



HAL
open science

Abstraction, partage de structure et retour arrière non aveugle dans la méthode de réduction matricielle en démonstration automatique de théorèmes

Ricardo Caferra

► To cite this version:

Ricardo Caferra. Abstraction, partage de structure et retour arrière non aveugle dans la méthode de réduction matricielle en démonstration automatique de théorèmes. Modélisation et simulation. Université Joseph-Fourier - Grenoble I, 1982. Français. NNT : . tel-00305297

HAL Id: tel-00305297

<https://theses.hal.science/tel-00305297>

Submitted on 23 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

présentée à

l'Université Scientifique et Médicale de Grenoble

pour obtenir le grade de
DOCTEUR DE 3ème CYCLE
«Informatique»

par

Ricardo CAFERRA



**ABSTRACTION, PARTAGE DE STRUCTURE ET RETOUR ARRIERE
NON AVEUGLE DANS LA METHODE DE REDUCTION MATRICIELLE
EN DEMONSTRATION AUTOMATIQUE DE THEOREMES.**



Thèse soutenue le 30 novembre 1982 devant la commission d'examen.

C. DELOBEL	}	Président
W. BIBEL		Examineurs
C. BOITET		
Ph. JORRAND		
H. SAYA		

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : M. TANCHE

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

(RANG A)

SAUF ENSEIGNANTS EN MEDECINE ET PHARMACIE

PROFESSEURS DE 1ère CLASSE

ARNAUD Paul	Chimie organique
ARVIEU Robert	Physique nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S.
AYANT Yves	Physique approfondie
BARBIER Marie-Jeanne	Electrochimie
BARBIER Jean-Claude	Physique expérimentale C.N.R.S. (labo de magnétisme)
BARJON Robert	Physique nucléaire I.S.N.
BARNOUD Fernand	Biosynthèse de la cellulose-Biologie
BARRA Jean-René	Statistiques - Mathématiques appliquées
BELORISKY Elie	Physique
BENZAKEN Claude (M.)	Mathématiques pures
BERNARD Alain	Mathématiques pures
BERTRANDIAS Françoise	Mathématiques pures
BERTRANDIAS Jean-Paul	Mathématiques pures
BILLET Jean	Géographie
BONNIER Jean-Marie	Chimie générale
BOUCHEZ Robert	Physique nucléaire I.S.N.
BRAVARD Yves	Géographie
CARLIER Georges	Biologie végétale
CAUQUIS Georges	Chimie organique
CHIBON Pierre	Biologie animale
COLIN DE VERDIERE Yves	Mathématiques pures
CRABBE Pierre (détaché)	C.E.R.M.O.
CYROT Michel	Physique du solide
DAUMAS Max	Géographie
DEBELMAS Jacques	Géologie générale
DEGRANGE Charles	Zoologie
DELOBEL Claude (M.)	M.I.A.G. Mathématiques appliquées
DEPORTES Charles	Chimie minérale
DESRE Pierre	Electrochimie
DOLIQUE Jean-Michel	Physique des plasmas
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques pures
GAGNAIRE Didier	Chimie physique

GASTINEL Noël	Analyse numérique - Mathématiques appliquées
GERBER Robert	Mathématiques pures
GERMAIN Jean-Pierre	Mécanique
GIRAUD Pierre	Géologie
IDELMAN Simon	Physiologie animale
JANIN Bernard	Géographie
JÖLY Jean-René	Mathématiques pures
JULLIEN Pierre	Mathématiques appliquées
KAHANE André (détaché DAFCO)	Physique
KAHANE Josette	Physique
KOSZUL Jean-Louis	Mathématiques pures
KRAKOWIAK Sacha	Mathématiques appliquées
KUPTA Yvon	Mathématiques pures
LACAZE Albert	Thermodynamique
LAJZEROWICZ Jeannine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre	Mathématiques appliquées
DE LEIRIS Joël	Biologie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences nucléaires I.S.N.
LOUP Jean	Géographie
MACHE Régis	Physiologie végétale
MAYNARD Roger	Physique du solide
MICHEL Robert	Minéralogie et pétrographie (géologie)
MOZIERES Philippe	Spectrométrie - Physique
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (biologie végétale)
PAYAN Jean-Jacques (détaché)	Mathématiques pures
PEBAY PEYROULA Jean-Claude	Physique
PERRIAUX Jacques	Géologie
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
RICHARD Lucien	Biologie végétale
RINAUDO Marguerite	Chimie CERMAV
SENGEL Philippe	Biologie animale
SERGERAERT Francis	Mathématiques pures
SOUTIF Michel	Physique
VAILLANT François	Zoologie
VALENTIN Jacques	Physique nucléaire I.S.N.
VAN CUTSEN Bernard	Mathématiques appliquées
VAUQUOIS Bernard	Mathématiques appliquées
VIALON Pierre	Géologie

