

PROPOSITION I.4.3

Le nombre de resolvantes de deux classes est fini

En effet soient $\dot{C}_1$ et $\dot{C}_2$ deux classes de clauses. Soient deux variantes C'_1 et C'_2 de C_1 et C_2, sans variables communes. Les resolvantes de $\dot{C}_1$ et $\dot{C}_2$ sont les classes des resolvantes de C'_1 et C'_2 qui sont en nombre fini:

Comme toute variante d'un facteur d'une clause est encore un facteur de cette clause on peut définir les facteurs d'une classe de la façon suivante:

DEFINITION I.4.10

Soit $\dot{C}$ une classe de clauses.
On appelle facteur de $\dot{C}$ toute classe d'un facteur de C.

PROPOSITION I.4.4

Le nombre de facteurs d'une classe est fini

En effet le nombre de facteurs de C obtenus par factorisation de base mais non par variantes, est fini.

On peut définir alors une dérivation de classes de la même façon qu'une dérivation de clauses.

DEFINITION I.4.11

Une dérivation $\underset{\sim}{\dot{D}}$ de classes à partir d'un ensemble de classe $\dot{S}$ est une suite de classes $\underset{\sim}{\dot{D}}=(\dot{C}_1,\ldots \dot{C}_n)$ telles que $\forall i$

$$\begin{cases} \dot{C}_i \in S \\ \text{ou} \\ \dot{C}_i \text{ est un facteur d'une classe } \dot{C}_j,\ j<K. \\ \text{ou} \\ \dot{C}_i \text{ est une resolvante de 2 classes } \dot{C}_j \text{ et } \dot{C}_K \text{ avec } j,K< i \end{cases}$$

Le principal intéret de l'introduction de cette notion de classes est que le nombre de classes produites à partir de 2 classes par resolution ou factorisation est fini.

D'autre part l'implémentation sur ordinateur d'une clause est plus proche de la notion de classes.
En fait par commodité d'écriture nous ne parlerons pas de classes de clauses mais simplement de clauses étant sous-entendu que nous considérerons une clause comme un représentant de sa classe.

I.5 METHODES DE DEDUCTIONS DERIVEES DE LA RESOLUTION

La resolution bien que beaucoup plus efficace que les méthodes trouvées jusqu'alors n'en est pas moins souvent peu efficiente.

Un certains nombre d'auteurs ont proposé alors des "raffinements" à cette méthode de façon à éviter deux écueils.

Le premier est la génération de clauses inutiles, c'est à dire de clauses qu'on peut prouver ne menant pas à la clause vide.

Le second est la génération de redondances, c'est à dire de clause déjà crées.

Le problème est de trouver des raffinements qui soient complets (c'est à dire qui n'empêchent pas la génération de réfutations) et efficaces.

La principale notion nécessaire à une bonne clarification des notions de raffinements et de stratégies est la notion de "graphe de recherche".

Le "graphe de recherche" d'un ensemble de clauses est l'ensemble des dérivations que l'on peut obtenir à partir de cet ensemble.

On peut dire qu'une méthode est un raffinement d'une autre si le "graphe de recherche" de la première est inclus dans celui de la seconde.

Sous-dérivations - méthodes de déductions

Soit $\mathcal{D}=(C_1,\ldots,C_n)$ une dérivation de la clause C_n. Toute clause C_i admet une dérivation

$$\mathcal{D}_i=(C_{i_1},\ldots,C_i) \quad \text{où} \quad C_{i_k} \in \mathcal{D} \quad \text{et} \quad i_k < i$$

Une telle dérivation est appelée sous-dérivation de $\mathcal{D}$. De plus on étendra la définition des dérivations aux suites constituées d'une seule clause.

Les deux opérations de bases étant resolution et factorisation une dérivation est:

-Soit une dérivation primitive (constituée d'une clause de départ seulement)

-Soit obtenue par factorisation à partir d'une autre dérivation

-Soit obtenue par resolution à partir de 2 autres dérivations

Une "méthode de déduction" (ou raffinement de la resolution) est définie à partir d'un prédicat φ défini pour toutes des déductions d'un ensemble de clauses quelconque S.
Une déduction $\mathcal{D}$ est dite admissible pour la méthode φ ssi $\varphi(\mathcal{D})$=vrai
Le prédicat φ est supposé en outre tel que :

si $\varphi(\mathcal{D})$=vrai alors $\varphi(\mathcal{D}')$=vrai pour toutes sous-dérivations $\mathcal{D}'$ de $\mathcal{D}$.

Cette restriction est évidemment nécessaire si l'on ne veut pas générer inutilement des clauses non admissibles, sous prétexte qu'elles peuvent mener à une dérivation qui l'est.
Une méthode de déduction permet donc d'éliminer un certain nombre de déductions-inutiles.
Une méthode sera dite complète ssi pour toute ensemble de clauses S insatisfaisable il existe une réfutation à partir de S qui est admissible .

Une dérivation admissible pour une méthode φ sera appelée une φ dérivation.

Le théorème de complétude de la resolution de Robinson est montrée par "généralisation". On sait que si S est insatisfaisable il existe une réfutation à partir d'instances terminales de clauses de S. On cherche alors à générer toute les dérivations générales susceptibles de "généraliser" la refutation terminale.

Lorsque l'on utilise une méthode de déduction qui est un raffinement de la résolution et qui est complète, on a donc intérêt à ce que toute refutation terminale soit généralisable par une refutation admissible.
D'où la définition:

DEFINITION I.5.1

Une méthode de déduction φ est généralisable ssi

$\forall \mathcal{D}'=(C'_1,\ldots,C'_n)$ φ dérivation à partir d'un ensemble S'

d'instances terminales de S,

$\exists \mathcal{D}=(C_1,\ldots,C_n)$ φ dérivation à partir de S telle que

$\mathcal{D}$ généralise $\mathcal{D}'$, c'est à dire que:

(1) $\forall i \in \{1,\ldots,n\}$ C'_i est une instance terminale de C_i

(2) $\forall i \in \{1,\ldots,n\}$ si C'_i est une instance d'une clause $C \in S$ alors C_i est un facteur de C.

(3) $\forall i \in \{1,\ldots,n\}$ si C'_i est une résolvante de C'_j et C'_k avec $j,k < i$ alors

C_i est un facteur d'une résolvante de C_j et C_k, les littéraux résolus dans C'_j et C'_k étant des instances de ceux résolus dans C_j et C_k.

Notons que la resolution sans restriction est une méthode généralisable.

Exemples de méthodes de déduction

Nous allons tout d'abord donner la définition du concept de "subsumption" dont l'utilisation peut être efficace mais difficile à manier sans faire intervenir la notion de stratégie. Voir à ce sujet la thèse de Kowalski (KOWALSKI-3)

DEFINITION I.5.2

Une clause est dite tautologie lorsqu'elle contient 2 littéraux complémentaire L et $\overline{L}$. Une telle clause est toujours satisfaite par n'importe quelle interprétation.

DEFINITION I.5.3

On dit que la clause C subsume la clause D ssi il existe une substitution σ_0 telleque

$$C\sigma_0 \subset D$$

PROPOSITION I.5.1

Soient S un ensemble de clauses, C et D deux clauses de S telles que C subsume D. Alors S est équivalent au système

$$S' = S - \{D\}$$

Autrement dit, on peut toujours dans un système supprimer une clause qui est subsumée par une autre.
En effet, soit I une interprétation qualconque, C la clause qui subsume D

$$(C\ \sigma_0 \subset D)$$

Si I satisfait S alors I satisfait S' puisque $S' \subset S$.
Réciproquement si I satisfait S' alors I satisfait C donc pour toute

instance terminale $C\sigma$ de C $\quad C\sigma \cap I \neq \Phi$
Soit alors D σ une instance terminale de D, $C\sigma_0\sigma$ est une instance terminale de C et

$$C\sigma_0\sigma \subset D\sigma \quad \text{donc} \quad (C\sigma_0\sigma) \cap I \subset D\sigma \quad \text{or} \quad C\sigma_0\sigma \neq \phi \quad \text{donc} \quad D\sigma \neq \phi$$

donc I satisfait D.
Donc si I satisfait S' il satisfait également S.

En fait ce théorème ne peut être utilisé qu'avec précaution pour une stratégie donnée car la suppression d'une clause risque de nuire à la complétude d'une méthode de déduction.

Donnons maintenant quelques exemples de méthodes de déduction.
La pluspart permettent d'éliminer les tautologies. Nous ne donnons ici que des méthodes complètes et généralisables.

DEFINITION I.5.4: Ensemble de support (WOS,CARSON,ROBINSON-2)
Soit un ensemble de clauses insatisfaisable, S_0 un sous-ensemble satisfaisable.($S-S_0$ est l'ensemble de support)
Une dérivation $\mathcal{R}$ est admissible pour l'ensemble de support ssi aucune clause dans $\mathcal{R}$ n'est une résolvante de deux facteurs de clause de S_0.

C'est à dire qu'aucune clause de $\mathcal{R}$ n'est dérivable qu'à partir de S_0 seulement. Le défaut de cette méthode est qu'il limite uniquement le nombre de clauses du 1er niveau et qu'il peut éliminer des preuves simples, en particulier les preuves utilisant des lemmes.

En effet en général S_0 consiste en un ensemble de "postulats" et $S-S_0$ en la négation du théorème à prouver à partir des postulats. Si on considère qu'un lemme est une clause dérivable à partir des postulats uniquement, il ne pourra jamais apparaître dans une dérivation admissible pour une méthode d'ensemble de support.

Cependant il semble quand même que le théorème à prouver contienne quelques informations et qu'il est naturel de l'utiliser au départ.
Il faut ajouter que l'ensemble de support peut être ajouté à un certain nombre de méthodes de déductions sans perdre la complétude.

DEFINITION I.5.5: P1-déduction (MELTZER -1)
Une dérivation est une P1-dérivation quand chaque resolvante a un parent positif (sans littéraux négatifs).

c'est une méthode plus restrictive que l'ensemble de support et qui élimine des tautologies au niveau terminal.

DEFINITION I.5.6: m-factorisation (ANDREWS-1)

Soit C une resolvante.Un m-facteur de C est un facteur de C qui unifie seulement 2 littéraux de C venant de parents distincts.

Une dérivation $\underset{\sim}{D}$ est admissible par m-factorisation ssi toute clause de $\underset{\sim}{D}$ qui est un facteur d'une resolvante est un m-facteur de cette resolvante.

Cette restriction sur la factorisation n'a bien sur aucun effet au niveau terminal, mais elle permet au niveau général d'éviter des redondances inutiles.
En outre toute dérivation terminale $\underset{\sim}{D}'$ est généralisable par une dérivation admissible $\underset{\sim}{D}$ telle que, pour toutes clauses C et C' de $\underset{\sim}{D}$ et $\underset{\sim}{D}'$ correspondantes, C et C' ont la même longueur.

DEFINITION I.5.7:Dérivations linéaires (LUCKHAM-1)

On appelle dérivation linéaire à partir d'un ensemble de clause S, toute dérivation $\underset{\sim}{D}$,

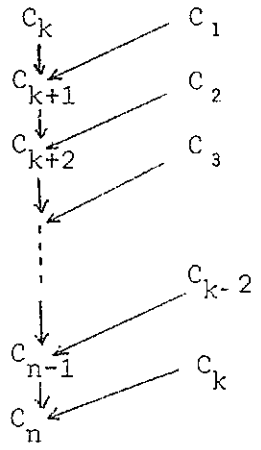
$\underset{\sim}{D}=(C_1,\ldots,C_n)$ telle que $C_1,\ldots,C_k$ facteurs de clauses de S

et C_{i+1}=facteur d'une resolvante de C_i et de C_j avec

$j< i \quad \forall i \in \{k+1,\ldots,n\}$.

C_i est le parent immédiat de C_{i+1}.
C_j est le parent lointain ($si K< j< i$) ou le parent d'entrée ($j<k$). Dans le 1er cas la resolution est dite ancestrale, dans le second d'entrée.

On peut représenter une telle dérivation par un graphe de la forme suivante:



C_{n-1} admet un parent ancêtre C_{k+2}

C_n admet un parent d'entrée C_{k-1}

Dans une telle dérivation on ne resoud une clause qu'avec un facteur d'une clause de départ ou un facteur d'un ancêtre de la clause.
Une telle méthode permet tout d'abord de générer des preuves de plus simples complexités. Elle est en outre assez naturelle car la plupart des raisonnements se font de la sorte.
Elle admet en outre un certain nombre de raffinement qui permettent de limiter encore le nombre de clauses généreés .
La méthode que nous utilisons dans le programme OEDIPE, SL-resolution; est une restriction de ces méthodes.

SL-resolution

Nous donnerons des règles de déductions utilisées, les démonstrations de cette méthode due à Koxalski se trouvant dans (KOWALSKI et KUEHNER-1). En fait dans cette méthode les clauses sont décrites non pas sous forme ensembliste mais sous forme de listes ordonnéesde littéraux.

On distingue en outre 2 sortes de littéraux: les A-littéraux et les B-littéraux, les premiers étant notés entourés d'un carré.
Exemple:

$$P(x,y)\ \boxed{Q(y)}\ \ R(z)$$

Les A-littéraux sont en fait des littéraux que l'on a déjà résolus mais que l'on conserve en les recopiant. Ceci permet d'effectuer les resolutions d'une clause avec un ancêtre sans avoir besoin de chercher cet ancêtre.

4 opérations sur une "pseudo-clause" pauvent être effectuées:

Expansion (resolution avec une clause d'entrée)
Reduction (m-factorisation et resolution avec un ancêtre)
Tronquation (suppression des A-littéraux inutiles)
Selection (choix d'un littéral qui sera résolu)

Les B-littéraux les plus à gauche sont les plus récents, ceux à droite les plus anciens.
Le principe de base est le suivant: on selecte un littéral que l'on essaie de resoudre sans intervenir sur les autres
Une chaîne d'entrée est un facteur d'une clause d'entrée.
Le B-littéral le plus à gauche est le littéral sélecté.
Voici les opérations possibles sut une clause C:

- Expansion: Le littéral L le plus à gauche (selecté) de C est un B littéral unifiable avec la complémentaire d'un littéral $\bar{L}'$ d'une clause d'entrée C', σ étant le p.g.u de L et L'
L'expansion D de C et C' est la clause obtenue en ajoutant à gauche de $C\sigma$ la clause $(C'-\{\bar{L}'\})\sigma$, et en entourant le littéral L qui devient un A littéral

- Tronquation: Le littéral le plus à gauche de C est un A littéral. La tronquation de C est la clause obtenue tous les A littéraux les plus à gauche de C et en sélectant un nouveau littéral parmi les B littéraux les plus à gauche de la clause obtenue.

- Réductions: Le littéral le plus à gauche de C est un B-littéral. Parmi les B-littéraux les plus à gauche il en existe un, L, qui est unifiable avec un littéral complémentaire d'un A-littéral de C ou avec un littéral qui est un B-littéral de C. Si σ est leur p.g.u on crée alors C' en enlevant L à $C\sigma$.

DEFINITION I.5.8

Une dérivation $\underset{\sim}{D}=(C_1,\ldots C_n)$ de pseudo-clauses est une S-L dérivation ssi

(1) C_1 est une chaine d'entrée de l'ensemble de support

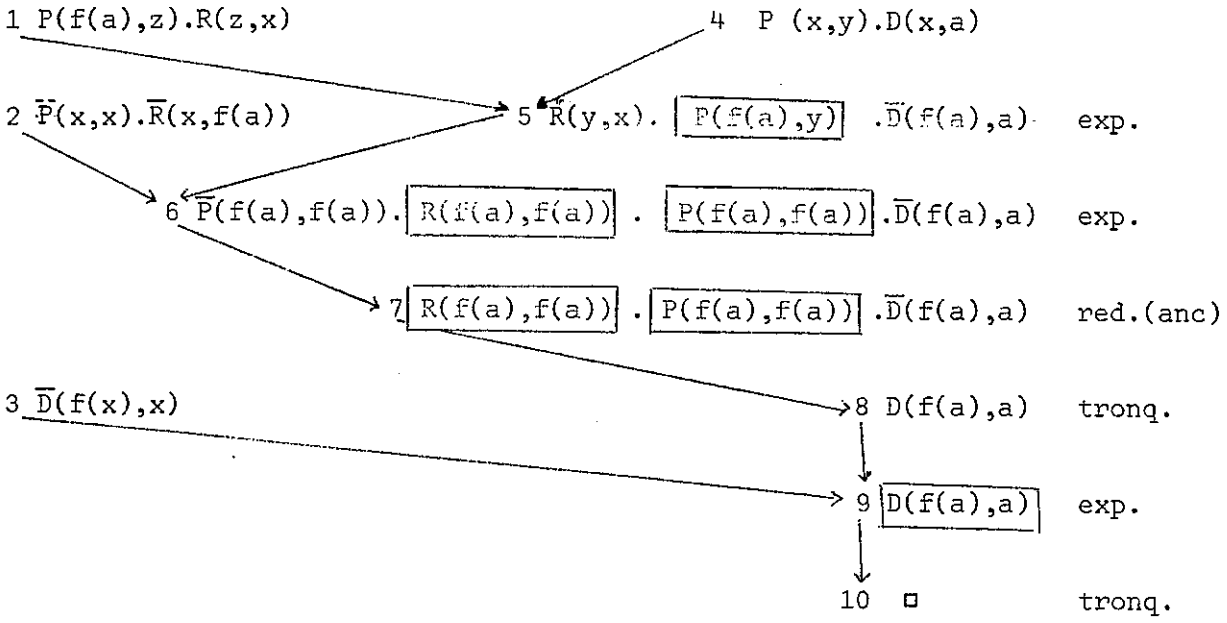
(2) C_{i+1} est obtenue de C_i par une des 3 opérations: expansion, réduction, tronquation.

(3) Deux littéraux de C_i à des places distinctes n'ont pas le même atome (sauf si C_{i+1} est une reduction de C_i)

(4) Une reduction ne peut suivre une tronquation

En outre la chaine C_1 peut être dans un ensemble de support

Voici un exemple:



Les clauses d'entrée sont les clauses 1,2,3 et 4

I.6 STRATEGIES

Etant donnée une méthode de déduction complète il s'agit, parmi l'ensemble des clauses déductibles par resolution et factorisation, d'un ensemble S de clauses insatisfaisable, de trouver la clause vide. Il faut pour cela utiliser une stratégie qui soit complète. La notion de stratégie de recherche dans les graphes a été étudiée entre autre par Hart, Millson et Raphael (HART,MILLSON,RAPHEL-1). Kowalski a repris ces notions pour les graphes de résolution. (KOWALSKI-2)

Graphes relatifs à une méthode de déduction.

Soit S un ensemble de clauses, φ une méthode de déduction. Soit G l'ensemble des φ dérivations à partir de S.

DEFINITION I.6.1

Le graphe des φ dérivations d'un ensemble de clauses S est l'ensemble des dérivations à partir de S muni de la relation $\geqslant$ définie par:

$n_1 \leqslant n_2$ ssi n_1 est une sous-dérivation de n_2

Il est évident qu'un tel graphe est en général infini.
Cependant à chaque φ dérivation $n \in G$ on peut associer un nombre unique, fini, niveau (n) tel que:

$$n \in S \Leftrightarrow \text{niveau}(n)=0$$

$n \in S \Rightarrow \text{niveau}(n)=\max\{\text{niveau}(n')/n' \text{ sous-dérivation immédiate de } n\}+1$

Une sous dérivation (immédiate) de n sera appelée ancêtre(immédiat) de n.

Si S est fini alors {n ∈ G tels que niveau(n)=K} est fini $\forall K \in \mathbb{N}$.

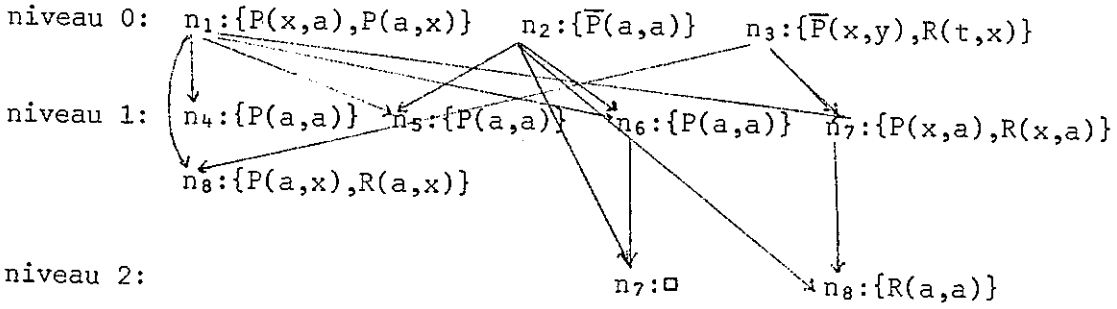
De même l'ensemble des ancêtres d'une dérivation est fini.

En fait une dérivation ne peut avoir que 1 ou 2 ancêtres immédiats:

1 ancêtre seulement : factorisation
2 ancêtres : résolution

Nous étiquetterons chacune des dérivations du graphe par la clause qu'elles dérivent.

Voici un exemple de graphe (non terminé) ou chaque noeud est étiquetté par une clause:



On est souvent amené à définir le cout d'une dérivation. Ce peut être par exemple le niveau de la clause qu'elle dérive, où le nombre de substitutions effectuées ou une "complexité" plus élaborée comme le nombre de symbole qu'elle contient,etc.... .

On recherche alors une solution (la clause vide) qui soit une réfutation de coût minimal. Voici la définition formelle donc d'un problème de démonstration automatique:

DEFINITION I.6.2

Un problème de démonstration automatique (relatif à une méthode de déduction φ) est un quadruplet

$$P=(G,\leqslant,F,g)$$

où G est la graphe des φ dérivations à partir d'un ensemble de clauses S insatisfaisable

$\leqslant$ est la relation de sous-dérivation

F est l'ensemble des réfutations à partir de S (dérivation de clauses vides)

$g:G \rightarrow \mathbb{R}^+$ est une fonction coût telle que:

$$\forall n,n' \in G \quad n \leqslant n' \Rightarrow g(n) \leqslant g(n')$$

Une solution à un tel problème est de construire un algorithme Σ qui génère à partir des noeuds de niveau 0 (ensemble qu'on appellera F_0) un noeud de F qui soit de coût minimum. On ne peut bien sur construire un noeud du graphe qu'après avoir généré tous ses ancêtres.

Stratégies de recherche

Une stratégie de recherche pour un problème P est une fonction $\Sigma : \mathbb{P}(G) \rightarrow \mathbb{P}(G)$ des parties de G dans l'ensemble des parties de G. Si pour chaque étape i on appelle Σ_i l'ensemble des noeuds déjà générés avant la (i+1)ème étape et $\tilde{\Sigma}_i$ l'ensemble des noeuds condidat à la génération de la (i+1)ème étape, l'ensemble des candidats est l'ensemble des noeuds successeurs de noeuds de Σ_i et des noeuds de $\tilde{\Sigma}_{i-1}$ sauf ceux déjà générés.
On aura donc:

$\Sigma_0 = \phi$ $\quad \tilde{\Sigma}_0 = F_0$ l'ensemble des candidats à la 1ere étape est l'ensemble des noeuds de niveau 0

$\Sigma_{i+1} = \Sigma_i \cup \Sigma(\Sigma_i)$

$\tilde{\Sigma}_{i+1} = (\{n \in G$ tels que les ancêtres immédiats de n sont dans $\Sigma_{i+1}\} \cup \tilde{\Sigma}_i) - \Sigma_{i+1}$

et $\Sigma(\Sigma_i) \subset \tilde{\Sigma}_i$

L'ensemble $\Sigma(\Sigma_i)$ est l'ensemble des noeuds générés à la (i+1)ème étape.

C'est à dire que Σ parmi les candidats possible en sélecte certain, les autres restant candidats pour les étapes futures.

DEFINITION I.6.3

Une stratégie Σ est complète pour un problème P ssi

$$G = \bigcup_{i=0}^{\infty} \Sigma_i .$$

On dit que Σ s'arrête à l'étape $i > 0$ ssi:

$$F \cap \Sigma_{i-1} = \phi \quad \text{et} \quad \begin{cases} F \cap \Sigma_i \neq \phi & \text{(solution trouvée)} \\ \text{ou} \\ \Sigma_i = \Sigma_{i-1} & \text{(pas de solution)} \end{cases}$$

DEFINITION I.6.4

Une stratégie Σ est admissible pour un problème P ssi elle est complète pour P et si elle s'arrête sur une solution n^* de coût minimale si P admet une solution ($F \neq \phi$)

$$n^* \in F \cap \Sigma_i \qquad F \cap \Sigma_{i-1} = \phi \quad \text{et} \quad n \in F \Rightarrow g(n^*) \leq g(n)$$

Mérites

DEFINITION I.6.5

Soient $\lesssim$ une relation réflexive sur G et totale (mérite)
Une stratégie de recherche Σ est compatible avec $\lesssim$ ssi:

$$\forall_{i>0} \quad \tilde{\Sigma}_i \neq \phi \Rightarrow \Sigma(\Sigma_i) \neq \phi$$

et

$$n \in \Sigma(\Sigma_i) \Rightarrow \quad n \lesssim n' \quad \forall n' \in \tilde{\Sigma}_i$$

C'est à dire que les noeuds générés sont parmi les candidats de meilleur mérite.

DEFINITION I.6.6

Un mérite $\lesssim$ est fini ssi $\forall n \in G$, $\{n' \in G / n' \lesssim n\}$ est fini

THEROEME I.6.I

Soit P un problème, $\lesssim$ un mérite fini sur G, Σ une stratégie de recherche pour P. Alors

$$\Sigma \text{compatible avec } \lesssim \Rightarrow \Sigma \text{ complet pour P}$$

Fonctions heuristiques-Stratégies diagonales

Soit $P=(G, \leq, F, g)$ un problème à couts positifs. Une fonction heuristique h est une fonction $h: G \to R^+$ qui est une estimation pour chaque $n \in G$ de $g(n^*)-g(n)$ ou n^* est une solution ayant n comme ancêtre.

Donc $f(n)=g(n)+h(n)$ est une estimation de $g(n^*)$

On prendra en fait h de telle sorte que :

$$\forall n^* \in F \quad n^* \geq n \Rightarrow g(n^*)-g(n) \geq h(n) \geq 0$$

donc

$$f(n) < g(n^*) \quad \text{et} \quad h(n^*)=0$$

Le principe de l'utilisation d'une telle heuristique est qu'il n'est pas besoin de générer un noeud n si on peut générer un noeud n' tel que

$$f(n') < f(n)$$

DEFINITION I.6.5

Le mérite $\preccurlyeq$ associé à une heuristique h dans un problème P est défini par

$$n_1 \preccurlyeq n_2 \quad \text{ssi} \quad \{f(n_1)=f(n_2) \quad \text{et} \quad h(n_1) \leq h(n_2)\} \text{ ou}$$

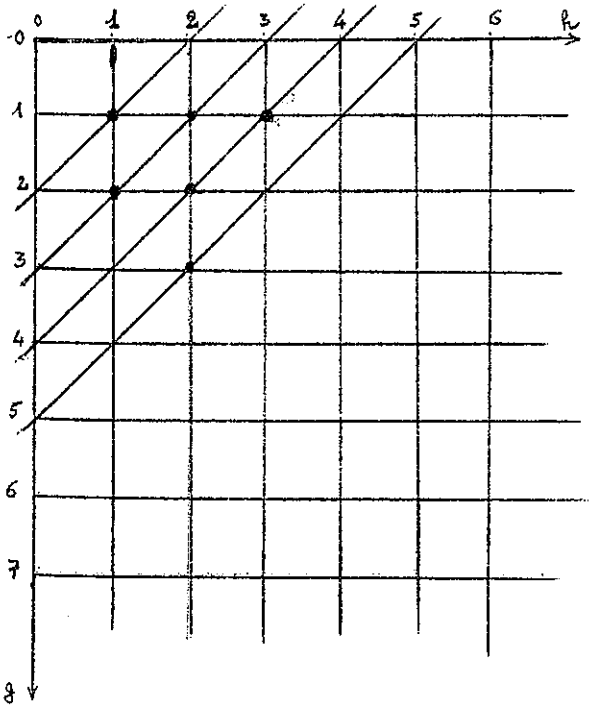
$$f(n_1) < f(n_2)$$

Une stratégie Σ pour P est une stratégie de diagonale montante pour P et pour l'heuristique h ssi

$$\Sigma \text{ est compatible avec } \preccurlyeq$$

La terminologie de diagonale provient du fait suivant:

Si on organise le graphe sous forme de cellules d'un tableau T à deux dimensions, T(i,j) étant l'ensemble des noeuds n tels que g(n)=i et h(n)=j, l'ensemble des noeuds ayant même valeur pour f sont sur une même diagonale.



Σ génère donc des noeuds de G en les plaçant dans la case correspondante du tableau

Si f est croissante pour $\leqslant$ c'est à dire si on a $n_1 \leqslant n_2 \Rightarrow f(n_1) \leqslant f(n_2)$ alors Σ va parcourir les diagonales f+g= constante, successivement de la diagonale 2 à n,n+1,.....
En effet si Σ est compatible avec $\preccurlyeq$ alors Σ ne peut générer à une étape donnée que les meilleurs candidats.
Supponsons que Σ génère $n \in G$ à l'étape (i+1). Donc n est un des meilleurs candidats c'est à dire que

$\forall\, n' \in \tilde{\Sigma}_i \quad n \preccurlyeq n'$ donc $f(n) \leqslant f(n')$. Donc tous les candidats,

c'est à dire tous les successeurs non encore générés de points déjà générés sont sur des diagonales plus grande que celle de n..
Donc dans les étapes suivantes Σ ne génèrera plus aucun élément sur une diagonale plus petite que celle de n.
Donc en fait Σ sature les diagonales les unes après les autres.
Si $\preccurlyeq$ est δ fini ces diagonales ont un nombre fini d'éléments et l'ensemble de diagonales meilleures que d diagonales données, est fini quelquesoit d.

En outre pour que Σ soit compatible avec $\preccurlyeq$ on doit respecter le principe suivant:

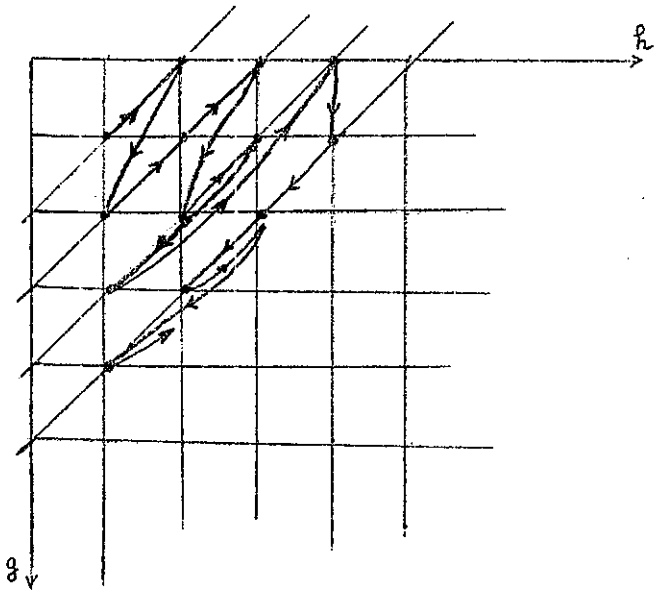
Si $f(n)=d$ est si n' est un successeur immédiat de n tel que $f(n')=f(n)=d$ on a alors

$h(n)+g(n)=h(n')+g(n')$ or $g(n) \leqslant g(n')$ donc $h(n) \leqslant h(n')$ et

$$n \preccurlyeq n'$$

donc si Σ génére n à l'étape i, il faut qu'il génère à l'étape (i+1), parmi tous les successeurs immédiats de n qui sont candidats, ceux qui sont sur la même diagonale que n, et qui ont une heuristique h minimum.

De plus lorsque les diagonales précèdent la diagonale d ont été saturés, Σ doit générer les candidats qui se trouvent sur d et qui ont l'heuristique la plus petite. D'où le terme de stratégie diagonale montante.



Parcours de Σ dans la génération de noeuds de G lorsque Σ est compatible avec $\preccurlyeq$ associé à h,

que f est croissante et que $\preccurlyeq$ estδ fini

Voici le théorème qui exprime l'optimalité des stratégies diagonales:

THEROEME I.6.2
Soit $P=(G, \leqslant, F, g)$ un problème, h une fonction heuristique, $h:G \rightarrow \mathbb{R}^+$, telle que

(1) $\forall n \in G \quad h(n) \leqslant g(n^*)-g(n) \quad \forall n^* \in F, \quad n \leqslant n^*$

(2) δfini

Si Σ est une stratégie diagonale montante pour P et h (c'est à dire compatible avec le mérite $\lesssim$ associé à h) alors:

Σ est admissible pour P

(c'est à dire que si F est non vide, Σ s'arrête sur une solution de coût minimal)

Ce théorème est une spécialisation aux graphes de résolutions, du théorème de Hart, Millson et Raphael.(HART-NILLSONET RAPHAEL-4)
Il a été introduit sous cette forme par Kowalski (KOWALSKI-2)

Ce théorème nous assure donc de l'optimalité d'une stratégie respectant les conditions annoncées. La fonction heuristique et la stratégie utilisé dans le programme OEDIPE obéissent à ces relations. Nous les expliciterons dans le chapitre III.

Ce tour d'horizon des principales notions de bases utilisées en démonstration automatique va nous permerttre de définir et de proposer une solution efficace au traitement de l'égalité formelle d'une part (ce qui nous ferons dans le chapitre II) et de donner les différents algorithmes qui ont permis l'implémentation sur machine d'une méthode de déduction et d'une stratégie de recherche des problèmes utilisant l'égalité formelle.

Chapitre II

DEFINITION et TRAITEMENT de L'EGALITE FORMELLE

Nous allons introduire dans ce chapitre la notion d'égalité formelle et expliciter son traitement automatique dans le cadre des méthodes de déductions dérivées de la résolution.

La nécessité d'introduire un tel prédicat (noté $\neq$) ainsi que les axiomes qui lui sont attachés nous sont apparus pour plusieurs raisons.

La première était d'éviter l'emploi de l'égalité classique dont le traitement, pourtant amélioré par des méthodes telles que paramodulation (ROBINSON et WOS-2), reste encore peu efficace.

L'égalité formelle n'a pas toute la puissance de l'égalité logique mais elle permet tout de même de résoudre un certain nombre de problèmes se rattachant aux domaines les plus divers.

La deuxième est que le traitement que nous proposons est extrêmement simple à implémenter sur ordinateur (on peut dire qu'il est en quelque sorte déterministe) et qu'il est compatible avec la plupart des méthodes de déduction.

La troisième raison enfin est que l'utilisation de ce prédicat permet d'améliorer l'efficacité des démonstrateurs en empêchant la génération de clauses inutiles ou redondantes (en particulier les redondances dues au fait que deux clauses peuvent avoir des instances communes).

Nous introduirons dans le programme 1 les axiomes de l'égalité formelle et la notion d'interprétation injective qui leur est attachée.

Le paragraphe 2 décrit la méthode permettant d'éliminer tous les littéraux négatifs, de la forme $\alpha \neq \beta$, d'un ensemble de clauses.

Le paragraphe 3 sera consacré au traitement d'un ensemble de clauses n'ayant plus de littéraux négatifs égalitaires.

Les théorèmes que nous démontrons dans ce paragraphe permettent de traiter automatiquement les axiomes de l'égalité formelle à partir de n'importe quelle méthode de déduction complète et généralisable. Il suffit pour celà d'ajouter une règle de contraction et une condition sur les dérivations admissibles (suppression des i tautologies). Une seule condition est imposée pour obtenir la complétude : c'est que l'ensemble de clauses considéré ait au moins un symbole fonctionnel de rang $N \geq 1$

Le dernier paragraphe traitera enfin de l'utilisation de l'égalité formelle en démonstration automatique.

1. SYSTEMES INJECTIFS

Les axiomes définissant l'égalité formelle que nous allons introduire permettent de dire que intuitivement une interprétation ne peut les satisfaire que si et seulement si deux éléments distincts de H (univers de Herbrand) sont interprétés par deux individus distincts. Autrement dit si α et β sont deux éléments de H, alors le littéral $\alpha \neq \beta$ n'est satisfait que ssi α et β sont deux "mots" égaux.

Interprétations injectives

Dans ce chapitre nous noterons les littéraux égalitaires (ayant $\neq$ comme symbole relationnel) non pas sous forme fonctionnelle mais sous la forme

$$\alpha \neq \beta \quad \text{ou} \quad \overline{\alpha \neq \beta}$$

DEFINITION II.1.1

Soit S un ensemble de clauses, $\neq$ un des symboles relationnels de S. Une interprétation de Herbrand I est dite injective ssi:

$$\forall t \in H\ ,\ \forall t' \in H \qquad (t \neq t') \in I \Leftrightarrow t = t'$$

Une telle interprétation sera également appelée une i-interprétation.

PROPOSITION II.1.1

Soit I une interprétation. I est une i-interprétation ssi pour tous termes de l'univers de Herbrand, t et t', on a:

(1) $t = t' \Rightarrow (t \neq t') \in I$

(2) $t \neq t' \Rightarrow \overline{(t \neq t')} \in I$

En effet I étant une interprétation $\overline{(t \neq t')} \in I \Leftrightarrow (t \neq t') \notin I$

DEFINITION II.1.2

Un ensemble de clauses S est i-satisfaisable ssi il existe une i-interprétation qui le satisfait et i-insatisfaisable dans le cas contraire.

Systèmes injectifs

Nous allons définir maintenant axiomatiquement l'égalité formelle.

DEFINITION I.1.3

Un ensemble de clauses S est dit injectif ssi il contient l'ensemble des clauses suivant, où $x, x_1, \ldots, x_n, y, \ldots, y_n$ désignent des variables.

(1) $\{x \neq x\}$

(2) $\{\overline{f(x_1,\ldots,x_n) = g(y_1,\ldots,y_m)}\}$ pour tous symboles fonctionnels distincts f et g de S, d'ordre n et $m \geqslant 0$.

(3) $\{\overline{f(x_1,\ldots,x_k,\ldots,x_n) = f(y_1,\ldots,y_k,\ldots y_n)}, (x_k \neq y_k)\}$

Pour tout symbole fonctionnel f de S d'ordre $n \geqslant 1$ et pour tout $K \in \{1,\ldots,n\}$.

L'ensemble de ces clauses constitue les axiomes de l'égalité formelle et sera noté AX(S).

Notons que parmi les axiomes (2) figurent ceux où f ou g sont des constantes (ou m=0).
Ces axiomes expriment en fait à quelle condition deux termes t' et t éventuellement non terminaux restent égaux (resp.différents) par toute instantiation par une même substitution σ . Voici un exemple:

$S=\{ C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$

$C_1=\{\overline{R}(x,x), R(a,a), (x \neq a)\}$ $\quad C_2=\{P(a,f(x,y)), \overline{Q}(x), \overline{R}(x,y), \overline{(x \neq y)}\}$

$C_3=\{\overline{(a \neq f(x,y))}\}$ $\quad C_4=\{\overline{(f(x,y) \neq a)}\}$

$C_5=\{\overline{(f(x,y) \neq f(x',y'))}, (x \neq x')\}$

$C_6=\{\overline{(f(x,y) \neq f(x',y'))}, (y \neq y')\}$

$C_7=\{(x \neq x)\}$

Ici $\{C_3, C_4\}$ constitue l'ensemble des axiomes de type (2).

$\{C_5, C_6\}$ constitue celui des axiomes de type (3).

C_7 est l'axiome (1).