PROFESSEURS DE 2^{ème} CLASSE

ADIBA Michel	Mathématiques pures
ARMAND Gilbert	Géographie

AURIAULT Jean-Louis	Mécanique
BEGUIN Claude (M.)	Chimie organique
BOEHLER Jean-Paul	Mécanique
BOITET Christian	Mathématiques appliquées
BORNAREL Jean	Physique
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CHARDON Michel	Géographie
COHENADDAD Jean-Pierre	Physique
DENEUVILLE Alain	Physique
DEPASSEL Roger	Mécanique des fluides
DOUCE Roland	Physiologie végétale
DUFRESNOY Alain	Mathématiques pures
GASPARD François	Physique
GAUTRON René	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude (M.)	Sciences nucléaires I.S.N.
GUITTON Jacques	Chimie
HACQUES Gérard	Mathématiques appliquées
HERBIN Jacky	Géographie
HICTER Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKOVE Claude (M.)	Géologie
LE BRETON Alain	Mathématiques appliquées
LONGEQUEUE Nicola	Sciences nucléaires I.S.N.
LUCAS Robert	Physiques
LUNA Domingo	Mathématiques pures
MASCLE Georges	Géologie
NEMOZ Alain	Thermodynamique (CNRS - CRTBT)
OUDET Bruno	Mathématiques appliquées
PELMONT Jean	Biochimie
PERRIN Claude (M.)	Sciences nucléaires I.S.N.
PFISTER Jean-Claude (détaché)	Physique du solide
PIBOULE Michel	Géologie
PIERRE Jean-Louis	Chimie organique
RAYNAUD Hervé	Mathématiques appliquées
ROBERT Gilles	Mathématiques pures
ROBERT Jean-Bernard	Chimie physique
ROSSI André	Physiologie végétale
SAKAROVITCH Michel	Mathématiques appliquées
SARROT REYNAUD Jean	Géologie
SAXOD Raymond	Biologie animale
SOUTIF Jeanne	Physique
SCHOOL Pierre-Claude	Mathématiques appliquées
STUTZ Pierre	Mécanique
SUBRA Robert	Chimie
VIDAL Michel	Chimie organique
VIVIAN Robert	Géographie

REMERCIEMENTS

Je tiens à remercier les membres du Jury de cette thèse :

. M. Claude Delobel, Professeur à l'USMG, qui a bien voulu être le Président du Jury.

. M. Wolfgang Bibel, Maître de Conférences à la Technische Universität München, Allemagne, qui a jugé ce travail en tant qu'autorité dans la matière et qui a accepté de faire partie du Jury. Je lui suis reconnaissant des remarques faites et de la qualité de son contact humain.

. M. Christian Boitet, Professeur à l'USMG, dont les remarques, du point de vue de la présentation du travail et des possibilités d'application du démonstrateur ont été très pertinentes,

. M. Philippe Jorrand, Directeur des Recherches au CNRS. Cette thèse a été réalisée dans le sein de son équipe. Je le remercie de ses observations intelligentes tout au long de mon travail et du climat qu'il a su créer dans son équipe. Qu'il soit assuré de ma gratitude et de mon estime.

. M. Henri Saya, ingénieur au CNRS. Il a été présent dès le début de ce travail et son expérience dans le domaine de la démonstration automatique (il a été l'un des pionniers à Grenoble) m'a été particulièrement utile.

Je tiens également à remercier Jorge Vidart, Professeur à l'Université Simón Bolívar de Caracas, Vénézuéla. Nous avons eu des conversations fructueuses pendant son année sabbatique passée à Grenoble. Son encouragement amical pendant cette période a été pour moi d'une importance décisive pour l'aboutissement de ce travail.