On peut remarquer que l'ensemble de ces axiomes est défini de manière unique par la donnée de l'ensemble F des symboles fonctionnels de S.

THEOREME II.1.1

Soit S un ensemble de clauses injectif, AX(S) l'ensemble des axiomes de l'égalité formelle.
Soit I une interprétation de Herbrand de S. Alors:

I injective $\Leftrightarrow$ I satisfait AX(S)

DEMONSTRATION DE $\Leftarrow$:

Soit I une interprétation qui satisfait AX(S).Montrons que I est injective (Prop.II.1.1):

Soit $t \in H$ quelconque. I satisfait toute instance terminale de l'axiome (1) donc $(t \neq t) \in I$.

Donc $t=t' \Rightarrow (t \neq t') \in I$

Montrons par récurrence sur la longueur de t et t' que:

$\forall\, t,t' \in H \quad t \neq t' \Rightarrow \overline{(t \neq t')} \in I$.

Soient t et t' distincts.
Si t (resp.t') est de longueur 1 alors t(resp.t') est une constante. Donc puisque t et t' sont distincts

$\overline{(t \neq t')}$ est une instance d'un axiome de type (2)

Donc $\overline{(t \neq t')} \in I$.

Si t et t' sont tous les deux de longueur $\geqslant 2$ alors:

$t=f(\mu_1, \ldots\ldots, \mu_n)$ et $t'=f'(\mu'_1, \ldots\ldots, \mu'_m)$.

or $t \neq t' \Rightarrow f \neq f'$ ou $(f=f'$ et $\exists k \in \{1,\ldots,n\}$ tel que $\mu_k \neq \mu'_k)$

Si $f \neq f'$ alors $(\overline{t \neq t'})$ est une instance de (2) donc $\overline{(t=t')} \in I$

Si $f=f'$ et $\mu_k \neq \mu'_k$ alors d'aprés l'hypothèse de récurrence $(\overline{\mu_k \neq \mu'_k}) \in I$ puisque la longueur de μ_k(respdeμ'_k) est plus petite que celle de t (resp.t').

Donc puisque $\{(\overline{t \neq t'}),(\mu_k \neq \mu'_k)\}$ est une instance de (3) et que $(\mu_k \neq \mu'_k) \notin I$, on a: $\overline{(t \neq t')} \in I$.

donc $\forall t \in H, \forall t' \in H \quad t \neq t' \Rightarrow \overline{(t \neq t')} \in I$

DEMONSTRATION DE $\Rightarrow$:
Montrons que toute i-interprétation de S,I, satisfait AX(S).
Soit $(t \neq t')$ une instance terminale de l'axiome (1).
D'aprés II.1.1 $(t \neq t') \in I$ donc cet axiome est bien satisfait par I.

Soit $\{(\overline{t \neq t'})\}$ une instance terminale d'un axiome du type(2)

avec $t=f(t_1,....t_n)$ et $t'=g(t'_1,...., t'_n)$.
$\Rightarrow$
$f \neq g$ on a alors $t \neq t'$ donc d'aprés II.1.1

$(\overline{t \neq t'}) \in I$. donc tous les axiomes (2) sont satisfaits par I.

Soit $C=\{(\overline{f(t_1,.....,t_n) \neq f(t'_1,....,t'_n)}),(t_i \neq t'_i)\}$ une instance terminale d'un axiome (3).

Soient $t=f(t_1,....,t_n)$ et $t'=f(t'_1,....t'_n)$, $t \in H$ et $t' \in H$

-Si $t_i \neq t'_i$ alors $t \neq t'$ donc $\overline{t \neq t'} \in I$ donc $C \cap I \neq \phi$

-Si $t_i=t'_i$ alors $(t_i \neq t'_i) \in I$ donc $C \cap I \neq \phi$

Donc toute instance terminale d'un axiome (3) est satisfaite par I.

PROPOSITION I.I.2
Soient t et t' deux termes (non nécessairement terminaux) non unifiables. Alors la clause unitaire $C=\{\overline{t \neq t'}\}$est logiquement déductible des axiomes de l'égalité formelle.

En effet soit I une interprétation qui satisfait AX(S), c'est donc une i-interprétation quelconque.
Soit $C\sigma$ une instance terminale quelconque de C, $C\sigma = \overline{\{t\sigma \neq t'\sigma\}}$.
Comme t et t' ne sont pas unifiables

$t\sigma \neq t'\sigma$ donc $\overline{\{t\sigma \neq t'\sigma\}} \in I$ donc $C\sigma \cap I \neq \phi$ donc
I satisfait C.

Nous avons vu que les axiomes de l'égalité formelle sont définis par la seule donnée de l'ensemble F des symboles fonctionnels de S. Cette remarque et le théorème II.1.1 conduisent à la définition des "systèmes injectifs".

DEFINITIONS I.1.4

On appelle <u>système injectif</u> tout couple (S_0,F) où S_0 est un ensemble de clauses et F est un ensemble de symboles contenant l'ensemble des symboles fonctionnels de S,F_0 .

A tout couple de cette sorte on peut associer de manière unique en ensemble injectif de clauses S tel que:

(1) $S = S_0 \cup AX(S)$

(2) F est l'ensemble des symboles fonctionnels de S

Réciproquement, S étant un ensemble injectif de clauses,il existe un couple unique (S_0,F) qui vérifie (1) et (2).

Notons qu'à priori l'ensemble F_0 des symboles fonctionnels peut être distinct de F. En fait F défini également l'univers de Herbrand dans lequel on considère les interprétations .

Lorsque nous parlerons d'interprétations de Herbrand d'un système (S_0,F) injectif l'univers considéré sera H défini par F (qui éventuellement pourra contenir d'autres symboles que ceux de S_0).

Exemples: Soit (S_0,F) le système suivant injectif:

$S_0 = \{C_1,C_2,C_3\}$, $F=\{a,b,f\}$ avec

$C_1=\{\overline{P}(x,y),P(a,a),x\neq y\}$ $C_2=\{Q(b)\}$ $C_3=\{\overline{P}(a,b),Q(a)\}$

l'ensemble de clauses S injectif qui lui est associé est le suivant:

$S=S_0 \cup AX(S)$ avec AX(S)={ axiomes de l'égalité formelle défini par F}

DEFINITION I.1.5
Un système injectif (S_0,F) est dit i-insatisfaisable ssi l'ensemble S est i-insatisfaisable (pour toute les interprétations dans l'Univers de Herbrand défini par F).

THEROEME II.1.2
Soient (S_0,F) un système injectif et S l'ensemble de clauses injectif associé.
Alors:

$$S \text{ insatisfaisable} \Leftrightarrow (S_0,F) \text{ i-insatisfaisable}$$

$S=S_0 \cup AX(S)$ où $AX(S)$ est l'ensemble des axiomes définis par F.
Soit H l'Univers de Herbrand de S.

DEMONSTRATION DE $\Rightarrow$:
Soit S insatisfaisable. Soit I une i-interprétation de S_0 dans H. I ne peut satisfaire S_0 puisque $S_0 \subset S$. Donc S_0 est i-insatisfaisable.

DEMONSTRATION DE $\Leftarrow$:
Soit I une interprétation de S dans H. Si I satisfait AX(S) alors I est une interprétation injective donc ne peut satisfaire S_0 donc I ne peut satisfaire S.
Si I ne satisfait pas AX(S) alors I ne peut satisfaire non plus S.
Dans les 2 cas S n'est jamais satisfaite par I si (S_0,F) est i-insatisfaisable.

Donc tester si un ensemble de clauses $S_0 \cup AX$, où AX est l'ensemble des axiomes de l'égalité formelle (pour tous les symboles fonctionnels au moins de S_0) est insatisfaisable revient à tester que l'ensemble S_0 est i-insatisfaisable pour toutes les interprétations de Herbrand dans l'Univers H défini par F ensemble des symboles fonctionnels de $S_0 \cup AX$.

La donnée de F est importante car par exemple l'ensemble S_0 suivant:

$$S_0=\{\{\overline{P}(x),P(a)\} , \{x\neq y,y\neq z,x\neq z\}\}$$

est i-insatisfaisable si l'on prend pour ensemble F, $F=\{a,b,c\}$ mais est i-satisfaisable si l'on prend $F=\{a\}$.

En effet dans le premier cas la clause $C=\{x\neq y, y\neq z, x\neq z\}$ admet comme instance terminale dans l'univers de Herbrand $H_1=\{a,b,c\}$ la clause C_1:

$C_1=\{a\neq b, b\neq c, a\neq c\}$ qui est i-insatisfaisable.

Par contre dans le second cas la seule instance terminale de C dans l'Univers de $H_2=\{a\}$ est

$C_2=\{a\neq a,\ a\neq a,\ a\neq a\}$ qui est i-satisfaisable.

Le paragraphe que nous allons maintenant étudier est consacré à l'élimination des littéraux égalitaires négatifs d'un système de clauses injectif. Lorsque nous parlerons d'une i-interprétation de clause cette interprétation aura toujours comme Univers de Herbrand un ensemble de termes définis à partir d'un ensemble de symboles fonctionnels contenant ceux de la clause ou de l'ensemble de clauses. (ceci en a accord avec les définitions II.1.4et II.1.5)

2.TRANSFORMATION D'UN SYSTEME INJECTIF EN UN SYSTEME i-POSITIF

i-littéraux

DEFINITION II.2.1

Soit C une clause et L un littéral de C.
L est un i-littéral de C ssi L est de la forme $t\neq t'$ (resp.$\overline{t\neq t'}$).

C est dite i-positive ssi elle ne contient pas de i-littéraux négatifs et i-vide si elle ne contient aucun i-littéral.

Un ensemble de clauses S est i-positif ssi toutes ses clauses sont i-positives.

Un système injectif (S_0,F) est i-positif ssi S_0 est i-positif.

La partie égalitaire d'une clause est l'ensemble de ses i-littéraux.

i-tautologies

DEFINITIONS II.2.2

Une clause contenant un littéral du type $\overline{t\neq t'}$ où t et t' ne sont pas unifiables est une i-tautologie négative.
Une clause contenant un littéral du type $t\neq t$ est une i-tautologie positive.

PROPOSITION II.2.1

Toute clause C qui est une i-tautologie positive (resp.négative) est satisfaite par une i-interprétation quelconque.

En effet si C est une i-tautologie positive elle est subsumée par l'axiome $\{x \neq x\}$.
Si C est une i-tautologie négative alors $C=D \cup \{\overline{t \neq t'}\}$ où t et t' ne sont pas unifiables.
Donc C est subsumée par une clause du type $\{\overline{t \neq t'}\}$ avec t et t' non unifiables. Or d'aprés II.1.2 une telle clause est logiquement déductible des axiomes AX(S) sonc C également.

Contraction positive d'une clause

DEFINITION II.2.3
Soit C une clause. On définit récursivement une contraction positive de C de la manière suivante:
(i) C est une contraction positive d'elle-même.
(ii) Si C' est une cont.pos. de C et si C'' est une cont.pos. de C' alors C'' est une contraction positive de C
(iii) Si $(\overline{t_1 \neq t_2}) \in C$ avec t_1 et t_2 unifiables, σ un de leur p.g.u alors:

$C'=(C-\{\overline{t_1 \neq t_2}\})\sigma$ est une contraction positive de C.

Voici un exemple: $C=\{\overline{x \neq f(y)}, \overline{y \neq a}, P(x,y), R(x)\}$

$C_1=\{\overline{y \neq a}, P(f(y),y), R(f(y))\}$ $C_2=\{\overline{x \neq f(a)}, P(x,a), R(x)\}$

$C_3=\{P(f(a),a), R(f(a))\}$ sont des contractions positives de C.

PROPOSITION II.2.2
Soit C' une contraction positive de C. Alors pour toute interprétation injective I

I satisfait C ssi I satisfait C'

En effet soit $C=D \cup \{t \neq t'\}$ et $C'=D\sigma$ σ étant un p.g.u de t et t'

DEMONSTRATION DE $\Rightarrow$:
C' est une résolvante de C et de l'axiome $\{x \neq x\}$ donc toute i-interprétation I, puisqu'elle satisfait cet axiome, satisfait également C' si elle satisfait C (puisque la résolution est une règle de déduction consistente).

DEMONSTRATION $\Leftarrow$:
Soit I une i-interprétation qui satisfait C' et soit $C\sigma_1$ une instance terminale quelconque de C

-ou bien $\exists K$ telle que $\sigma_1=\sigma.K$ auquel cas

$$C\sigma_1=C\sigma.K = D\sigma.K \cup \{\overline{(t\neq t')}\sigma.K\}$$

or $D\sigma.K$ est une instance terminale de C' donc $D\sigma.K \cap i \neq \phi$ et donc

$C\sigma_1 \cap I \neq \phi$

-ou bien $\forall K\ \sigma_1 \neq \sigma.K$ auquel cas, d'aprés le théorème I.3.1,

$t\sigma_1 \neq t'\sigma_1$ donc $\overline{(t\neq t')}\sigma_1 \in I$

donc $C\sigma_1 \cap I \neq \phi$

dans les deux cas $C\sigma_1$ est satisfaite par I, ceci $\forall C\sigma_1$ instance terminale de C donc C est satisfaite par I

PROPOSITION II.2.3
Toute clause C admet une contraction positive C' qui est soit une i-tautologie négative, soit une clause i-positive, et qui peut être obtenue à partir de C par un nombre fini de contractions du type (iii)

Montrons le peu de récurrence sur le nombre de i-littéraux négatifs $l(C)$ de C.
-si $l(C)=0$ alors C'=C et C' est une clause i-positive

-supposons la propriété vraie pour les clauses telles que $l(C) < n$ $(n > 0)$

-Soit C une clause avec $l(C)=n > 1$.
ou bien C est une i-tautologie négative auquel cas C'=C
ou bien $\exists t$ et t' tels que $\overline{t\neq t'} \in C$ avec t et t' unifiables, σ un de leur p.g.u.
Soit $C''=(C-\{\overline{t\neq t'}\})$. C"est une contraction positive de C et $l(C'') < n$
donc C" admet une contraction positive C' qui est une i-tautologie négative ou une clause i-positive.
C' étant une contraction de C le théorème est donc démontré.

THEOREME II.2.1

Soit (S_0,F) un système injectif. Alors il existe un système injectif (S'_0,F) qui est tel que :

(S_0,F) i-insatisfaisable ssi (S'_0,F) i-insatisfaisable

-Soit $c \in S_0$ - $\exists$ $\varphi(c)$ clause i-positive ou i-tautologie qui est une contraction de C.
-Soit S'_0 = {$\varphi(c)$ telles que $\varphi(c)$ i-positive}
montrons que S_0 i-satisfaisable $\Leftrightarrow$ S'_0 i-satisfaisable

$\Rightarrow$ Soit I une i-interprétation qui satisfait S_0. Soit $\varphi(c) \in S'_0$ d'aprés la proposition II.2.2 comme c est satisfait par I, $\varphi(c)$ l'est également. Donc S'_0 est satisfait par I

$\Leftarrow$ Soit I une i-interprétation qui satiqfait S'_0. Soit $c \in S_0$ quelconque. Ou bien $\varphi(c)$ est une tautologie négative auquel cas $\varphi(c)$ et donc est satisfaite par I (porposition II.2.1).
Ou bien $\varphi(c)$ n'est pas une i-tautologie négative. Donc $\varphi(c) \in S'_0$ et $\varphi(c)$ est donc satisfaite par I.
$\varphi(c)$ étant une contraction de c,c est aussi satisfaite par I.
Donc S_0 est satisfaite par I.

L'algorithme pour passer d'un système injectif à un système i- positif est très simple:
pour chaque clause C on construit une contraction C' i-positive ou i-tautologie par la méthode III.2.3. On remplace C par la clause alors obtenue en éliminant cette dernière si c'est une i-tautologie.
En fait sans le programme OEDIPE où se trouve implémenté le traitement de l'égalité formelle, le soin est laissé à l'utilisateur de faire cette transformation, le principal intérêt de = résidant dans son utilisation dans les littéraux positifs.
C'est la seconde phase qui va être maintenant étudiée qui est de loin la plus importante dans le traitement de l'égalité formelle.

3 i-METHODES DE RESOLUTION DE SYSTEMES i-POSITIFS INJECTIFS

Une fois effectuée la transformation d'un système injectif en un système i-positif (c'est à dire sans littéraux de la forme $\alpha \neq \beta$) par la méthode décrite précédemment on peut alors utiliser n'importe quelle méthode de déduction (§I.5) pour traiter ce système.
Le traitement automatique des axiomes de l'égalité formelle (qu'on n'a pas, bien sûr, besoin d'introduire dans l'ensemble des clauses de départ) est fait grâce à deux opérations sur les clauses que l'on ajoute à la méthode utilisée:
1) une opération de "contraction" qui consiste à éliminer d'une clause tous les littéraux de la forme $t \neq t'$, où t et t' sont deux termes non unifiables.
2) une opération qui consiste à tester la présence d'un littéral de la forme $t \neq t$ dans la clause (et de considérer alors cette clause comme non admissible).

A part ces deux opérations, la partie égalitaire des clauses n'interviendra pas quelque soit la méthode utilisée.
On peut donc voir que ces deux opérations sont vraiment simples à ajouter à la méthode qu'on utilise et que n'importe quel programme de démonstration automatique pourra traiter l'égalité formelle de façon automatique moyennant un minimum de modifications.

Contractions

La méthode de déduction pour tester la i-insatisfabilité d'un système i-positif peut être déduite d'une méthode de déduction quelconque complète et généralisable à laquelle on ajoute une opération de contraction

DEFINITION II.3.1
Soit C une clause. On appelle contraction de C la clause
$C' = C - \{L \in C / L = (t \neq t') \text{ avec } t \text{ et } t' \text{ non unifiables}\}$

Exemple:
$C = \{P(x), Q(y), x \neq f(x)\}$ $\qquad$ $C' = \{P(x), Q(y)\}$

PROPOSITION II.3.1
Soit C une clause et C' sa contraction. Alors $\forall$I i-interprétation

$$I \text{ satisfait } C \Leftrightarrow I \text{ satisfait } C'$$

$C' \subset C$ donc C' subsume C d'où la démonstration $\Leftarrow$.

D'autre part C' est obtenue à partir de C par une suite de resolutions sur des clauses du type $\{\overline{t=t'}\}$ où t et t' sont non unifiables. Or, d'aprés II.2.1 ces clauses sont des i-tautologies donc toute i-interprétation qui satisfait C satisfait C'. D'où la démonstration de $\Rightarrow$.

i-dérivations à partir d'un ensemble de clauses i-positif

φ étant une méthode de déduction, une i-φ-dérivation est une φ-dérivation telle que les littéraux égalitaires ne sont ni résolus, ni factorisés mais seulement "contractés"! Nous allons donner une définition précise de cette nôtion et un exemple pour l'illustrer.

DEFINITION II.3.2

Soit φ une méthode de déduction, S un ensemble de clauses et S° l'ensemble des clauses parties non égalitaires des clauses de S.

On dit que la suite $\underset{\sim}{D}=(C_1,\ldots,C_n)$ est i-φ-dérivation à partir de S ssi il existe une φ-déduction $\underset{\sim}{D}^\circ=(C_1^\circ,\ldots.C_n^\circ)$ à partir de S_0 telle que:

(i) $\forall i \in \{1,\ldots.n\}$ C_i n'est pas une i-tautologie positive

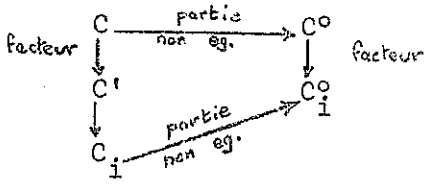
(ii) $\forall i \in \{1,\ldots.n\}$ C_i° est la partie non égalitaire de C_i

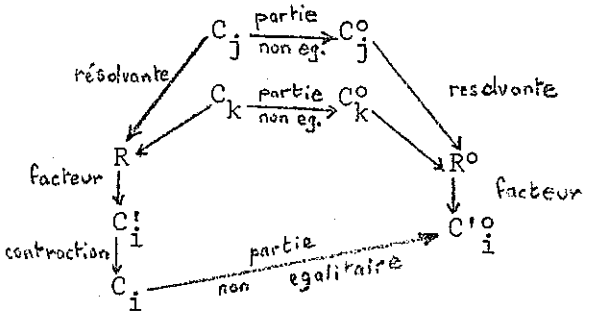
(iii) $\forall i \in \{1,\ldots.n\}$

Si C_i° est un facteur de $C^\circ \in S^\circ$ (C° partie égalitaire de $C \in S$) alors C_i est la contraction d'un facteur de C (les littéraux factorisés étant les mêmes dans C et C°)

Si C_i° est un facteur d'une résolvante de C_j° et C_k° alors C_i est la contraction d'un facteur d'une résolvante de C_j et C_k (les littéraux factorisés et résolus étant les mêmes dans C_j° et C_j, C_k° et C_k)

On peut illustrer cette définition par les 2 schémas suivants





Donnons maintenant un exemple d'une i-dérivation

Soient $C=\{\overline{Q}(x,y),Q(y,x),Q(y,z),x\neq y,\ x\neq f(z)\}$

$D=\{Q(a,b)\}$ $\quad F=\{Q(a,a)\}$

$E=\{\overline{Q}(b,a)\}$

Considérons la i-déduction suivante:

Factorisation de C: $C'_1=\{\overline{Q}(z,y),Q(y,z),\ z\neq y,\ z\neq f(z)\}$

contraction de C'_1: $C_1=\{\overline{Q}(z,y),Q(y,z),z\neq y\}$

resolution de C_1 et D: $C'_2=\{Q(b,a),a\neq b\}$

contraction de C'_2: $C_2=\{Q(b,a)\}$

resolution de C_2 et E: $C_3=\square$

On peut remarquer qu'en fait on résoud et on factorise les clauses normalement sauf sur les i-littéraux. Ceux-ci servent en quelque sorte "d'interdiction" pour certaines unifications.

Ainsi dans cet exemple les clauses F et C donnent comme resolvante la clause quivante:

$\{Q(a,a),Q(a,z),a\neq a,a\neq f(z)\}$ qui est une i-tautologie donc qui ne peut

intervenir dans aucune i-déduction.
On peut donc considérer que les i-littéraux positifs sont des restrictions aux quantificateurs universels.
Par exemple la clause C peut s'interpréter par:

$\forall x\ \forall y\ \forall z$ sauf peut être pour $x=y$ et $x=f(z)$ on a: $\overline{Q}(x,y)\vee Q(y,x)\vee Q(y,z)$

On peut alors utiliser toute instance de$\{\overline{Q}(x,y),Q(y,x),Q(y,z)\}$ à condition que les conditions soient vérifiées.
Dans l'exemple la déduction $\underset{\sim}{D}^{o}$ associée à $\underset{\sim}{D}=(C_1,C_2,C_3,)$ est la suivante:

$\underset{\sim}{D}^{o}=(C^{o}_1,C^{o}_2,C^{o}_3,)$ avec

$C^{o}_1=\{Q(z,y),Q(y,z)\}$ $\quad C^{o}_2=\{Q(b,a)\}$ $\quad C^{o}_3=\square$

THEOREME II.3.1

Soit φ une méthode de déduction complète et généralisable. Alors si (S,F) est un système i-positif, on a la propriété suivante:

Si (S,F) est i-insatisfaisable il existe une i-φ-déduction à partir de S d'une clause C dont la partie non égalitaire est vide. C'est à dire que:

$$C = \bigcup_{i+1}^{n} \{t_i \neq t'_i\} \quad \text{avec } t_i \neq t'_i \quad \forall\, i \in \{1,\ldots n\}$$

ou $C = \square$

Nous aurons besoin pour la démonstration de ce théorème de 2 lemmes.

LEMME 1

Soit (S,F) un système i-positif, injectif, i-insatisfaisable. Alors il existe un ensemble fini S' d'instance terminales de clauses de S, qui ne soient pas i-tautologies et tel que (S',F) est i-insatisfaisable.

(S,F) étant i-insatisfaisable l'ensemble S $\cup$ AX où AX est l'ensemble des axiomes de l'égalité formelle défini par F est insatisfaisable (théorème II.1.2).
Donc d'aprés le théorème de Herbrand il existe un ensemble T fini, d'instances terminales de S $\cup$ AX(F), insatisfaisable.
Soit $T = S_1 \cup T_1$ cet ensemble avec S_1 ensemble d'instances de S et T_1 ensemble d'instances de AX(F).
Soit $S' = \{C_1 \in S_1 / C_1$ ne soit pas i-tautologie$\}$. Montrons que (S',F) est i-insatisfaisable.

Soit I une i-interprétation quelconque. T étant insatisfaisable $\exists\, C \in T$ telle que C n'est pas satisfaite par I, i-e $C \cap I = \phi$ (puisque C est terminale).
Or I étant une i-interprétation dans l'univers de Herbrand H défini par F, tous les axiomes de AX sont satisfait par I donc T_1 est satisfait par I. Donc $C \in S_1$
D'autre part C ne peut être une i-tautologie donc $C \in S'$. Donc S' n'est pas satisfait par I.
(S',F) est donc i-insatisfaisable.

LEMME 2

Soient 3 clauses C,C" et D' telles que

C=D $\dot{\cup}$ E (E partie égalitaire de C), C"=D" $\dot{\cup}$ E"

(E" partie égalitaire de C") avec

C" instance terminale de C

D" instance terminale de D'

D' instance de D

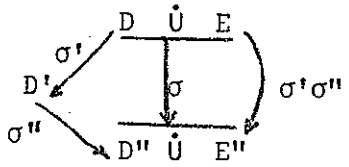
Alors il existe deux substitutions σ' etσ " telles que

$C"=C\sigma'.\sigma"$

$D"=D'\sigma''$

$D'=D\sigma'$

On peut illustrer ce lemme par le schéma suivant



Soient σet σ_D les substutions minimales telles que

C"=Cσ i-e telle que D"=Dσ et E"=Eσ et

$D"=D\sigma_D$

D'aprés la proposition I.3.3 et puisque D"=Dσ on a $\sigma_D \subset \sigma$

donc $\exists\sigma_E$ telle que $\sigma=\sigma_D \dot{\cup} \sigma_E$ avec $V(\sigma_E) \cap D= \phi$

Soient ρ' et $\rho"$ les substitutions minimales telles que D'=Dρ' et D"=D'$\rho"$.

Toujours d'aprés I.3.3 $V(\rho') \subset V(D)$

Soient $\sigma'= \rho' \dot{\cup} \sigma_E$ et $\sigma"= \rho"$

D'aprés I.3.4, puisque $V(D) \cap V(\sigma_E)= \phi$; Dσ'=Dρ'=D' et

D'$\sigma"$=D'$\rho"$=D"

Donc D"=D$\sigma'\sigma"$ donc d'aprés I.3.3 $\sigma_D \subset \sigma'\sigma"$

D'aprés I.3.5 comme σ_E est une substitution terminale:

$(\rho' \dot{\cup} \sigma_E).\sigma" \supseteq \sigma_E$ donc $\sigma'.\sigma" \supset \sigma_E$ donc $\sigma \subset \sigma'.\sigma"$

D'aprés I.36 puisque $C\sigma$ est une instance terminale:

$$C\sigma'.\sigma''=C\sigma$$

D'où le lemme.

Démonstration du théorème II.3.1

Le système (S,F) étant i-insatisfaisable, d'après le lemme 1, il existe S' ensemble fini d'instances de clauses de S tels que ces instances ne soient pas des i-tautologies et telles que (S',F) est i-insatisfaisable.
Soit S° l'ensemble des parties non égalitaires de clauses de S, et S'° celui des clauses de S'. Alors S'° est un ensemble d'instances terminales de S°. Plus précisemment on a:

(1) $\forall C'^\circ \in S'^\circ \quad \exists C' \in S' , \exists C \in S$ tels que $C'=C'^\circ \dot{\cup} E'^\circ$

où E'° est la partie égalitaire de C' et ne contient pas de i-littéraux

de la forme $t \neq t$, et $C=C^\circ \cup E^\circ$ avec C' instance terminale de C

Donc C'° est la contraction de C' (puisque au départ (S,F) est un système i-positif et que C' est une clause terminale) et d'aprés la proposition II.3.1 toute i-interprétation qui satisfait C'° et réciproquement

(S',F) est un système i-insatisfaisable donc (S'°,F) aussi puisque pour chaque clause de S' sa contraction appartient à S'°. Or S'° ne contient pas de i-littéraux donc c' est un ensemble insatisfaisable.

D'où le premier résultat: S'° est un ensemble d'instances terminales de S° fini insatisfaisable.

Donc puisque φ est complète, il existe une φ-déduction terminale de $\square$ à partir de S'°, soit: $\underset{\sim}{D}'^\circ=(C'^\circ_1,.....C'^\circ_n)$ avec $C'^\circ_n= \square$

φ étant généralisable et S'° étant un ensemble d'instances de S°, $\exists$ une φ-déduction de $\square$ à partir de S° qui généralise $\underset{\sim}{D}'^\circ$

$\underset{\sim}{D}^\circ=(C^\circ_1,....C^\circ_n)$ avec $C^\circ_i= \square$ et C'°_i instance terminale de C°_i

$\forall i \in \{1,...n\}$

Nous allons construire par récurrence une i-φ-dérivation $\underset{\sim}{D}=(C_1,...C_n)$ à partir de S qui admet $\underset{\sim}{D}^\circ$ comme φdérivation associée et qui soit telle que

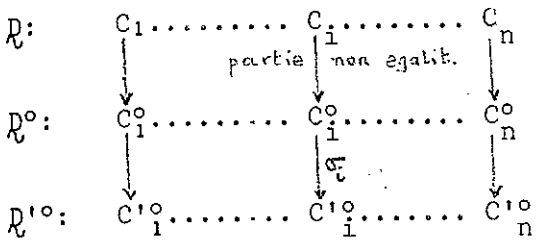
(2) $\forall i \in \{1,....n\}$ $\exists \sigma_i$ avec:

(i) $C_i = C^\circ_i \dot{\cup} E^\circ_i$ $\quad$ E°_i partie égalitaire de C_i

(ii) $C'^\circ_i = C^\circ_i \cdot \sigma_i$

(iii) $E^\circ_i.\sigma_i$ n'a pas de littéraux de la forme $(t \neq t)$, et est terminale

On aura donc le schéma suivant:



1°) Considérons le cas où C°_i et C'°_i sont des instances de $C^\circ \in S^\circ$. D'aprés (1)

$\exists C \in S$ et $C' \in S'$ tel que $C = C^\circ \dot{\cup} E^\circ$ $\quad C' = C'^\circ_i \cup E'^\circ_i$ où C' est une

instance terminale de C, E'°_i sans littéraux $t \neq t$.

Or D° généralise D donc C°_i est un facteur de C° qui admet comme instance C'°_i terminale.
Donc d'aprés le lemme 2 $\exists \sigma'$ et σ'' tels que

(3) $C' = C\sigma'\sigma''$ $\quad C^\circ_i = C^\circ_j.\sigma'$ et $C'^\circ_i = C^\circ_i.\sigma''$ $\quad E^\circ\sigma'\sigma'' = E'^\circ_i$

Soit $\overline{C_i} = C\sigma'$.

$\overline{C_i} = C^\circ.\sigma' \dot{\cup} E^\circ.\sigma' = C^\circ_i \dot{\cup} E^\circ\sigma'$ $\quad \overline{C_i}$ est un facteur de C puisque C°_i est

un facteur de C°.

Soit C_i la contraction de $\overline{C_i}$ on a alors:

(4) $C_i = C^\circ_i \dot{\cup} E^\circ_i$ où E°_i la partie égalitaire de C_i est telle que $E^\circ_i \subset E^\circ\sigma'$

Montrons que $E_i^\circ.\sigma''$ n'est pas une i-tautologie et qu'il est terminal.

$E_i^\circ.\sigma'' \subset E^\circ\sigma'.\sigma'' = E'^\circ_i$ or E'°_i vérifie ces 2 conditions donc

$E_i^\circ.\sigma''$ aussi. Posons $\sigma_i=\sigma''$ on a alors

- C_i contraction d'un facteur de $C \in S$ (les littéraux factorisés étant les mêmes que pour C_i°)
- C_i° est la partie non égalitaire de C_i (4) et $C_i = C_i^\circ \dot{\cup} E_i^\circ$

  $C_i^\circ.\sigma_i = C'^\circ_i$ (3)
- $E_i^\circ.\sigma_i$ est une non i-tautologie terminale

On peut résumer par le schéma suivant:

$$\begin{array}{ccccc} C & = & C^\circ & \dot{\cup} & E^\circ \\ \Downarrow & & \downarrow \sigma' & & \downarrow \sigma' \\ \overline{C}_i & = & C_i^\circ & \dot{\cup} & E^\circ\sigma' \\ \Downarrow & & \downarrow \sigma_i & & \downarrow \sigma_i \\ C'_i & = & C'^\circ_i & \dot{\cup} & E'^\circ_i \end{array} \xrightarrow{\text{contraction}} C_i = C_i^\circ \dot{\cup} E_i^\circ$$

2°) Considérons le cas où C'°_i est une résolvante de C'°_j et C'°_k et C_i° un facteur d'une résolvante de C_j° et C_k°. Nous supposons C_j et C_k sans variables communes.
D'aprés l'hypothèse de récurrence (2) on a alors:

(5) $C_j = C_j^\circ \dot{\cup} E_j^\circ$ $\quad C'^\circ_j = C_j^\circ.\sigma_j$ $\quad E'^\circ_j.\sigma_j$ terminale et non i-tautologie

$C_k = C_k^\circ \dot{\cup} E^\circ_k$ $\quad C'^\circ_k = C_k^\circ.\sigma_k$ $\quad E'^\circ_k.\sigma_k$ terminale et non i-tautologie

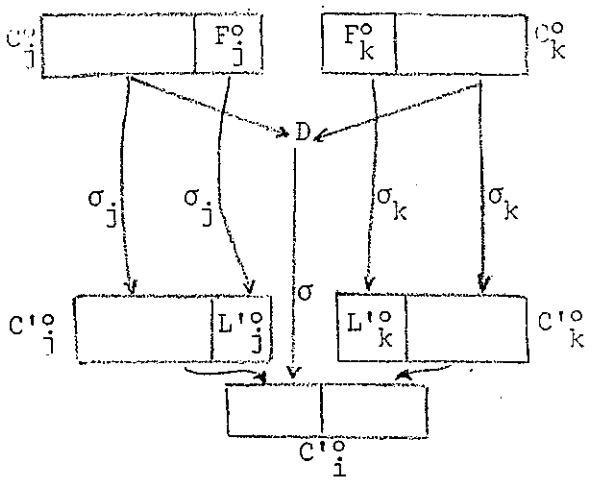
Soient F_j° et F_k° les deux sous-ensembles de C_j° et C_k° resolus, E'°_j et E'°_k les littéraux resolus dans C'°_j et C'°_k

F_j° et $\overline{F_k^\circ}$ sont unifiables et comme $\mathcal{R}'^\circ$ est généralisée par $\mathcal{R}^\circ$ on a:

$$F_j^\circ\sigma_j = \{L'^\circ_j\} \quad \text{et} \quad F_k^\circ\sigma_k = \{L'^\circ_k\}$$

(6) $C'^{\circ}_i=(C'^{\circ}_j-\{L'^{\circ}_j\}) \cup (C'^{\circ}_k-\{L'^{\circ}_k\})=(C^{\circ}_j-F^{\circ}_j)\sigma_j \cup (C^{\circ}_k-F^{\circ}_k)\sigma_k$

Donnons le schéma explicatif suivant:



Posons $\sigma=\sigma_j \dot{\cup} \sigma_k$ (σ est bien une substitution puisque $v(\sigma_j) \cap v(\sigma_k)=\phi$)

On a alors d'aprés (6)

$C'^{\circ}_i= ((C^{\circ}_j-F^{\circ}_j) \cup (C^{\circ}_k-F^{\circ}_k))\sigma$ (proposition I.3.4)

Soit (7) $D=(C^{\circ}_j-F^{\circ}_j) \cup (C^{\circ}_k-F^{\circ}_k)$ alors $C'^{\circ}_i=D\sigma$

$F^{\circ}_j \cup \overline{F}^{\circ}_k$ étant l'ensemble des littéraux unifiés dans la resolvante de C_j et C_k et C°_i étant un facteur de cette resolvante $\exists\sigma_1$ telle que:

$C^{\circ}_i=((C^{\circ}_j-F^{\circ}_j) \cup (C^{\circ}_k-F^{\circ}_k))\sigma_1 =D\sigma_1$

Considérons les 3 clauses $A=D \dot{\cup} (E^{\circ}_j \cup E^{\circ}_K)$ $B=C'^{\circ}_i \dot{\cup} (E^{\circ}_j \cup E^{\circ}_k)\sigma$ et C°_i

B est une instance terminale de A (B=Aσ puisque $C'^{\circ}_i = D\sigma$)
C°_i est une instance de D

C'°_i est une instance de C°_i (puisque $\mathcal{Q}^{\circ}$ généralise $\mathcal{Q}'^{\circ}$)

$E^{\circ}_j \cup E^{\circ}_k$ est la partie égalitaire de A

$(E^{\circ}_j \cup E^{\circ}_k)\sigma$ est la partie égalitaire de B

donc d'aprés le lemme 2 $\exists$ σ' et σ'' telles que

(8) $B=A\sigma'\sigma''$

$C'^{\circ}_i=C^{\circ}_i.\sigma''$

$C^{\circ}_i= D\sigma'$

(9) Soit $\overline{C}_i=A\sigma'=(D \dot{\cup} (E^{\circ}_j \dot{\cup} E^{\circ}_k))\sigma'=D\sigma' \dot{\cup} (E^{\circ}_j \dot{\cup} E^{\circ}_k)\sigma'=C^{\circ}_i \cup (E^{\circ}_j \dot{\cup} E^{\circ}_k)\sigma'$

D'aprés la définition de D (7)

$\overline{C}_i=((C^{\circ}_j-F^{\circ}_j) \dot{\cup} (C^{\circ}_k-F^{\circ}_k) \dot{\cup} E^{\circ}_j \dot{\cup} E^{\circ}_k) \sigma'$

$\overline{C}_i=(((C^{\circ}_j \dot{\cup} E^{\circ}_j)-F^{\circ}_j) \dot{\cup} ((C^{\circ}_k \dot{\cup} E^{\circ}_k)-F^{\circ}_k)) \sigma'$

donc d'aprés (5) $\overline{C}_i=((C_j-F^{\circ}_j) \dot{\cup} (C_k-F^{\circ}_k))\sigma'$

C_i est donc un facteur d'une résolvante de C_j et C_k, les littéraux résolus et factorisés étant les mêmes que pour C°_i.