Ricardo J. Caferra

TABLE DES MATIERES

0.	INTRODUCTION	
1.	PRINCIPES DE LOGIQUE ET DE DEMONSTRATION AUTOMATIQUE	
1.1	Aperçu historique	1-1
1.2	Le Calcul propositionnel	1-2
1.3	Systèmes formels	1-6
1.4	Théorie de la démonstration	1-8
1.5	Propriétés des systèmes formels	1-9
1.6	Calcul des prédicats du premier ordre	1-12
1.7	Mécanisation de la démonstration des théorèmes	1-26
1.8	Pourquoi ce chapitre ?	1-40
2.	OUTILS DE BASE EN DEMONSTRATION AUTOMATIQUE ET QUELQUES METHODES DE PREUVE	
2.1	Outils de base; substitution et unification	2-1
2.2	Quelques méthodes de preuve et concepts associés	2-13
2.3	Complexité des preuves et des procédures de preuve	2-35
2.4	Quelques questions en rapport avec la complexité des preuves	2-37
3.	LA METHODE DE REDUCTION MATRICIELLE	
3.1	Sa place parmi les autres méthodes	3-1
3.2	Principes et description de la méthode	3-2
4.	ABSTRACTION, PLANS, VALIDATIONS, RETOUR ARRIERE ET PARTAGE DE STRUCTURE DANS LA REDUCTION MATRICIELLE	
4.1	Abstraction et connexions dans la réduction matricielle: les ap-graphes	4-2
4.2	Preuve = plan + validation	4-11
4.3	Ensemble de conditions de substitution minimalement incompatibles; utilisation pour le retour arrière	4-15
4.4	Une technique de partage de structure pour les matrices et pour les substitutions	4-22
4.5	Structure de données utilisée pour le retour arrière	4-28
5.	CONCLUSION	
6.	ANNEXE (Exemples)	
7.	BIBLIOGRAPHIE	

INTRODUCTION

Depuis 1965 (date de publication du principe de résolution de Robinson) le domaine de la démonstration automatique de théorèmes n'a pas cessé de croître en importance, en tant que discipline indépendante et comme outil dans d'autres domaines de l'Informatique (vérification et synthèse de programmes, systèmes question-réponse, divers domaines de l'Intelligence Artificielle, langages de programmation (PROLOG)).

Parmi les nombreuses raisons que l'on pourrait évoquer pour illustrer l'utilité des travaux dans ce domaine nous en retiendrons deux.

La première est en rapport avec l'utilité de la démonstration automatique dans la vérification de programmes; nous laissons parler Boyer et Moore [21]: "La démonstration automatique de théorèmes est cruciale pour l'automatisation du raisonnement sur les programmes... Peu de programmes aujourd'hui peuvent être certifiés corrects. La raison principale étant la carence d'un démonstrateur de théorèmes... "

La deuxième est plus en rapport avec le futur et correspond à l'utilisation de démonstrateurs automatiques de théorèmes pour essayer de résoudre des problèmes ouverts, notamment en mathématique, (voir, par exemple, la conférence par L.Wos au dernier Congrès sur la démonstration automatique à New York, en Juin 1982). Cette voie qui pouvait paraître il y a quelques années très peu prometteuse a donné déjà quelques résultats très encourageants.

Nous avons choisi comme sujet de travail une méthode de démonstration qui avait été quelque peu délaissée et qui n'avait fait l'objet d'aucune implantation (en principe elle exige plus de place que la résolution). L'intérêt de certains principes sur lesquels cette méthode est fondée a été confirmé depuis par leur utilisation dans d'autres méthodes (voir par exemple Andrews [3], Henschen [62] et notamment les développements très intéressants faits par Bibel [11], [12], [13], [14]).

La méthode, dite de "réduction matricielle", date de 1969 (Prawitz [11]). C'est une méthode de démonstration pour la logique du premier ordre (la logique du premier ordre, comme on le sait, permet de spécifier un très grand nombre de problèmes).

Pour répondre aux objections qui avaient détourné d'elle les efforts d'implantation, nous avons proposé d'y incorporer les notions de connexion et d'abstraction sur les connexions, ainsi que l'utilisation d'un algorithme d'unification adapté à notre but.

L'introduction de ces notions a deux conséquences: une à un niveau technique et l'autre à un niveau conceptuel. La première est une technique de partage de

structure et la deuxième correspond à la division en deux de la recherche d'une preuve (ou de la solution d'un problème en général): recherche d'un "plan" et "essai de validation" (vérification de l'adéquation au cas particulier traité) du plan proposé.

La recherche d'un plan correspond à la recherche d'une réfutation d'un ensemble de clauses (considérées comme des multiensembles de littéraux constants) et l'essai de validation du plan à la résolution d'un système d'équations sur les termes.

Bien que l'implantation faite soit séquentielle, cette division en deux se prête parfaitement (et a été conçue dans ce but) à une implantation en parallèle du démonstrateur.

Un avantage additionnel de la technique proposée est de permettre une utilisation des échecs de l'unification pour un retour arrière non aveugle (un algorithme pour ce faire est donné au Chapitre 4).

Nous avons validé les idées exposées dans la thèse par une implantation réalisée en Pascal sur un microordinateur LSI11.

La thèse comporte 4 chapitres, une annexe et une bibliographie.

Le contenu de chaque partie est organisé comme suit:

Le chapitre 1 est un résumé de connaissances en Logique nécessaires à une bonne compréhension de la suite. Ce chapitre pourrait aussi constituer une synthèse des connaissances en Logique les plus utilisées dans le domaine de la démonstration automatique.

Le chapitre 2 est plus explicitement dédié à certaines méthodes de démonstration automatique et à la présentation de quelques concepts fondamentaux comme ceux de substitution et d'unification.

Nous avons choisi de parler des méthodes dont les idées sous-jacentes se sont avérées les plus fertiles et qui ont influencé les travaux ultérieurs dans le domaine. Parmi ces méthodes ont été retenues celles qui avaient le plus de rapport avec la méthode de réduction matricielle.

Dans le chapitre 3 nous parlons de la méthode de réduction matricielle. Nous la présentons avec la notation introduite au chapitre 2 et d'une façon plus formelle que celle de la présentation originale. Une généralisation, qui tire parti du fait que la méthode isole les littéraux des clauses, est donnée pour le cas propositionnel.

Bien que la démonstration n'en soit pas donnée dans le texte, une extension est possible à la logique du 1^{er} ordre.

Pour le cas général, une modification de la méthode est aussi donnée. Elle permet la factorisation pour la résolution mais, à différence de celle-ci,

elle n'est pas nécessaire à la complétude de la réduction matricielle.

Dans le chapitre 4, les idées de "graphe abstrait" de "plan" et "validation du plan", d' "ensemble minimalement incompatible" (et l'algorithme pour le détecter) utilisé pour un retour arrière non aveugle, sont données. Quelques détails d'implantation sont aussi mentionnés.

Dans l'annexe quelques exemples du fonctionnement du démonstrateur sont donnés; le but est d'en montrer les caractéristiques principales et non d'établir des statistiques ou des comparaisons.

Pour la bibliographie consultée nous avons essayé "d'aller aux sources" ainsi que d'être exhaustifs dans la mesure du possible.

CHAPITRE 1

PRINCIPES DE LOGIQUE ET DE DEMONSTRATION AUTOMATIQUE

Dans ce chapitre, nous donnerons un aperçu de l'évolution historique des concepts étroitement liés à la démonstration automatique et nous ferons un résumé des outils logiques nécessaires à la compréhension de la suite.

1.1 APERÇU HISTORIQUE

La démonstration automatique de théorèmes par ordinateur représente l'aboutissement historique de préoccupations qui remontent loin dans la logique et la philosophie (en effet la logique ne s'est séparée de la philosophie qu'en 1847 avec la parution de l'ouvrage de G.Boole "The mathematical analysis of logic"). La logique a été longtemps (dans le monde occidental) soumise à l'influence d'Aristote, si longtemps que Kant disait, au XVIII^e siècle, qu'elle n'avait plus fait le moindre progrès depuis Aristote et qu'il la pensait même achevée. Cette influence est tout à fait justifiée, et il faut signaler que la syllogistique d'Aristote constitue déjà une étude systématique de la théorie de la démonstration puisqu'elle se fixe comme but de répertorier tous les procédés de raisonnement correct, ceux qui permettent de tirer des conclusions justes, et tous les procédés erronés, ceux qui ne conduisent pas à des conclusions certaines.

Dans l'évolution historique qui conduit à la démonstration automatique nous devons citer Leibniz [84] dont les idées constituent le programme du logicisme tel qu'il sera développé à partir de Frege.

Nous pouvons situer la naissance de la logique moderne, ou logique mathématique, au milieu du XIX^e siècle [15]. Elle s'est présentée successivement comme l'algèbre de la logique (avec Boole) et comme ce qu'on appellera, à partir de 1904, la logistique (avec Frege, Russell et aussi Peano).