De plus C°_i est la partie égalitaire de $\overline{C}_i$

Soit C_i la contraction de $\overline{C}_i$:

$C_i=C^{\circ}_i \cup E^{\circ}_i$ avec $E^{\circ}_i \subset (E^{\circ}_j \dot{\cup} E^{\circ}_k)\sigma'$ (d'aprés (9))

Montrons que $E^{\circ}_i.\sigma''$ est une clause terminale non i-tautologie:

$E^{\circ}_i.\sigma'' \subset (E^{\circ}_j \cup E^{\circ}_k)\sigma'\sigma''=(E^{\circ}_j \dot{\cup} E^{\circ}_k)\sigma$ (d'aprés 8)

Comme $\sigma=\sigma_j \dot{\cup} \sigma_k$, d'aprés la proposition I.3.4

$(E^{\circ}_j \dot{\cup} E^{\circ}_k)\sigma \subset (E^{\circ}_j.\sigma_j) \cup (E^{\circ}_k\sigma_k)$

D'aprés l'hypothèse de recurrence (5) $E^{\circ}_j.\sigma_j$ et $E^{\circ}_k.\sigma_k$ sont des clauses terminales non i-tautologies donc $E^{\circ}_i.\sigma''$ aussi d'où en posant $\sigma_i=\sigma''$ on a

C_i contraction d'un facteur d'une resolvante de C_j et C_k (les littéraux resolus et factorisés étant les mêmes que pour C°_i

C°_i est la partie non égalitaire de C_i et $C_i=C^{\circ}_i \dot{\cup} E^{\circ}_i$

$C^{\circ}_i.\sigma_i = C'^{\circ}_i$ (d'aprés 8)

$E^{\circ}_i.\sigma_i$ est terminale et non i tautologie

Donc $D=(C_1,....C_n)$ est bien une i-φ-déduction à partir de S car C_i ne peut être une i-tautologie puisque E_i^o partie égalitaire de C_i admet une instance terminale $E_i^o.\sigma_i$ qui n'en est pas une .
(en effet si $t \neq t \in E_i^o$ alors $(t\sigma_i \neq t\sigma_i) \in E_i^o.\sigma_i$ et $E_i^o.\sigma_i$ serait i-tautologie)

D'autre part $C_n^o = \phi$ et C_n admet C_n^o comme partie non égalitaire donc $C_n = E_n^o$.
ou bien $E_n^o = \phi$ auquel cas $C_n = \phi$

ou bien $E_n^o \neq \phi$ soit $C_n = E_n^o = \bigcup_{K+1}^{m} \{t_k \neq t'_k\}$

or C_n n'est pas une i-tautologie donc $t_k \neq t'_k \quad \forall K \in \{1,....m\}$

d'où le théorème II.3.1

on peut dire que si φ est une méthode de déduction complète et généralisable, le i-φ- méthode associée est complète
Nous allons montrer maintenant qu'elle est consistente.

THEOREME II.3.2
Soit C une clause i-positive qui n'est pas une i-tautologie et dont la partie non égalitaire est vide.
Alors quelquesoit la i-interprétation I dans un univers de Herbrand infini, C n'est pas satisfaite parI.

pour établir ce théorème nous aurons besoin du lemme suivant:

LEMME 3
Soient t et t' deux termes distincts et unifiables, σ_0 un de leurs p.g.u et σ_1 une substitution telle que :

(1) $\sigma_1 = \{x/k\}$, K terme terminal:

(2) $\sigma_1 \not\subset \sigma_0$

Alors $t.\sigma_1 \neq t'.\sigma_1$

Notons que puisque t et t' sont distincts σ_0 existe et est dinstinct de

En outre d'aprés I.3.8 on a:

(1) $\forall x/t \in \sigma_0 \qquad V(t) \cap V(\sigma_0) = \phi$

Supposons $t\sigma_1 = t'\sigma_1$: alors d'aprés la définition des p.g.u $\exists\, \sigma$ tel que

$\sigma_1 = \sigma_0.\sigma$

Soit $\sigma=\{y_1/k_1,\ldots,y_m/k_m\}$

on a donc

$\{x/k\}= A \dot{\cup} B$ avec $A=\{x_i/t_i.\sigma$ tels que $x_i/t_i \in \sigma_0$ et $t_i\sigma \neq x_i\}$

$B=\{y_i/k_i$ tels que $y_i/k_i \in \sigma$ et $y_i \notin V(\sigma_0)\}$

1er cas:

$A=\{x/k\}$ et $B=\phi$ donc $\nexists\ y \in V(\sigma_0)$ tel que $y \notin V(\sigma_0)$ donc $V(\sigma) \subset V(\sigma_0$

$\forall\ i \in \{1,\ldots n\}\ V(t_i) \cap V(\sigma) \subset V(t_i) \cap V(\sigma_0)= \phi$ d'aprés (1)

Donc $t_i.\ \sigma=t_i$ donc $A= \{x_i/t_i\ /t_i \neq x_i\}=\sigma_0=\{x/k\}$

or $x/k \notin \sigma_0$ donc ceci est impossible

2ème cas

$A=\phi$ et $B=\{x/k\}$ donc $\forall\ y \in V(\sigma)$, $y \notin V(\sigma_0) \Rightarrow y=x$ donc

$V(\sigma) \subset V(\sigma_0) \dot{\cup}\{x\}$

de plus $\forall i \in \{1,\ldots.n\}\ t_i\sigma=x_i$ donc en particulier $t_1.\sigma =x_1$

donc t_1 est une variable et $\{t_1/x_1\} \in\sigma$. Or $V(t_1) \cap V(\sigma_0) = \phi$

donc $t_1 \notin v(\sigma_0)$ donc

comme $V(\sigma) \subset V(\sigma_0) \cup \{x\}$ on a $t_1=x$ et $t_1/x_1 = x/k$

donc $x_1= K$

ce qui est impossible puisque K est terminal.

DEMONSTRATION du THEOREME III.3.2

La clause C peut s'écrire $C= \overset{n}{\underset{i=1}{\cup}} \{t_i \neq t'_i\}$ avec $t_i \neq t'_i\ \ \forall i \in \{1,\ldots n\}$

(le cas où $C= \square$ est trivial)

Une clause C étant i-équivalente à sa contraction on peut supposer t_i et t'_i unifiables $\forall i\ \in\{1,\ldots n\}$

Nous allons démontrer le théorème par récurrence sur le nombre de variables N de C

N=O alors soit I une i-interprétation quelconque. C est terminale

Donc

$\forall i \in \{1,...n\}$ $t_i \neq t'_i \Rightarrow t_i \neq t'_i \notin I$ donc $C \cap I = \phi$ donc C n'est pas satisfaite par I

Supposons la propriété vraie pour O,...N-1. Soit C ayant N variables

Soient $\sigma_1,\ldots_n$ les p.g.u de (t_1,t'_1) (t_2,t'_2) $,\ldots.(t_n,t'_n)$ tels que

$\forall x/t \in \sigma_i$ $v(t) \cap v(\sigma_i) = \phi$

soit $x_1/k_1 \in \sigma_1$ $(\sigma_1 \neq \varepsilon$ donc $v(\sigma_1) \neq \phi)$ et K un terme terminal tel que

$x_1/k \notin \sigma_1 \cup \ldots\ldots \cup \sigma_n$. Soit alors $\sigma = \{x_1/k\}$

K existe puisque H est infini. D'aprés le lemme 3 on a

$\forall i \in \{1,...n\}$ $t_i \sigma \neq t'_i \sigma$

Donc $C\sigma$ possède (N-1) variables et vérifie l'hypothèse de récurrence donc C est bien i-insatisfaisable puisque $C\sigma$ l'est.

Nous pourrons énoncer maintenant le théorème de consistence

THEOREME II.3.3

Soit φ une méthode de déduction et (S,F) système de clauses i-positif, F ayant au moins un symbole fonctionnel de rang $n \geq 1$
Alors:

$\exists$ une i-φdéduction d'une clause C à partir de S telle que (1) et (2)

$\Rightarrow$ (S,F) est i-insatisfaisable. Les propriétés (1) et (2) sont les suivantes:

(1) C ne soit pas une i-tautologie

(2) la partie non égalitaire de C est vide

Soit $\underset{\sim}{D}=(C_1,\ldots.C_n)$ une i-φdéduction à partir de S d'une clause C vérifiant (1) et (2).

Toute interprétation qui satisfait une clause satisfait ses facteurs et Toute interprétation qui satisfait 2 clauses satisfait leurs resolvantes.

Donc à fortiori pour toute i-interprétation.
En outre une i-interprétation qui satisfait une clause satisfait sa contraction. Or $C_1,.....C_n$ sont obtenues à partir de clauses de S par ces 3 opérations donc

Si I satisfait S alors I satisfait C.

D'aprés le théorème II.3.2 C n'est satisfaite par aucune i-interprétation dans H(F) donc S est i-insatisfaisable
Notons que la condition (1) est satisfaite par le fait que $\underset{\sim}{D}$ est une i-φdéduction.
On peut énoncer les deux théorèmes II.3.3 et II.3.1 sous la forme suivante:

THEOREME II.3.4

Soit (S,F) un système injectif i-positif tel que F admette un symbole fonctionnel de rang $n \geqslant 1$.
Soit φ une méthode de déduction complète et généralisable.
Alors:
(S,F) est i-insatisfaisable ssi il existe à partir de S une i-φdéduction d'une lcause C telle que C ait une partie non égalitaire vide.

4. UTILISATION DE L'EGALITE FORMELLE

Nous allons maintenant voir comment utiliser de façon pratique l'égalité formelle en démonstration automatique. La première utilisation permet d'éviter certaines redondances dans le graphe des resolutions, redondances dues entre autre au fait que des clauses peuvent avoir des instances communes et que ces instances peuvent donc mener à des déductions identiques.

Limitation des redondances

Donnons un exemple pour illustrer cette application.
Soit $S=\{C_1,C_2,C_3,C_4\}$ l'ensemble des clauses suivantes:

$C_1=\{\overline{P(x,y)},\overline{P(y,z)},P(x,z)\}$ transivité de P

$C_2=\{P(x,x)\}$ réflexivité de P

$C_3=\{P(a,b)\}$

$C_4=\{P(b,a)\}$

Considérons la déduction suivante: $\underset{\sim}{D}=(C_5, C_6)$

resolvante de C_1 et C_3 $\quad C_5=\{\overline{P(b,z)}, P(a,z)\}$

resolvante de C_5 et C_4 $\quad C_6=\{P(a,a)\}$

On voit que C_6 est en fait une instance de C_2 et que cette déduction est sans intérêt.
On peut alors placer une condition sur la clause C_1 et la remplacer par la clause C'_1 .

$C'_1=\{\overline{P(x,y)}, \overline{P(y,z)}, P(x,z), x\neq z\}$

Le système (nous allons le montrer) S est équivalent au système injectif (S',F) avec:

$S'=\{C'_1, C_2, C_3, C_4\}$ et $F=\{a,b\}$

ou plus exactement S est satisfaisable ssi (S',F) est i-satisfaisable et une i-déduction à partir de S' ne pourra jamais générer C_6 .

NOTATION

Soit $\sigma=\{x_1/t_1, \ldots . x_n/t_n\}$ une substitution.

on définit le littéral $L(\sigma)$ par:

$L(\sigma) = (f(x_1, \ldots . x_n) \neq f(t_1, \ldots . t_n))$ où f est un nouveau

symbole fonctionnel d'ordre n.

Si $\sigma = \varepsilon$ on définit alors $L(\varepsilon)$ par:

$L(\varepsilon)=(f(x)\neq f(x))$

THEOREME II.4.1

Soient S ensemble de clauses, A et B deux clauses de S telles que $\exists \sigma_1$ et σ_2 avec

(1) $A\sigma_1 \subset B\sigma_2$

Soit (S',F) le système injectif défini par

(1) $S'=(S-\{B\}) \cup \{B'\}$ avec $B'=B \cup \{L(\sigma_2)\}$ (B remplacé par B')

(2) $F=\{$symboles fonctionnels de S$\} \cup \{f\}$ où f est le symbole de $L(\sigma_2)$

Alors: S est insatisfaisable ssi (S',F) est i-insatisfaisable

DEMONSTRATION DE ⇒ :

Soit $L(\sigma_2)=\{f(x_1,\ldots,x_n)\neq f(t_1,\ldots t_n)$ $\quad (n \geq 1)$

Supposons (S',F) i-satisfaisable et soit I une i-interprétation de (S',F)
Montrons que B est satisfaite par I

Soit $B\sigma$ une instance terminale quelconque de B,σ minimum. on a alors

$$B'\sigma = B\sigma \cup \{L(\sigma_2)\}\sigma$$

(on peut supposer σ_2 minimum donc ayant toutes ses variables dans B)

σ étant terminale on a

$v(\sigma) = v(B)$ donc $v(\sigma) \supset v(L(\sigma_2))$ donc $L(\sigma_2).\sigma$ est terminale donc $B'\sigma$ aussi

B' est satisfaite par I donc $B'\sigma$ aussi donc $B'\sigma \cap I \neq \phi$ donc

$B\sigma \cap I \neq \phi$ ou $\{L(\sigma_2).\sigma\} \cap I \neq \phi$

Si $\{L(\sigma_2).\sigma\} \cap I \neq \phi$ alors $L(\sigma_2).\sigma \in I$ donc

$\{f(x_1,\ldots x_n) \neq f(t_1,\ldots t_n)\}\sigma \in I$

Or I est une i-interprétation donc $f(x_1,\ldots x_n)\sigma = f(t_1,\ldots t_n)\sigma$ donc
σ est un unifieur de $A=\{f(x_1,\ldots x_n), f(t_1,\ldots t_n)\}$
De plus σ_2 est un p.g.u de A donc d'aprés la définition des p.g.u

$$\exists\ \sigma' \text{ tel que } \sigma=\sigma_2.\sigma'$$

Donc $B\sigma=B\sigma_2.\sigma'$ et d'aprés (1) $A\sigma_1.\sigma' \subset B\sigma$ donc comme A est satisfaite par I, $B\sigma$ aussi.
Donc B est satisfaite par I et puisque

$S \subset S' \cup \{B\}$ et que S' est satisfait par I, S également.
Donc (S',F) i-satisfaisable ⇒ S satisfaisable

DEMONSTRATION DE $\Leftarrow$:

Soit S satisfaisable et soit I une interprétation qui satisfait S. I satisfait également B' puisque $B \subset B'$. Donc S' est satisfaite par I.
Soit I' la i-interprétation définie par

$I' = I \cup \{t \neq t \text{ avec } t \in H(F)\} \cup \{\overline{t \neq t'} \text{ avec } t,t' \in H(F),\ t \neq t'\}$

I est une interprétation qui est une extension de I et S' étant satisfaite par I, (S',F) est satisfaite par I'.

Donc S satisfaisable $\Rightarrow$ (S',F) i-satisfaisable

REMARQUE 1

Ce théorème est une extension du théorème de subsumption car A subsume B ssi $\exists\sigma$ tel que $A\sigma \subset B$ donc dans ce cas $\sigma_2 = \varepsilon$ et $B' = B \cup \{f(x) \neq f(x)\}$ qui est une i-tautologie donc qui peut être supprimée.

REMARQUE 2

Dans le cas où $\sigma = \{x/t\}$ i-e n'a qu'un seul composant on peut prendre $L(\sigma) = (x \neq t)$ au lieu de $(f(x) \neq f(t))$ puisque ce sont deux littéraux équivalents.

Donnons un exemple d'utilisation de ce théorème.

Soit à démontrer le théorème suivant: Si une loi de composition est associative, admet un élément neutre et si tout élément est d'ordre 2 alors la loi est commutative.

Nous utiliserons le prédicat $P(x,y,z)$ pour $x * y = z$.

C_1: $\{\overline{P}(x,y,u), \overline{P}(y,z,v), \overline{P}(u,z,w), P(x,v,w)\}$ associativité

C_2: $\{\overline{P}(x,y,u), \overline{P}(y,z,v), \overline{P}(x,v,w), P(u,z,w)\}$ associativité

C_3: $\{P(x,e,x)\}$ e un élément neutre à droite

C_4: $\{P(e,x,x)\}$ e est élément neutre à gauche

C_5: $\{P(x,x,e)\}$ tout élément est d'ordre 2

C_6: $\{P(a,b,c)\}$

C_7: $\{\overline{P}(b,a,c)\}$

négative du théorème: $\forall x\ \forall y\ \forall z.\ P(x,y,z) \Rightarrow P(y,x,z)$

Soient $\sigma_1=\{x/e\}$ alors $C_5\sigma_1=C_3\sigma_1$ $C_5\sigma_1=C_4\sigma_1$

donc on peut remplacer C_3 et C_4 par $C_3 \cup L(\sigma_1)$ et par $C_4 \cup L(\sigma_1)$

De même si $\sigma_2=\{x/a,v/b,w/c\}$ alors $C_6 \subset C_1.\sigma_2$ donc C_1 peut être remplacé par $C_1 \cup L(\sigma_2)$

si $\sigma_3=\{u/a,z/b,w/c\}$ alors $C_6 \subset C_2.\sigma_3$ donc C_1 peut être remplacé par $C_2 \cup L(\sigma_3)$

Donc l'ensemble $S=\{C_1,....C_7\}$ est insatisfaisable ssi $S'=\{C'_1,...C'_7\}$ est i-insatisfaisable avec

$C'_1=\{\overline{P}(x,y,u),\overline{P}(y,z,v),\overline{P}(u,z,w),f_3(x,v,w)\neq f_3(a,b,c),P(x,v,w)\}$

$C'_2=\{\overline{P}(x,y,u);\overline{P}(y,z,v),\overline{P}(x,v,w),P(u,z,w),\ f_3(u,z,w)\neq f_3(a,b,c)\}$

$C'_3=\{P(x,e,x),x\neq e\}$

$C'_4=\{P(e,x,x),x\neq e\ \}$

$C'_5=\{P(x,x,e)\}$

$C'_6=\{P(a,b,c)\}$

$C'_7=\{\overline{P}(b,a,c)\}$

En outre on est assuré que pour une méthode quelconque de déduction, le nombre de clauses générées à partir de S est plus grand qu'à partir de S'. Donc la méthode sera améliorée par l'introduction de ces "conditions" puisque le graphe des resolutions est réduit.

Utilisation de l'égalité formelle dans les systèmes de questions-réponses.

Un certain nombre de systèmes de questions-réponses admettent comme principe de base la démonstration automatique.(cf GREEN-1) La plupart utilisent comme langage de base le langage de calcul des prédicats du 1er ordre. Le "fichier" sur lequel des questions vont être posées est constitué d'un ensemble de clauses (qu'on suppose consistent), la question posée étant associée à un théorème à démontrer.

Par exemple une question du type "qui vérifie P..." peut être formulée sous la forme suivante:

$\exists x$ tel que $P(x)$?

Le théorème à démontrer à partir du fichier est alors le suivant:

$\exists x \; P(x)$

On essaie donc de démontrer ce théorème et de trouver la "valeur" x_0 de x telle que $P(x_0)$ puisse être déductible des clauses constituant le fichier.
On formule alors la question sous la forme suivante:

$\forall x \; (P(x) \Rightarrow$ Réponse $(x))$ qui s'écrit sous forme clausale de la façon suivante:

$\overline{P}(x) \vee$ Réponse(x)

Si l'on suppose qu'il existe vraiement une valeur x_0 de x telle que $P(x_0)$ soit déductible des clauses du fichier S l'ensemble

$S' = S \cup \{\overline{P}(x)\}$ est inconsistent.

On peut donc trouver une déduction de la clause vide à partir de S' donc une déduction de Réponse (x_0) à partir de

$S'' = S \cup \{\overline{P}(x), \text{Réponse } (x)\}$ par une méthode de resolution

Exemple:
Considérons les informations suivantes:

1: Pierre est le père de Paul
2. Marie est la mère de Jacques
3. Pierre est le mari de Marie
4. Si deux individus ont le même père alors ils sont frères
5. Le mari de la mère d'un individu est le père de cet individu

question posée:

6. Qui est frère de Paul?

On peut représenter le fichier par l'ensemble de clauses suivantes:

1. {père (Pierre, Paul)}
2. {mère (Marie,Jacques)}
3. {mari(Pierre,Marie) }
4. {$\overline{\text{père}}$ (x,y), $\overline{\text{père}}$ (x,z), frère(y,z)}
5. { $\overline{\text{mari}}$(x,y),$\overline{\text{mère}}$(y,z),père(x,z)}

6. {$\overline{\text{frère}}$(x,Paul),Réponse (x)} question

résolvante 6.4: 7.{ $\overline{\text{père}}$(x,y),$\overline{\text{père}}$(x,Paul),Réponse (y)}

résolvante 7.1: 8.{ $\overline{\text{père}}$(Pierre,y),Réponse(y)}

résolvante 8.5: 9.{ $\overline{\text{mari}}$(Pierre,y), $\overline{\text{mère}}$(y,z), Réponse(z)}

résolvante 9.2: 10.{ $\overline{\text{mari}}$ (Pierre,Marie), Réponse(Jacques)}

résolvante 10.3 11 { Réponse(Jacques).}

D'où une première réponse à la question. On peut en trouver une seconde qui peut paraitre surprenante, mais qui est parfaitement déductible des axiomes.

résolvante 8.1: 12 Réponse(Paul)

D'où la réponse suivante: Paul est son propre frère.