Frege [51] propose de placer la logique comme fondement des mathématiques. Le système de Frege manipule des signes en tenant compte exclusivement de la *forme* des expressions manipulées. Il comporte des axiomes et des règles d'inférence (nous appellerons ces systèmes *axiomatico-déductifs* ou *formels*) qui permettront d'obtenir des théorèmes de la théorie. Aucune idée de la *signification* (sémantique) des signes manipulés n'est introduite.

Un des points prévus dans le programme de Leibniz était la construction d'un langage formalisé capable de servir de moyen d'expression pour la logique pure. La même idée a été reprise par Frege qui développa une idéographie qui était un langage pour la *pensée pure* construit à partir de symboles spécifiques et manipulé d'après certaines règles parfaitement définies. Ce symbolisme, qui était trop encombrant, n'a pas été utilisé par la suite.

radoxe de Russell qui montrait la possibilité de construire des antinomies dans le système de Frege. C'était une première crise des fondements et une alternative se présentait: soit abandonner le programme logiciste, soit le réaliser d'une autre façon. La théorie des types de Russell (1903) permit le deuxième choix.

Parallèlement au logicisme, l'école ensembliste arriva aussi à la découverte de paradoxes (Cantor, Zermelo) engendrant dans les deux écoles des problématiques tout à fait analogues. La solution au problème des paradoxes ensemblistes fut donnée par l'axiomatisation que Zermelo a faite de la théorie des ensembles (1908).

Le formalisme radical de Hilbert et sa conception de la *métamathématique* (méthodologie des sciences déductives - 1920) furent aussi une réponse à la crise de fondements. La méthode métamathématique étudie des théories mathématiques complètement formalisées. La démonstration des théorèmes dans une telle théorie constitue l'objet d'étude de la théorie métamathématique (Hilbert exigeait d'un système formel qu'il possède un algorithme de vérification des démonstrations). Le but de cette conception était de démontrer la non-contradiction d'une théorie mathématique quelconque. Cette possibilité allait échouer dans la deuxième crise des fondements provoquée par la découverte de Gödel (1931) d'un théorème dit de *limitation*. A partir de l'observation de l'analogie entre métamathématique et arithmétique il arriva à des résultats sur l'existence, dans certains systèmes formels, de formules qu'on ne peut ni démontrer ni réfuter dans ce système. Il démontra aussi (ce qui établissait des limites très précises à la formalisation) que si la métamathématique d'un système S admet une arithmétisation dans S , alors elle ne permet pas de démontrer la non-contradiction de S .

Ces théorèmes de limitation donnent une réponse définitive à des questions inévitables sur les systèmes formels (comme, par exemple, leur non contradiction). Après cet aperçu historique sur la recherche des fondements des mathématiques et des méthodes sûres pour trouver des démonstrations de théorèmes ou tester leur exactitude - recherches d'ailleurs intimement liées - nous concentrons notre intérêt sur la formalisation du langage de la logique (tel qu'il est employé actuellement) et sur les résultats qui ont permis une mécanisation de la démonstration des théorèmes dans le Calcul des Prédicats.

Nous présentons d'abord le calcul propositionnel et nous introduisons des définitions et des notions auxquelles nous reviendrons pour l'étude du Calcul des Prédicats, en ajoutant la notion de quantification.

1.2 LE CALCUL PROPOSITIONNEL

Traditionnellement une proposition est un énoncé déclaratif auquel on peut assigner une valeur vrai ou faux (que nous noterons V et F respectivement), mais -- les deux -- nous restons donc dans le cadre de la logique classique.

l'on *interprète* certaines suites finies de symboles (que nous appellerons par la suite *expressions* ou *formules*) d'un certain langage. Cette approche sémantique, appelée *théorie des modèles*, est celle que nous utiliserons pour le calcul propositionnel.

1.2.1 APPROCHE SEMANTIQUE

A. LE VOCABULAIRE DU LANGAGE

. Les connectifs propositionnels;

- 1) \neg négation (non)
- 2) \wedge conjonction (et)
- 3) \vee disjonction (ou)
- 4) \supset implication (si alors)
- 5) \equiv équivalence (si et seulement si)

. Les symboles propositionnels: Toutes les majuscules de l'alphabet ou des majuscules suivies d'indices numériques.

B. LE LANGAGE

Les *formules bien formées* (fbf) ou plus brièvement les *formules* sont les mots du langage définis récursivement comme suit:

- i) Tous les symboles propositionnels sont des fbf et on les appelle *atomes*.
- ii) Si P et Q sont des fbf alors
 $\neg P$ $P \wedge Q$ $P \vee Q$ $P \supset Q$ et $P \equiv Q$ sont aussi des fbf.
- iii) Seulement les expressions satisfaisant i) et ii) sont des fbf.

Pour éviter les ambiguïtés et minimiser l'emploi des parenthèses, on introduit une *hiérarchie des connectifs* qui coïncide par ordre décroissant avec l'énumération des connectifs logiques ci-dessus.

Nous appellerons les formules obtenues par application de la règle ii) *formules composées*.

1.2.1.1 TERMINOLOGIE

La valeur de vérité des formules composées en fonction des valeurs de vérité des atomes (ou de l'atome) qui la composent est donnée par des *fonctions de vérité*. Une fonction de vérité n-aire est une application:

$$\{ V, F \}^n \longrightarrow \{ V, F \} \quad (n \geq 1)$$

On définit pour chaque connectif une fonction de vérité par des équations [132]:

Négation:

$$H_{\neg}(V) = F \quad H_{\neg}(F) = V$$

Conjonction:

$$H_{\wedge}(V, V) = V$$

$$H_{\wedge}(V, F) = H_{\wedge}(F, V) = H_{\wedge}(F, F) = F$$

Disjonction:

$$H_V (F, F) = F$$

$$H_V (V, V) = H_V (V, F) = H_V (F, V) = V$$

Implication:

$$H_D (V, F) = F$$

$$H_D (F, V) = H_D (F, F) = H_D (V, V) = V$$

Equivalence:

$$H_E (F, F) = H_E (V, V) = V$$

$$H_E (F, V) = H_E (V, F) = F$$

Ces fonctions peuvent être schématisées par des *tables de vérité*:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \supset Q$	$P \equiv Q$
V	V	F	V	V	V	V
V	F	F	F	V	F	F
F	V	V	F	V	V	F
F	F	V	F	F	V	V

Bien que la découverte des tables de vérité soit attribuée à Peirce et à Wittgenstein on sait qu'elles étaient connues des grecs anciens [15].

On peut voir que les tables de vérité nous fournissent une procédure mécanique pour *décider* la valeur de vérité de n'importe quelle fbf.

Nous donnons ci-dessous quelques définitions pour le calcul propositionnel.

$\Delta^{(*)}$ Une *interprétation* I pour une fbf A est une fonction

$$I: P_A \rightarrow \{V, F\}$$

P_A : ensemble des symboles propositionnels de A. ∇

Pour une fbf comportant n atomes, nous aurons un maximum de 2^n interprétations possibles. Dans le calcul propositionnel le nombre d'interprétations possibles pour une formule est donc toujours fini.

NOTATION: Pour représenter une interprétation d'une fbf A ayant n composants on écrit:

$$I = \{v_1, \dots, v_n\}$$

où v_i ($1 \leq i \leq n$) représente un atome (ou sa négation) composant de A. Si l'atome figure nié cela signifie qu'on lui assigne (dans l'interprétation) la valeur F; s'il apparaît non nié on lui assigne la valeur V.

EXEMPLE:

Une interprétation possible de la formule

$$M \wedge \neg N \vee P \supset Q$$

est $I = \{M, \neg N, \neg P, \neg Q\}$.

(*) Pour alléger l'écriture le symbole " Δ " remplace dans tout le texte le mot "interprétation" et le symbole " ∇ " le mot "fin Définition".

Δ Une évaluation E d'une fbf A est une fonction (partielle).

$$E: A \times I \rightarrow \{V, F\}$$

A : ensemble de fbf.

I : ensemble d'interprétations des fbf de A .

La valeur de $E(A, I)$ est calculée par application des fonctions de vérité définies auparavant V .

Δ Une fbf A est dite *vraie* pour une interprétation I ssi $E(A, I) = V$, autrement A est dite *fausse* dans l'interprétation. V

Δ Une fbf est dite *valide* ssi elle est évaluée à V pour toutes ses interprétations. Ces formules sont aussi appelées *tautologiques* [160]. V

NOTATION:

$\models A$ sera une abréviation de A est valide.

Δ Deux fbf A et B sont dites *équivalentes* (ou *logiquement équivalentes* ou A est équivalent à B) ssi les valeurs de vérité de A et B sont les mêmes pour toute interprétation de A et B . V .