Pour éviter de telles réponses, on peut alors introduire des "conditions" dans les clauses grâce à l'égalité formelle.
L'information 4 aurait dû en effet être formulée autrement:
si deux individus distincts ont le même père alors ils sont frères.
Sous forme clausale ceci peut s'écrire:

4' {$\overline{\text{père}}$(x,y),$\overline{\text{père}}$(x,z), frère(y,z),y$\neq$z}

d'où la première réponse identique à celle trouvée précédemment

résolvante6.4': 7' {$\overline{\text{père}}$(x,y),$\overline{\text{père}}$(x,Paul),y$\neq$Paul, Réponse (y)}

résolvante 7'.1: 8' {$\overline{\text{père}}$(Pierre,y),y$\neq$Paul,Réponse(y)}

résolvante 8'.5: 9' {$\overline{\text{mari}}$(Pierre,y),mère(y,z),z$\neq$Paul,Réponse (z)}

résolvante 9.2: 10' {$\overline{\text{mari}}$(Pierre,Marie), Jacques$\neq$Paul,Réponse (Jacques)}

Contraction 10': 10" {$\overline{\text{mari}}$(Pierre,Marie),Réponse(Jacques)}

résolvante 10".3: 11' {Réponse (Jacques)}

Par contre la seconde réponse n'aurait pû être trouvée:

résolvante 8'.1: 12' {Paul≠Paul,Réponse (Paul)} qui est éliminée

puisque i-tautologie.

Il faut cependant remarquer que l'utilisation de l'égalité formelle nécessite certaines précautions. On sait que la clause $\{t_1 \neq t_2\}$ terminale ne peut être interprétée dans l'univers de Herbrand comme "vraie" que si (et seulement si) t_1 et t_2 sont égaux formellement. Donc deux individus dont les "noms" sont distincts seront interprétés comme étant différents dans l'univers de l'interprétation. Il faut donc écrire les informations en tenant compte de ce fait.
Entre autre on ne pourra jamais montrer par exemple que Pierre≠Marie.
Mais ceci semble quand même assez naturel.
Le problème apparait lorsqu'on introduit des fonctions de Skölem (soit dans la question, soit dans la traduction des informations sous forme clausale).

Nous venons donc de voir un certain nombre d'utilisations possibles de l'égalité formelle en démonstration automatique.
Un problème reste pourtant irrésolu en ce qui concerne les fonctions de Skölem. Il semble alors naturel de mêler l'égalité formelle et l'égalité logique dans un même formalisme de resolution.
En particulier on peut différencier par exemple un certain sous-ensemble de vocabulaire terminal pour lequel les axiomes de l'égalité formelle resteraient valides, et utiliser l'égalité logique pour les autres.

C'est cette voie que nous choisirons pour les recherches futures, ainsi que l'extension du traitement de l'égalité formelle à d'autres prédicats ayant de telles propriétés qu'ils puissent être traités formellement. Ceci permettrait entre autres d'utiliser la démonstration automatique dans les analyseurs de langage naturels qui très souvent ont besoin de traitements formels sur les "arbres".

Chapitre III

OEDIPE : UN PROGRAMME DE DEMONSTRATION AUTOMATIQUE PERMETTANT LE TRAITEMENT DE L'EGALITE FORMELLE

OEDIPE est constitué d'un ensemble de procédures éxécutant les divers algorithmes que nous aurons vu dans les 2 chapitres précédent.Le présent chapitre constitue une vue d'ensemble de ce programme.
Nous donnerons tout d'abord la syntaxe du langage dans lequel doivent être écrits les problèmes pouvant être traités par OEDIPE et les résultats qu'il transmet.
La manière dont les clauses sont codées en mémoire et la grammaire context-free du langage d'entrée d'OEDIPE seront explicités dans le paragraphe I.
Les différents algorithmes concernant le traitement des clauses (unifications, copie d'une clause, utilisation d'un dictionnaire de liste de termes terminaux) constituent le deuxième paragraphe.
Nous donnerons ensuite la méthode de déduction (SL resolution) et la stratégie utilisée, puis différents exemples de problèmes posés par OEDIPE.

III.1 SYNTAXE ET CODAGE DES CLAUSES

Syntaxe des clauses utilisées en entrée

Nous représentons en fait une clause par une liste ordonnée de littéraux plutôt que par un ensemble de littéraux.
Ces littéraux sont constitués d'un signe suivi d'une formule atomique (+ pour affirmation, - pour négation) le signe de conjonction étant omis. E est le symbole de l'égalité formelle.

<clause> ::= <liste littéraux >.

<liste littéraux> ::= <littéral> <liste littéraux> |< vide>

<littéral> ::= <signe> <formule atomique>

<formule atomique> ::= <symbole> | <symbole> (< liste termes>) |

E (<termes >, <termes>)

< liste termes > ::= < terme > , < liste termes > | < terme >

< terme > ::= < variable > | < symbole > | < symbole > (< liste termes >)

< variable > ::= * < caractère > < liste a;n >

< symbole > ::= < caractère > < liste a.n >

< liste a.n > ::= < caractère a.n > < liste a.n > | < vide >

< caractère a.n > ::= 0 | 1 .....9 | A | B .....Y | Z

< caractère > ::= < caractère a.n > | | < | > | .....

< signe > ::= + | -

Un problème est constitué d'un commentaire, d'une borne fixée quand aux niveaux des réfutations à chercher, et de deux listes de clauses supposées factorisées.
La première liste (éventuellement vide) doit être consistente. La seconde constitue l'ensemble de support.

<problème> ::= <commentaire> < borne niveau> < partie consistente>

<ensemble de support>

<commentaire> ::= <liste de caractères>

<borne niveau> ::= <nombre positif>

<partie consistente> ::= < liste clauses> FIN | < vide> FIN

<ensemble de support> ::= < liste clauses> FINAX

Voici un problème:

PROBLEME DES PSYCHIATRES:

20

-PS(* Y) - P(*Y,*Z) + MALADE (* Z)

+P(*Y,*Y) + MALADE (* Y)

+PS(A)

FIN

-MALADE(A)
FINAX

ensemble de support

Codage des clauses en mémoire

Dans toute la suite du chapitre les algorithmes seront exprimés en ALGOL-W. En particulier la structure des données fera largement appel à la notion d'enregistrement disponible dans ce langage.
Rappelons qu'un enregistrement (record) est en fait une suite de mémoires contigues pouvant chacune avoir une valeur de type simple dont l'adresse (référence) d'un enregistrement.

Nous appelerons expressions un terme ou une liste de termes.
3 types d'enregistrements seront utilisés:

```
1  record   ARBRE (integer NOM; référence (ARBRE,VARIABLE)FILS,FRERE);

2  record   VARIABLE(reférence(ARBRE,VARIABLE,SINGLE)ALT;

                    référence(ARBRE,VARIABLE)SUIV);

3  record   SINGLE (reférence(VARIABLE)ADRESSE);
```

A chaque symbole du langage on associe un nombre entier, son code, de manière injective, avec les conventions suivantes:

$$\begin{cases} \text{si } \alpha \text{ est différent de } \beta \text{ alors: } code(\alpha) \neq code(\beta) \\ Code(\alpha) \in N \end{cases}$$

En fait nous allons définir récursivement comment coder une liste d'expressions.
A chaque liste d'expressions on associe une valeur référence de la façon suivante:
Une liste d'expression étant définie par:

<liste d'expression> ::=< expression vide>|< expression> ,< liste d'expressi

les règles suivantes doivent être respectées:

Règle 1. (valeur référence d'une liste d'expressions)
La valeur référence d'une liste vide est null.
La valeur référence d'une liste l=e,l' où e est une expression et l' une liste, est l'adresse d'un enregistrement

$$\begin{cases} \text{de type VARIABLE si } e \text{ est variable} \\ \text{de type ARBRE si } e \text{ n'est pas une variable.} \end{cases}$$

La valeur référence de l est aussi celle de l'occurence de l'expression e qui est la première de la liste

Règle 2 : (valeur référence d'une variable)
La valeur référence de l'occurence d'une variable est l'adresse d'un enregistrement de type VARIABLE.
Parmi toutes des occurences d'une même variable dans une même clause l'une d'entre elles est appelée occurence principale.
Le premier champ (ALT) de l'occurence d'une variable dans une lcause est égal à:

- null si cette occurence est l'occurence principale
- la valeur référence de l'occurence principale de cette variable, si l'occurence n'est pas principale

Règle 3: (valeur du dernier champ d'expression ne figurant pas dans une liste terminale)
Pour toute occurence d'une expression e, si e figure comme première expression d'une liste l (l=e,l') et si l n'est pas une liste terminale alors:

le dernier champ de l'enregistrement ayant la valeur référence de l comme adresse, a pour valeur la valeur référence de la liste l'.

Règle 4 (valeur référence d'un terme qui n'est pas une variable)

La valeur référence d'un terme e qui n'est pas une variable (par exemple $f(l_1)$ où l_1 est éventuellement vide) et qui figure comme occurence de la première expression d'une liste l=e,l' non terminale, est l'adresse d'un enregistrement du type ARBRE:

Le premier champ de cet enregistrement a pour valeur code(f)
le deuxième champ (fils) a pour valeur la valeur référence de l_1
le dernier champ (frère) a pour valeur la valeur référence de l'
(en accord avec la règle 3)

Règle 5 (injective des valeurs référence des listes non terminales)

Deux occurences distinctes de deux listes non terminales ont des valeurs référence distinctes.

Règle 6: (dictionnaire des listes terminales)

Deux occurences distinctes d'une même liste terminale ont la même valeur référence. Si une liste terminale est constituée par l=e,l' alors la valeur référence ad(l) de l est l'adresse d'un enregistrement de type ARBRE dont le premier champ a une valeur négative. Cette valeur est la clé de la liste l.

Si $e=f(l_1)$ alors la liste $l=f(l_1),l'$ où l_1 et l' sont des listes terminales.

Le second champ (fils) de l'enregistrement dont l'adresse est ad(l) a pour valeur l'adresse d'un enregistrement de type ARBRE (code(f), $ad(l_1)$,ad(l'))

En outre toutes les listes terminales ayant la même clé sont regroupées en liste, un tableau de références permettant d'accéder à toutes les listes (dictionnaire).
La clé d'une liste terminale est définie récurssivement par:

Si l est vide alors clé(l)=0

Si $l=f(l_1),l_2$ alors clé(l)=$-\varphi$ (code(f),-clé(l_1),-clé(l_2))

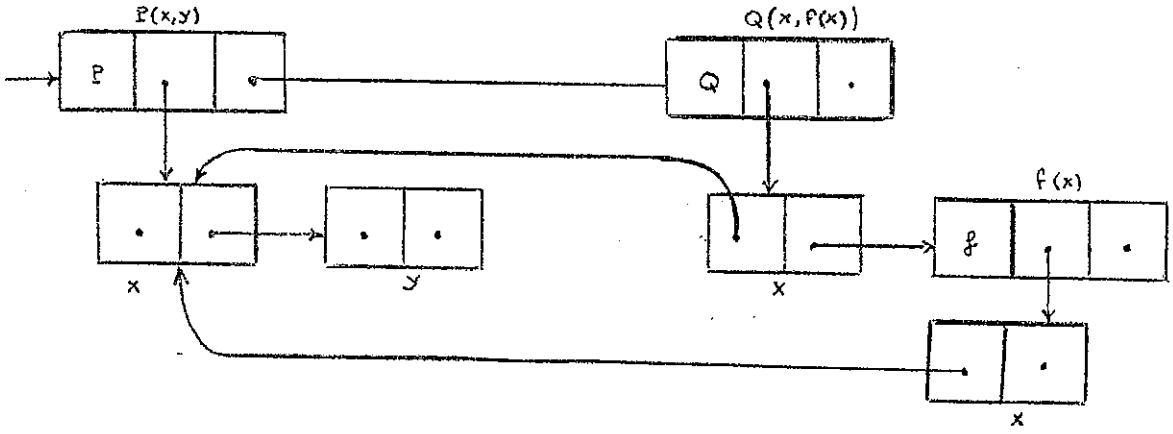
où φ est une fonction $\varphi:N^3 \rightarrow \{n \in N$ tels que $1 \leqslant n \leqslant T\}$ où T est

un nombre entier positif qui est la taille du dictionnaire.

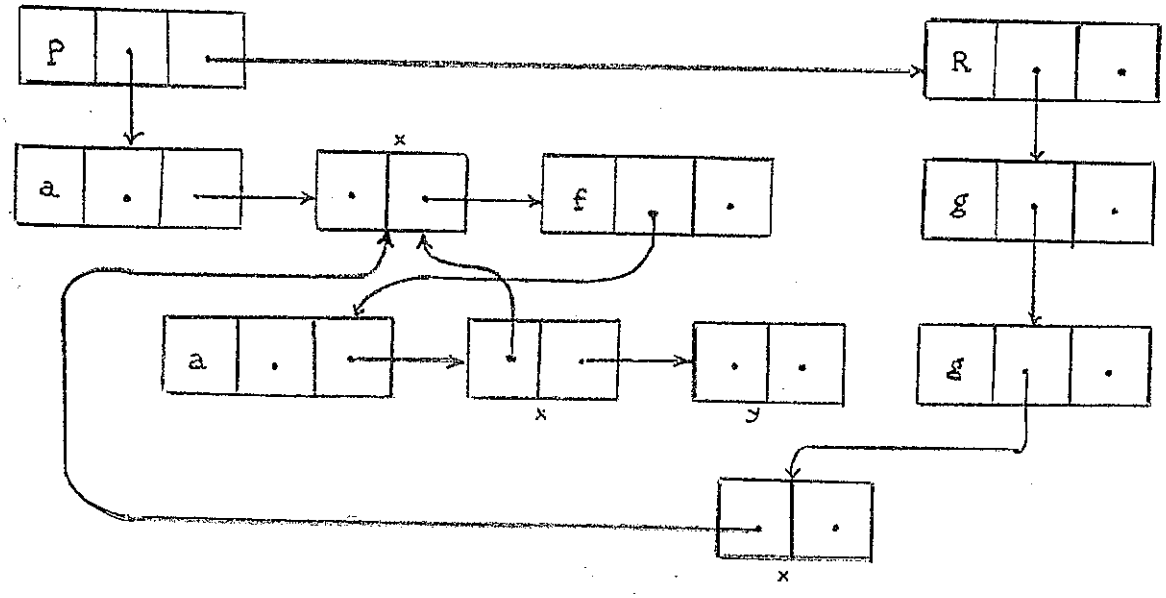
On peut déduire de ces règles que deux occurences distinctes de listes d'expressions non terminales ont des valeurs références distinctes. Par contre des occurences d'une même liste terminale ont toujours la même valeur référence

Voici quelques exemples de codages de listes d'expressions:

La liste P(x,y),Q(x,f(x)) sera codée par:



La liste d'expressions P(a,x,f(a,x,y)),R(g(g(x))) sera codée par:



Utilisation du dictionnaire des listes terminales:

Soient à coder les listes suivantes

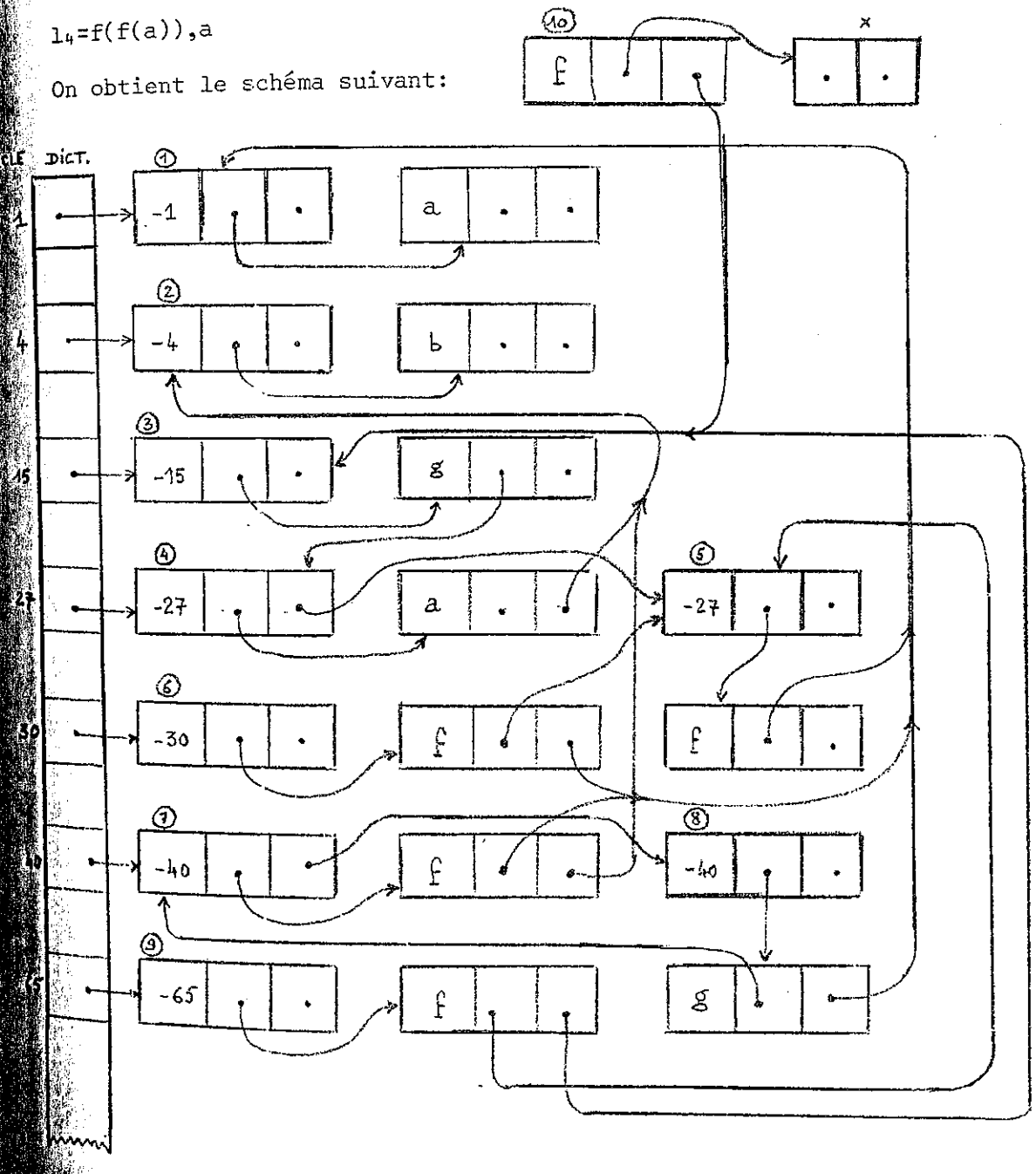
l_1=f(x),g(a,b)

l_2=f(f(a)),g(a,b)

l_3=g(f(a),b),a

l_4=f(f(a)),a

On obtient le schéma suivant:



la liste n° 1 représente: a

2 : b

3 : g(a,b)

4 : a,b

5 : f(a)

6 : f(f(a)),a

7 : f(a),b

8 : g(f(a),b),a

9 : f(f(a)),g(a,b)

10 : f(x),g(a,b)

L'intéret d'un tel dictionnaire réside en plusieurs points:
-les test d'égalité, de non égalité et les procédures d'unifications seront plus rapides
-Pour des listes terminales utilisées plusieurs fois dans un graphe de résolution, la place prise en mémoire est optimisée.
-La recopie des clauses est plus rapide

On pourrait être tenté d'utiliser un dictionnaire pour les listes non terminales. Cependant l'occurence d'une liste ayant des variables est déterminante. Les variables étant en effet quantifiées universellement pour chaque clause, l'occurence de la liste x,f(x) dans une clause C et l'occurence de x,f(x) dans une autre clause C' ne peuvent être représentées par la même adresse puisqu'une substitution { x/t } appliquée à C ne doitpas intervenir sur C'.
C'est un des problèmes que nous nous proposons d'étudier dans le futur.