Cette définition peut être inadéquate si l'ensemble des symboles propositionnels de A et de B ne sont pas les mêmes. C'est pourquoi nous préférons la définition suivante:

Δ Deux fbf A et B sont dites *équivalentes* ssi $A \equiv B$ est une tautologie, autrement dit si $A \supset B$ et $B \supset A$ sont des tautologies. V

Δ Une fbf est dite *insatisfaisable* (ou *contradictoire*) ssi elle est fausse pour toutes ses interprétations. Elle est dite *satisfaisable* ssi elle n'est pas contradictoire (c'est à dire s'il existe au moins une interprétation pour laquelle elle est vraie). V

Si une fbf A est vraie (fausse) pour une interprétation I , on dit que I *satisfait* (*falsifie*) A .

Δ Quand une interprétation satisfait une formule, on dit que l'interprétation est un *modèle* de la formule. V

Δ Si S est un ensemble de fbf, M est un *modèle de S* ssi M est un modèle pour toutes les formules de S . V

Δ Un ensemble de fbf est dit *satisfaisable* ssi il existe un modèle de S . V

Une propriété bien connue à laquelle nous nous référerons souvent par la suite est la suivante:

THEOREME: Une formule A est une tautologie ssi $\neg A$ est une contradiction.

PREUVE: Par application directe des définitions de tautologie et contradiction. \blacksquare

Etant donné une fbf A , on est souvent amené à se poser la question: "quelles sont les conclusions que l'on peut tirer de A si l'on suppose A vraie?".

Cela justifie la définition suivante:

Δ On dit que la fbf B est une *conséquence logique* de A ou que B *découle logiquement* de A ou que B est une *conséquence valide* de A ssi toute interprétation qui satisfait A satisfait aussi B . \forall

Nous noterons ce fait:

$$A \models B$$

Il est important, en particulier dans les méthodes de démonstration de théorème: dont nous parlerons, d'avoir les formules sous une *forme normale*.

Δ Une formule est dite sous *forme normale conjonctive* ssi elle est de la forme

$$C_1 \wedge C_2 \wedge \dots \wedge C_n \quad (n \geq 1)$$

où chaque C_i ($1 \leq i \leq n$), appelé *conjoint*, est de la forme:

$$A_1 \vee A_2 \vee \dots \vee A_{m_i} \quad (1 \leq i \leq n)$$

où chaque A_j ($1 \leq j \leq m_i$) est un atome ou la négation d'un atome. \forall

Δ Une formule est dite sous *forme normale disjonctive* ssi elle est de la forme

$$D_1 \vee D_2 \vee \dots \vee D_n \quad (n \geq 1)$$

où chaque D_i ($1 \leq i \leq n$), appelé *disjoint*, est de la forme:

$$A_1 \wedge A_2 \wedge \dots \wedge A_{m_i} \quad (1 \leq i \leq n)$$

où chaque A_j ($1 \leq j \leq m_i$) est un atome ou la négation d'un atome. \forall

Pour toute formule, on peut trouver une formule équivalente sous forme normale conjonctive et une autre sous forme normale disjonctive (voir par exemple [27])

La présentation qui vient d'être faite du calcul propositionnel est suffisante pour répondre à toutes les questions intéressantes qui peuvent s'y poser. En particulier, elle permet l'usage d'une procédure mécanique pour tester la validité d'une formule quelconque.

Cependant, une approche *syntaxique* (ou *axiomatique*) est aussi possible.

Nous ne donnerons pas ici l'approche syntaxique du calcul propositionnel (voir approche syntaxique du calcul des prédicats: §1.6.2) mais, par contre, nous donnerons quelques définitions et résultats plus généraux de la *théorie de la démonstration* (qui utilise cette approche).

1.3 SYSTEMES FORMELS

Frege [51] fut le premier à donner un *système formel* pour le calcul propositionnel, composé d'axiomes et de règles formelles d'inférence. Dans cette approche, *n'interprète* plus les expressions du langage mais on les manipule (à l'aide de certaines règles) pour obtenir d'autres expressions.

Un *système axiomatico-déductif* ou *système formel* ou *théorie formelle* \mathcal{S} utili

un langage dont les mots sont les formules bien formées (fbf) de \mathcal{S} . Souvent on parle simplement de *théorie* (sous-entendu: formelle).

Du point de vue de la méthodologie des sciences déductives (métamathématique) toute discipline déductive est un système d'expressions ou de formules bien formées.