On peut donc résumer ces quelques principes en disant que pour les listes non terminales, c'est chacune de leurs occurences que nous codons en mémoire, tandis que pour les listes terminales toutes les occurences d'une même liste ont le même codage (c'est à dire valeur référence).

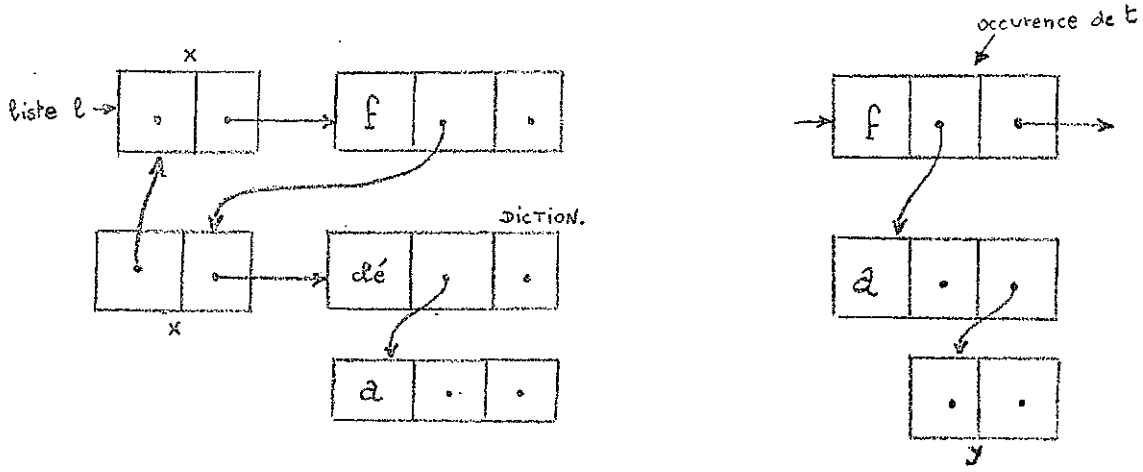
III.2 ALGORITHMES PRINCIPAUX

Algorithme d'unification de 2 listes de termes

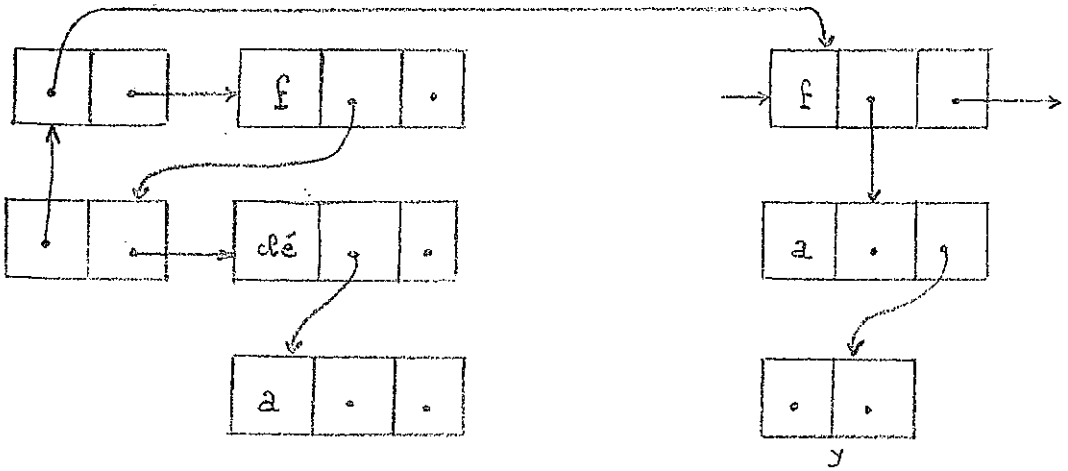
Nous allons donner ici la procédure qui procède à l'unification de 2 listes de termes.
Donnons tout d'abord une idée de la façon dont on procède pour créer une instantiation..

Soit une occurence de la liste l=x,f(x,a) et une occurence du terme t=f(a,y), occurence figurant dans une certaine liste.

l'occurence de la liste l sera représentée par la schéma ci-dessous ainsi que le terme t



Alors l'instance l.σ avec σ={x/t} sera représentée par:



Evidemment toutes les occurences de x dans la clause où figure l sont de ce fait substituées puisque toutes référent à l'occurence principale de x qui elle-même pointe vers le terme qui la remplace. Il ne reste plus alors qu'à recopier la clause ainsi obtenue, en remplaçant chaque occurence de x par t.
Voici donc l'algorithme d'unification basé sur ce principe.
C'est une fonction ayant comme arguments les valeurs référence de 2 listes à unifier.
Son évaluation donne true si ces 2 listes sont inifiables, false sinon. En outre après l'appel d'une telle fonction, des substitutions telles que nous venons de les décrire subsistent. Si l'on recopie alors une de ces deux listes ainsi unifiées on obtient l'instance commune la plus générale.

```
Logical procedure UNIFIE (référence(ARBRE,VARIABLE) value X,Y);

begin
        procedure OCCUR(référence(ARBRE,VARAIBLE)value X,Y);

        comment teste si la variable X a une occurence dans la liste y
        et si oui stoppe l'unification;
        while  ¬ TERMINAL (Y)  do
        begin
          référence (ARBRE,VARIABLE) Y1 ;
          Y1:=REPRISE(Y)
          if Y1 ¬=X then
          begin
            if Y1 is ARBRE then  OCCUR(X,FILS(Y1));
            Y:= AVFRERE (Y)
          end
          else   goto   FINUNIFIE
        end OCCUR;

        logical TEST; référence (ARBRE,VARIABLE) FX,FY;
        TEST:=false;
        Boucle:
        if TERMINAL (X) and TERMINAL(Y) then  TEST:=(X=Y)
        else
        begin
          FX:=REPRISE(X); FY:=REPRISE(Y);
          if  FX ¬=FY  then
          begin
             if (FX = null) or (FY=null) then goto FINUNIFIE;
             if  FX is  VARIABLE then
```

```
            begin
                if FY is ARBRE then OCCUR(FX,FILS(FY));
                ALT(FX):=FY; POINTEUR:=POINTEUR+1; DICVAR(POINTEUR):=FX
            end
            else
            if FY is VARIABLE then
            begin
                OCCUR(FY,FILS(FX));
                ALT(FY):=FX; POINTEUR:=POINTEUR+1;DICVAR(POINTEUR):=FY
            end
            else
            if(NOM(FX) ¬=NOM(FY)) or ¬(UNIFIE(FILS(FX),FILS(FY))) then goto
                FINUNIFIE

        end;
        X:AVFRERE(X); Y:=AVFRERE(Y);
        goto BOUCLE
    end
    FINUNIFIE:
    TEST
end UNIFIE;
```

Dans cette procédure DICVAR est le tableau des adresses des variables ayant été modifiées lors des subsistutions effectuées (POINTEUR est initialisé à 0 au départ, et est le numéro de la dernière variable modifiée)
TERMINAL est une fonction permettant de tester si une liste est terminale en testant le nom de l'arbre à qui il réfère (négatif ou positif suivant le cas).

```
logical procedure TERMINAL (référence(ARBRE,VARIABLE) value X);

(X=null) or ((X is ARBRE) and (NOM(X) < 0));
```

REPRISE permet de trouver quel est le terme par lequel a été substitué une variable ou de sélecter l'adresse du 1er élément d'une liste terminale.

```
référence (ARBRE,VARIABLE) procedure REPRISE (référence(ARBRE,VARAIBLE)valueX);
begin
    while(X is VARIABLE ) and (ALT(X)¬= null) and ¬(ALT(X) is SINGLE)do X:=ALT(
    if (X is ARBRE) and (NOM(X) < 0) then FILS(X) else X
end REPRISE;
```

Enfin AVFRERE donne le terme suivant d'un terme d'une liste:

```
référence (ARBRE,VARIABLE) procédure AVFRERE (référence (ARBRE,VARAIBLE)valueX);
if X is VARIABLE then SUIV(X) else
if NOM(X) < 0  then FRERE (FILS(X)) else FRERE(X);
```

Une fois deux listes unifiées il reste à faire le copie de leur instance commune la plus générale (si bien sûr elles sont unifiables)

Algorithmes de copie d'une liste d'expressions

Cette procedure est une fonction ayant une liste (où les variables ont pu éventuellement être substituées par des termes) comme argument et donnant comme résultat de son évaluation l'adresse d'une liste identique,où les variables ont cependant été recopiées.
En outre dès qu'elle trouve une liste terminale non encore crée elle cherche dans le dictionnaire des listes terminales si cette liste y figure et si non l'y ajoute. C'est la procédure RECHERCHE qui effectue ce travail (nous le verrons plus loin). Dès qu'une nouvelle variable est recopiée on note l'adresse de sa recopie dans un enregistrement du type SINGLE, le champ ALT de la variable copiée piontant vers cet enregistrement.

```
référence (ARBRE,VARIABLE) procédure COPIELIT(référence (ARBRE,VARIABLE)valueX);

begin
   reference(ARBRE,VARIABLE) RECOP;
   if TERMINAL(X) then RECOP:=X
   else
   begin
      reference(ARBRE,VARIABLE)X ; X :=AVFRERE(X); X:=REPRISE(X);
      If X is VARIABLE then
      begin
         if ALT(X)=null then
         begin
            RECOP:=VARIABLE(null,null);ALT(X):=SINGLE(RECOP);
            POINTEUR:=POINTEUR+1; DICVAR(POINTEUR):=X;
            SUIV(RECOP):= COPIELIT(X )
         end
         else  RECOP:=VARIABLE(ADRESSE(ALT(X)),COPIELIT(X ))
      end
      else
      RECOP:=RECHERCHE(NOM(X),COPIELIT(FILS(X)),COPIELIT(X ))
   end;
   RECOP
end  COPIELIT;
```

Une fois les unifications et les recopies terminées il ne reste quà restituer aux variables modifiées leur état initial (c'est à dire remettre à null leurs champs ALT)

```
procedure MISANULL;
while POINTEUR ¬=0 do
begin
   ALT(DICVAR(POINTEUR)):=null; POINTEUR:=POINTEUR-1
end;
```

Algorithme de recherche dans le dictionnaire les listes terminales

La procédure RECHERCHE est une fonction ayant comme argument unesymbole fonctionnel f, et deux références vers des listes l_1 et l_2.
Elle va alors créer la liste:

$l=f(l_1),l_2$ où $l_1$et l_2 sont éventuellement vides.

Si l_1 et l_2 sont des listes terminales alors RECHERCHE va créer éventuellement dans le dictionnaire des listes terminales la liste l (si elle n'y est déjà pas) et donnera comme résultat l'adresse de cette liste.
Par contre si l_1 et l_2 n'est pas terminale RECHERCHE crée un enregistrement de type ARBRE ayant f comme NOM, les valeurs référence de l_1 et l_2 comme FILS et FRERE. Il donne alors comme résultat l'adresse de l'arbre créé.

```
référence(ARBRE) procédure RECHERCHE(integer FONC; reference(ARBRE.
                                                  VARIABLE)valueX,Y)
begin
   reference(ARBRE)Z;
   if TERMINAL (X) and TERMINAL (y) then
   begin
      integer  CLEX ,CLEY , CLE; reference(ARBRE)ZZ;
      CLEX :=if x=null then 0 else-NOM(X);
      CLEY :=if Y=null then 0 else-NOM(Y);
      CLE:=PHI(FONC,CLEX,CLEY);
      Z:=DICTIONNAIRE (CLE);
      while Z¬=null  do
      begin
         ZZ:=FILS(Z);
         if (NOM(ZZ)=FONC) and (FILS(ZZ)=X) and (FRERE(ZZ)=Y) then goto FIN;
         Z:FRERE(Z)
      end;
      Z:=DICTIONNAIRE(CLE):=ARBRE(-CLE,ARBRE(FONC,X,Y),(DICTIONNAIRE(CLE));
   end
   else Z:=ARBRE(FONC,X,Y);
   FIN:
   Z
end  RECHERCHE;
```

PHI est une fonction permettant le clacul des clés récursivement comme nous l'avons vu au paragraphe III.1.
Ces quelques procédures de bases permettant de construire les algorithmes de stratégie et de déduction utilisées dans OEDIPE.

III.3 STRATEGIE ET METHODE DE DEDUCTION UTILISEES PAR OEDIPE

La méthode de déduction que nous avons choisie d'implémenter, aprés plusieurs essais, est la S-L résolution (chapitre I.5). Cette méthode a l'avantage de minimiser le nombre de déductions possibles à partir d'un ensemble de clauses S, d'être linéaire, de générer des preuves de niveau minimal et enfin de se prêter à l'utilisation des stratégies diagonales. L'ensemble des clauses de départ est supposé factorisé.

S-L déductions

Rappelons qu'une S-L déduction est une suite de "pseudo-clauses" constituées de A et de B littéraux (les A littéraux étant en quelque sorte des ancêtres).

Soit $D=(C_1,C_2,\ldots,C_n)$ une telle déduction:

C_1 est une "chaîne" de l'ensemble de support.

C_{i+1} est obtenu à partir de C_i par réduction, expansion avec une clause d'entrée, ou tronquation.
L'opération quelle qu'elle soit est suivie d'une "contraction" sur les littéraux égalitaires etd'un test pour vérifier si ce n'est pas une i-tautologie.
Le niveau d'une telle déduction est le nombre de réductions et d'expansions effectuées.

-Lorsque C_{i+1} est obtenue par réduction on a alors le schéma suivant:

C_i: $L_1\ L_2\ \ldots\ L_k\ \ldots\ \boxed{L'_1}\ \ldots\ \boxed{L'_j}\ \ldots$

C_{i+1}: $(L_1\ L_2\ \ldots\ L_{k-1}\ L_{k+1}\ \ldots\ \boxed{L'_1}\ \ldots\ \boxed{L'_j}\ \ldots)\sigma$ { avec $L_k\sigma = L'_j\sigma$ (factorisation) ; ou $L_k\sigma = \overline{L'_j}\sigma$ (resolution ancestrale) }

Tout B-littéral de C_{i+1} admet un "ancêtre" dans C_i et tout B-littéral de C_i sauf L_k admet un descendant dans C_{i+1}.

-Lorsque C_{i+1} est obtenu par expansion, le littéral sélecté étant le plus à gauche, avec une clause d'entrée C le schéma est le suivant:

C_i : $\underline{L_1}\ L_2\ \ldots\ \boxed{L'_1}\ \ldots\ \ldots$ $C=M_1\ldots M_k\ \overline{M}_{k+1}$

C_{i+1} : $(M_1\ldots\ldots M_k\ \boxed{L_1}\ L_2\ldots\ \boxed{L'_1}\ \ldots)\sigma$ avec $M_{k+1}\sigma=L_1\sigma$

Tout B-littéral de C_i, sauf L_1, admet un descendant dans C_{i+1}.

Tout B-littéral de C_{i+1}, sauf les nouveaux, admet un ancêtre dans C_i.

-Lorsque C_{i+1} est obtenu par tronquation on a alors:

C_i : $\boxed{L_1}\ \boxed{L_2}\ \boxed{L_3}\ \ldots\ \boxed{L_k}\ L_{k+1}\ldots L_n$

C_{i+1} : $L_{k+1}\ldots L_n$

On peut faire les remarques suivantes:

1) Les B-littéraux réductibles figurent mécessairement dans la partie des B-littéraux les plus à gauche.
2) Les B-littéraux non réductibles admettent nécessairement un ancêtre.
3) Un B-littéral non réductible ne peut avoir de descendant réductible.

Coût et heuristique

On définit le coût d'une dérivation comme étant la somme des expansions et des réductions effectuées. Nous le noterons g.
D'autre part nous assimilerons une dérivations à la clause qu'elle dérive (en remarquant bien sûr que deux clauses identiques peuvent être générées par des dérivations distinctes et donc ne doivent pas être confondues).

L'heuristique h d'une clause doit être une estimation minimale de la distance de la solution passant par cette clause. Il faut pouvoir bien sûr être capable de la calculer avant de générer cette clause. C'est en fait par l'intermédiaire de son ancêtre immédiat que nous évaluerons l'heuristique d'une clause.

On définit donc le poids d'un littéral de la façon suivante (l(C) désignant la longueur d'une clause):

DEFINITION III.3.1

Soit L un B-littéral d'une pseudo clause C.

Soit R_L={C',C' chaine d'entrée,C'résolvable avec C,L étant le littéral résolu}

On définit le poids de L par:

$$P(L) = \begin{cases} +\infty \text{ si } R_L = \phi \\ \min\{l(C'), C' \in R_L\} \text{ si } R_L \neq \phi \end{cases}$$

On a alors $P(L) \geqslant 1$.

Etant donné une clause C de niveau n,chaque B-littéral L de C augmente le niveau probable de réfutations passant par C de 1 si L est déductible et de P(L) si L n'est pas réductible.
Mais un tel calcul ne peut être fait qu'une fois C connu. On est donc amené à faire ces calculs sur l'ancêtre immédiat de C.

DEFINITION III.3.2 : heuristique

Soit L un B-littéral d'une clause C.

Si g(C)=0 (C clause de départ) h(L)=P(L)

Si $g(C) \geqslant 1$ $h(L) = \begin{cases} 1 \text{ si L est réductible dans C} \\ P(L') \text{ sinon (L' ancêtre immédiat de L)} \end{cases}$

L'heuristique d'une clause est alors $h(C) = \sum_{\substack{\text{LB-litt} \\ \text{de C}}} P(L)$

Cette définition est bien valide car tout B-littéral non réductible admet un ancêtre si la clause où il figure n'est pas une clause de départ.

REMARQUE III.3.1

Soit L un B-littéral d'une clause et L' un ancêtre de L.
On a alors:

(1) $P(L') \leqslant P(L)$

(2) $h(L') \leqslant h(L) \leqslant P(L)$

-La remarque (1) vient du fait que $L=L'\sigma$ donc $R_L \subset R_{L'}$ donc $P(L) \leqslant P(L')$

-La remarque (2) peut se démontrer de la façon suivante:

$h(L) \leqslant P(L')$ puisque suivant les cas $h(L)=1$ ou $P(L')$. Donc d'aprés (1) on a $h(L) \leqslant P(L') \leqslant P(L)$
D'autre part si L est réductible alors L' aussi donc dans ce cas $h(L)=h(L')=1$
Dans l'autre cas $h(L)=P(L') \geqslant h(L')$

PROPOSITION III.3.1

La fonction f=g+h est une fonction croissante, i-e que:

$C' \leqslant C \Rightarrow f(C') \leqslant f(C)$

Soit C' l'ancêtre immédiat de C:

-Si C est obtenue à partir de C' par réduction sur le littéral L'_0 alors $C=(C'-\{L\})\sigma$ donc

$$h(C')=\sum_{\substack{L'\in C' \\ L'\neq L'_0 \\ L' \text{ B-litt}}} h(L') + h(L'_0)=\sum_{\substack{L'\neq L'_0 \\ L'\in C' \\ L' \text{ Blitt}}} h(L')+1 < \sum_{\substack{L \text{ B-litt} \\ L \in C}} h(L)+1=h(C)+1$$

or $g(C)=g(C')+1$ donc $g(C)+h(C)=g(C')+1+h(C) \geqslant g(C')+h(C')$

-Si C est obtenu par une expansion de C' sur une clause C_0 d'entrée, L'_0 étant le littéral résolu dans C', on a alors:

$P(L'_0) < l(C_0)$ puisque $C_0 \in R_{L'_0}$

$$\text{donc } h(C')=\sum_{\substack{L' \text{ B litt} \\ L'\neq L' \\ L'\in C'}} h(L')+h(L'_0) \leqslant \sum_{\substack{L' \text{ Blitt} \\ L' \in C' \\ L' \neq L'_0}} h(L')+l(C_0) \leqslant$$

$$\sum_{\substack{L \in C \\ L \text{ Blitt} \\ L \text{avec anc}}} h(L) + \sum_{\substack{L \in C \\ L \text{ sans anc}}} h(L)+1 \leqslant h(C)+1$$

et comme $g(C)=g(C')+1$ on a bien $f(C') \leqslant f(C)$

REMARQUE III.3.2
La fonction coût g est croissante.
Le mérite associé à f et h est δ -fini
La condition (1) du théorème I.6.2 est bien vérifiée.

-Le mérite est δ-fini puisque f est une fonction à valeurs entière croissante et que le nombre de clauses de départ est fini.
-La condition (1) du théorème I.6.2 peut se démontrer de la façon suivante:

soient C un ancêtre d'une clause C* vide: $C \leqslant C^*$.

on a alors $g(C)+h(C)=f(C) \leqslant f(C^*)$. Or $f(C^*)=g(C^*)+h(C^*)=g(C^*)$

Donc $g(C)+h(C) \leqslant g(C^*)$

On peut donc affirmer que toute stratégie de diagonale montante pour h est admissible, donc que l'on est assuré d'obtenir une solution de coût minimal s'il en existe.

Stratégie Σ

Le problème est de générer, à chaque étape, des clauses qui soient de meilleures heuristiques parmi tous les candidats.
Pour celà on associe, dans le codage des clauses, à chaque littéral d'une clause, son poids. En outre une fois un littéral sélecté dans une clause on garde la liste des clauses d'entrée susceptible de se résoudre avec C, le littéral sélecté étant celui qu'on résoud.
Cette liste et le poids de la clause sont dynamiques:

-tant qu'un littéral est réductible dans une clause et qu'il n'a pas encore été réduit de toutes les façons on lui attache un poids 1
-lorsqu'il a été réduit de toutes les façons on lui attache le poids tel qu'il est défini en III.3.1
-En ce qui concerne le littéral sélecté L d'une clause C, on considère la liste (dynamique) R_L des clauses d'entrée susceptibles de se résoudre avec C mais qui ne l'ont pas encore été.
Le poids de L est alors la longueur minimum de ces clauses;(+ si cette liste est vide, auquel cas la clause est supprimée).
Le poids d'une clause est alors la somme des poids de ses B-littéraux.

REMARQUE III.3.3

Si P(C) est le poids d'une lause C à un moment donné de la procédure utilisée, l'heuristique le plus petite des descendants immédiats de C est

$$P(C) - 1$$

La stratégie Σ employée par OEDIPE est alors une stratégie de diagonale montante.
Chaque clause est rangée dans un tableau SIGMA à double entrée, le premier indice donnant le niveau g, le second le poids P.
Chaque élément du tableau a pour valeur une liste de clause.

On parcours alors les diagonales g+P=Constante de façon croissante, chaque diagonale étant parcourue de bas en haut. (P croissant)
Supposons que (g_0,P_0) soit le mérite étudié:
SIGMA (g_0,P_0) a pour valeur la liste de toutes les clauses qui ont pour descendant immédiat des clauses de coût g_0+1 et l'heuristique P_0-1.

Pour chacune des clauses C de SIGMA (g_0,P_0) on génère alors leurs descendants de meilleur mérite, ceci récursivement tant qu'on reste sur la diagonale $P_0+P_0=g+P$.

Une fois ces descendants générés on modifie éventuellement la liste des clauses d'entrée susceptibles de se résoudre avec C et on calcule le nouveau poids de C en remplaçant C éventuellement dans le tableau SIGMA sur une diagonale supérieure.
On est alors assuré d'avoir une stratégie compatible avec associé à g et h.

Remarque:

Lorsqu'on génère les descendants immédiats d'une clause C on commence par générer les réductions s'il en existe. Sinon on choisit dans R_L (L littéral sélecté de C) les clauses de longueur minimales qu'on résoud avec C sur L.
Une fois ces générations effectuées on modifie R_L en conséquence et replace C s'il y a lieu.

Voici les schémas des procédures utilisées par la stratégie Σ :

<u>générer</u> (C): génère à partir d'une clause C, une réduction ou une expansion de meilleur mérite en appelant l'une des deux procédures <u>réduction</u> ou <u>expansion</u>.

réduction (L,C): génère à partir d'une clause C toutes les réductions possibles sur le littéral L ∈ C. Une fois les réductions faites C est replacé dans SIGMA.

expansion (C) génère les expansions de C avec les clauses les plus petites de R_L (L littéral sélecté de C) et replace C dans SIGMA.

tronquation (C,DIAG): Cette procédure est appelée pour chaque clause C nouvellement créée. Elle traite les parties égalitaires,les tests de minimalité (tautologie etc. et aprés tronquation éventuelle sélecte un littér Elle apelle éventuellement générer (C) si la diagonale ou l'on remplace C est DIAG.

Nous allons maintenant terminer ce chapitre en donnant un exemple de traitement de problème par OEDIPE.

IV EXEMPLE D'UN PROBLEME TRAITE PAR OEDIPE

Le listing que nous allons présenter est celui d'une ancienne version de OEDIPE dans laquelle le traitement de l'égalité formelle n'était pas inclus. Les annotations montrerons les modifications qu'apportent l'utilisation de littéraux égalitaires en appliquant le théorème II.4.1 Les littéraux égalitaires supprimés par contraction seront soulignés.

Nous donnerons deux exemples de problèmes portant sur la théorie des nombres entiers. Les axiomes utilisés ne sont bien sûr que partiels et ne permettent pas de définir toute l'axiomatique des nombres entiers Ce sont, à proprement parler, plutôt des lemmes que des axiomes.

Premier exemple

Il s'agit de démontrer à partir du théorème d'Euclide que si A,B et C sont trois nombres entiers alors si A est premier et si $A=B^2/C^2$ alors A divise B.

Les prédicats utilisés sont D(x,y),P(x),M(x,y,z) et le symbole fonctionnel S désigne le carré d'un nombre entier.

P(x) sera interprété par : x est premier
D(x,y) " " " : x divise y.
M(x,y,z) " " " : z=x.y.
S(x) " " " : x^2

Les axiomes généraux sont les suivants:

$x^2=x.x$ (définition de x^2) (clause n°4)

x.y=y.x (commutativité de la multiplication) (clause n°5)

Tout facteur d'un produit divise ce produit. (clause n°6)

Si un nombre premier divise un produit de facteurs il divise au moins l'un des facteurs (Théorème d'Euclide). (clauses n°7 et 8)

Le problème est alors de montrer que:

$\forall A \ \forall B \ \forall C$ (A premier et $A=B^2 / C^2$) $\Rightarrow$ (A divise B)

La négation de cette formule donne sous forme clausale l'ensemble des clauses 1,2 et 3.

L'ensemble de support sera constitué de la clause n°1.

Deuxième exemple

Le deuxième exemple consiste à montrer que tout nombre plus grand que 1 a un diviseur premier.
On utilise pour celà un raisonnement par réccurence et le théorème que l'on demande de démontrer est un pas de ce raisonnement par réccurence: on suppose la propriété vraie pour les nombres compris entre 1 et a, a donné quelconque, et on démontre qu'elle l'est aussi pour a.

Les prédicats utilisés sont P(x),D(x,y),L(x,y) et les symboles fonctionnels F et G. P et D sont interprétés comme précédemment, L(x,y) sera interprété par x < y.

Les axiomes généraux sont les suivants:

$\forall x$ x divise x. (clause n°2)

$\forall x \ \forall y \ \forall z$ si x divise y et si y divise z alors x divise z.(clause n°3)

$\forall x$ x non premier $\Rightarrow \exists y \ 1<y<x$ tel que y divise x (clause n°4,5,6 où G est une fonction de Skölem)

Le théorème à démontrer est le suivant:

$\forall a > 1 \quad (\forall x \cdot 1 < x < a \Rightarrow \exists y$ tel que y divise x et y est premier)

$\Rightarrow \exists z$ tel que z premier et z divise a

La négation de cette formule donne sous forme clausale les clauses 7,8, et 1 où A et F sont deux fonctions de Skölem.

L'ensemble de support est constitué de la clause 1

********** A X I O M E S **********

NIVEAU 0

1 -D(A,B)

****

NIVEAU -1

2 +P(A)

NIVEAU -1

3 +M(A,S(C),S(B))

NIVEAU -1

4 +M(*R,*R,S(*R))

NIVEAU -1

5 +M(*R,*S,*T)-M(*S,*R,*T)+E(*R,*S)+E(F(*S,*T),F(*R,S(*R)))

NIVEAU -1

6 +D(*R,*S)-M(*R,*T,*S)

NIVEAU -1

7 +D(*R,*S)+D(*R,*T)-D(*R,*U)-M(*T,*S,*U)-P(*R)
FACTEUR DE 7 +E(*S,*T)+E(*S,*U)+E(*T,*U)+E(F(*R,*U),F(A,B))

NIVEAU -1

8 -P(*R)-M(*S,*S,*T)-D(*R,*T)+D(*R,*S)

****
+E(*S,*T)+E(F(*R,*T),F(A,B))

exemple 1

$\forall A \in N \ \forall B \in N \ \forall C \in N \qquad A = B^2/C^2$ et A premier $\Rightarrow$ A divise B

```
****** DEMONSTRATION*******

EXPANSION DE    1  et   8
   NIVEAU    1
                  9 :    -P(A)-M(B,B,*R)-D(A,*R)   /-D(A,B)/ +E(B,*R)
                         +E(F(A,*R),F(A,B))
EXPANSION DE    9  et   2
   NIVEAU    2
                 10 :    -M(B,B,*R)-D(A,*R)   /-D(A,B)/ +E(B,*R)
                         +E(*R,B)
EXPANSION DE   10  et   4
   NIVEAU    3
                 11 :    -D(A,S(B))   /-D(A,B)/ +E(B,S(B)) +E(S(B),B)

EXPANSION DE   11  et   6
   NIVEAU    4
                 54 :    -M(A,*R,S(B))   /-D(A,S(B))/     /-D(A,B)/

EXPANSION DE   54  et   3
   NIVEAU    5
                 55 :    □

 NO D EXPANSIONS    36
NO DE REDUCTIONS    12
NO DE CLAUSES RETENUES    39
NO DE CLAUSES AUX LIT TROP LONGS     1
NO DE CLAUSES SUPPRIMEES PAR SUBS    0

018.62 SECONDS IN EXECUTION
```

Avec l'égalité formelle le nombre d'expansions est 30
le nombre de réductions est 11
le nombre de clauses retenues est 30

********** A X I O M E S **********

NIV 0 1 $-D(*R,A)-P(*R)$

****

NIV -1 2 $+D(*R,*R)$

NIV -1 3 $+D(*R,*S)-D(*T,*S)-D(*R,*T)+E(*R,*S)+E(*R,*T)+E(*T,*S)$

NIV -1 4 $+D(G(*R),*R)+P(*R)$

NIV -1 5 $+L(1,G(*R))+P(*R)$

NIV -1 6 $+L(G(*R),*R)+P(*R)$

NIV -1 7 $+L(1,A)$

NIV -1 8 $+P(F(*R))-L(*R,A)-L(1,*R)$

NIV -1 9 $+D(F(*R),*R)-L(*R,A)-L(1,*R)$

****

exemple 2

Tout nombre > 1 a un diviseur premier

```
****** DEMONSTRATION*******

EXP    1      2
NIV    1                    10  -P(A)

EXP   10      6
NIV    2                    11  +L(G(A),A)  //-P(A)//

EXP   11      8
NIV    3                    13  +P(F(G(A)))-L(1,G(A))  //+L(G(A),A)-P(A)//

EXP   13      1
NIV    4                    39  -D(F(G(A)),A)  //+P(F(G(A)))//
                              -L(1,G(A))  //+L(G(A),A)-P(A)//

EXP   39      3
NIV    5                    86  -D(*R,A)-D(F(G(A)),*R)  //-D(F(G(A)),A)+P(F(G(A))
                              -L(1,G(A))  //+L(G(A),A)-P(A)// +E(F(G(A)),A) +E(*R,A)
                               +E(F(G(A)),*R)

EXP   86      4
NIV    6                   135  +P(A)  //-D(G(A),A)//
                              -D(F(G(A)),G(A))  //-D(F(G(A)),A)+P(F(G(A)))//
                              -L(1,G(A))  //+L(G(A),A)-P(A)// +E(G(A),A) +E(F(G(A)),G(A))

RED  135
NIV    7                   136  -D(F(G(A)),G(A))  //-D(F(G(A)),A)+P(F(G(A)))//
                              -L(1,G(A))  //+L(G(A),A)-P(A)//

EXP  136      9
NIV    8                   183  -L(G(A),A)-L(1,G(A))  //-D(F(G(A)),G(A))-D(F(G(A)
                              -L(1,G(A))  //+L(G(A),A)-P(A)//                +P(F(G(A)))//

RED  183
NIV    9                   184  -L(1,G(A))  //-D(F(G(A)),G(A))-D(F(G(A)),A)+P(F(G
                              -L(1,G(A))  //+L(G(A),A)-P(A)//

RED  184
NIV   10                   185  -L(1,G(A))  //+L(G(A),A)-P(A)//

EXP  185      5
NIV   11                   186  +P(A)  //-L(1,G(A))+L(G(A),A)-P(A)//

RED  186
NIV   12                   187
*******C.Q.F.D. ******

NO D EXPANSIONS  159
NO DE REDUCTIONS   46
NO DE CLAUSES RETENUES  113
NO DE CLAUSES AUX LITT TROP LONGS   27
NO DE CLAUSES SUPPRIMEES PAR SUBS    0

059.86 SECONDS IN EXECUTION
```

Avec l'emploi de l'égalité formelle

nombre d'expansions 131
nombre de reductions 41
nombre de clauses retenues 97

CONCLUSION

On peut maintenant voir quels sont les avantages mais aussi les limites de l'emploi de l'égalité formelle. Un des principaux atouts est que son emploi : est très simple et son traitement relativement efficace. Par contre il nécessite un certain nombre de précautions à prendre quand à l'interprétation que l'on peut donner aux axiomes qui définissent ce prédicat.
Notre recherche future sera donc d'essayer de dégager les mécanismes inhérents au traitement de ces axiomes et de généraliser ce traitement pour des classes plus larges de prédicats.

Une autre voie, beaucoup plus ambitieuse, est d'essayer de généraliser le formalisme clausal en introduisant par exemple le concept de graphe de clauses. Ceci permettrait entre autre de considérer la logique du 1er ordre comme un langage de programmation permettant d'écrire des programmes non déterministes. Ce lien entre les langages algorithmiques et la logique du 1er ordre a déjà été la source de création de langages de programmation comme LISP par exemple mais n'a pas encore donné naissance à un langage qui regrouperait l'efficacité de la programmation et la puissance de la logique.

Nous pensons donc que c'est une des voies les plus interessantes à l'heure actuelle dans le domaine de la démonstration automatique.

Appendice 1

BIBLIOGRAPHIE

ALLEN,J., et LUCKHAM,D.,

1 An interactive theorem proving program. Machine Intelligence 5, Edinburgh University Press (1970) PP, 321-336.

ANDREWS,P.B.,

1 Resolution with merging. Journal of the A.C.M, vol 15 (1968) pp. 367-381.

BAUER,H.R., GRAHAM S.L., SATTERTHWAITE E., et BECKER S.,

1 Algol-W langage description CS 110. Computer Science Departement, Stanford University.

CHANG C.L.,

1 The Unit Proof and the Impur Proof in Theorem Proving. Journal of the A.C.M., vol 17 (1970) pp 698-707.

DARLINGTON J.L.,

1 Theorem-Proving and information retrieval. Machine Intelligence 4 Edinburgh University Press (1969) pp 173-181.

FIKES R.E. et NILSON N.J.,

1 Strips: a new approach to the application of theorem proving to problem solving. Proceedings Second International Joint Conference on Artificial Intelligence (1971). The British Computer Society. pp 608-620.

GREEN C.C.,

1 Application of theorem-proving to problem-solving. Proceedings First International Joint Conference on Artificial Intelligence. Washington, D.C. (1969) pp 219-239.

HART T.P.,NILSON N.J., et RAPHAEL B.,

1 A formal basis for the heuristic determination of minimum cost paths. I.E.E.E. Transactions on System Sciences and Cybernetics. (Juillet 1968).

HAYES,P.J., et KOWALSKI R.A.,

1 Semantic trees in automatic theorem-proving. Machine Intelligence 4. Edinburgh University Press (1969) pp 87-101.

Appendice 2

KNUTH D.E.,

1 The Art of Computer programming. Vol 1: Fundamental Algorithms Addison Wesley (1969).

KOWALSKI R.A.,

1 The case for using equality axioms in automatic demonstration. Proceedings IRIA, Symposium on Automatic Demonstration, (Versailles, 1968), Springer Verlag

2 Searche Strategies for theorem proving. Machine Intelligence 5 Edinburgh University Press (1970) pp 181-201.

3 Studies in the completeness and efficiency of theorem-proving by resolution. Ph. D. Thesis. University of Edinburgh (1970).

KOWALSKI R.A., et KUEHNER D.,

1 Linear function and selection function. Metamatics Unit. Meme 43 University of Edinburgh (1971)

KUEHNER D.,

1 Strategies for improving the efficiency of automatic theorem-proving Metamatics Unit. Memo 44. Ph. D. Thesis. University of Edinburgh (1971).

LOVELAND D.W.,

1 Mechanichal theorem-proving by model elimination. Journal of A.C.M vol 15 (1968) pp 236-251.

2 A linear format resolution. Proceeding IRIA Symposium on Automatic Demonstration. (Versailles, 1968) Springer Verlag,

LUCKHAM D.,

1 Refinements theorems in resolution theory. Proceeding IRIA Symposium on Automatic Demonstration (Versailles 1968) Springer Verlag

LUCKHAM D., et NILSON N.J.

1 Extracting Information from Resolution Proof Trees. Artificial Intelligence vol 12 Number 1 (1971) pp 27-54.

MELTZER B.,

1 Some notes on resolution strategies. Machine Intelligence 3, Edinburgh University Press (1968) pp 71-75.

2 Power amplification for theorem-provers. Machine Intelligence 5 Edinburgh University Press (1970) pp 165-179.

Appendice 3

PRAWITZ D.,

1 Advances and problems in mechanical proof procedures. Machine Intelligence 4, Edinburgh University Press (1969) pp 59-71.

ROBINSON G.A. et WOS L.,

1 Paramodulation and theorem-proving in first-order theories with equality. Machine Intelligence 4, Edinburgh University Press (1969) pp 135-150.

ROBINSIN J.A.,

1 A machine-oriented logic based on the resolution principle. Journal of the A.C.M, vol 12 51965) pp 23-41.

2 A review of automatic theorem-proving. Proceeding of Symposia in Applied Mathematics vol 19 (1967) pp 1-18.

3 The generalised resolution principle. Machine Intelligence 3 Edinburgh University Press (1968) pp 77-94.

SANDEWALL E.,

1 Formal methods in the design of question-answering systems. Departement of Computer Sciences Report NR 28 (1970) Uppsala University

SLAGLE J.R.,

1 Automatic Theorem-Proving with Buit-in Theories Including Equality, Partial Ordering and Sets. Heuristics Laboratory National Institute of Health Memo interne (1970)

SUTY D.,

1 Le langage Algol-W. Université de Grenoble Service de Mathématiques appliquées (1969-1970).

WOS L.,CARSON D.F.,et ROBINSON G.A.,

1 The Unit Préference Strategy in theorem proving. Proceeding of the AFIPS 1964 Fall Joint Computer Conference, vol 26,pp 616-621.

2 Efficiency and completeness of the set of support Strategy in theorem-proving. Journal of the A.C.M. vol 12 (1965) pp 536-541.

F32

71