Naturellement, toutes les suites de symboles (d'un certain vocabulaire) ne sont pas acceptées comme des fbf, mais seulement celles ayant une certaine structure.

Soit A un ensemble d'expressions d'une discipline particulière. A l'aide de certaines opérations (les règles d'inférence) des formules nouvelles sont dérivées à partir de A . Ces formules sont les conséquences de l'ensemble A .

Δ L'ensemble de toutes les conséquences de l'ensemble A (noté $C_n(A)$) est l'intersection de tous les ensembles contenant A , fermés pour les règles d'inférences données. ∇

Tarski [150] donne 4 axiomes qui expriment des propriétés élémentaires des deux concepts primitifs: celui d'expression et celui de conséquence. Ces axiomes sont satisfaits dans toutes les disciplines formalisées connues.

S : ensemble de toutes les fbf.

A1. $\text{card}(S) \leq \aleph_0$ (comme d'habitude \aleph_0 est la puissance du dénombrable).

A2. si $A \in S$ alors $A \in C_n(A) \in S$.

A3. si $A \in S$ alors $C_n(C_n(A)) = C_n(A)$.

A4. si $A \in S$ alors $C_n(A) = \bigcup_{X \in P_f(A)} C_n(X)$.

où $P_f(A)$: ensemble des parties finies de A .

Δ Un ensemble de fbf qui contient toutes ses conséquences est appelé un système déductif ou un système fermé. ∇

Δ En ce qui nous concerne, nous dirons qu'un système formel \mathcal{S} est défini si les conditions 1 à 4 ci-dessous sont satisfaites (il est facile de voir que les systèmes formels ainsi définis satisfont A1 - A4).

1. Le vocabulaire de \mathcal{S} est un ensemble dénombrable de symboles.
2. Il existe un sous-ensemble de tous les mots qu'on peut engendrer à partir du vocabulaire de \mathcal{S} : les fbf de \mathcal{S} . En général on dispose d'une procédure mécanique pour déterminer si un mot est une fbf.
3. Parmi les fbf, on en distingue un sous-ensemble: les axiomes de \mathcal{S} . Quand on peut décider si une fbf est un axiome (ce qui est le cas le plus courant), on dit que \mathcal{S} est une théorie axiomatique.
4. Il existe un ensemble fini de relations dans l'ensemble des fbf de \mathcal{S} : R_1, \dots, R_n appelées règles d'inférence. On peut décider pour une fbf A si elle est en

relation R_i avec une autre (d'autres) fbf. A est dite une *conséquence directe* de cette (ces) fbf par R_i . \forall

Δ Une *démonstration* dans \mathcal{S} est une suite finie A_1, \dots, A_n de fbf telle que tout A_i ($1 \leq i \leq n$) est soit un axiome de \mathcal{S} , soit une fbf obtenue d'une fbf (plusieurs fbf) précédente (s) par application d'une règle d'inférence. \forall

Δ Un *théorème* A de \mathcal{S} est une fbf telle qu'il y a une démonstration dans \mathcal{S} dont A est la dernière fbf. \forall

NOTATION:

On note le fait que A est un théorème d'une théorie formelle \mathcal{S} par

$$\overline{\mathcal{S}} \vdash A$$

Généralement, on peut se permettre la notation plus simple:

$$\vdash A$$

(sous-entendu: dans une théorie formelle donnée).

Quand on démontre informellement des théorèmes, on introduit naturellement dans les démonstrations les hypothèses du théorème que l'on est en train de démontrer. La définition suivante formalise cette notion:

Δ Si A est une fbf et Γ un ensemble de fbf d'une théorie \mathcal{S} , on dit que A est une *conséquence de Γ (dans \mathcal{S})* ou que A est *déductible à partir de Γ* ssi il y a une suite finie A_1, \dots, A_n de fbf telle que $A = A_n$ et A_i ($1 \leq i \leq n$) est soit un axiome, soit une fbf dans Γ , soit une conséquence directe par une règle d'inférence des fbf précédant A_i dans la suite. Les éléments de Γ sont appelés les *hypothèses* ou *prémises* de la déduction. \forall

NOTATION:

Nous noterons " A est conséquence de Γ " par:

$$\Gamma \vdash A$$

1.4 THÉORIE DE LA DÉMONSTRATION : QUELQUES THÉOÈMES IMPORTANTS

THEOREME (de la déduction [64] (*)): Soit Γ un ensemble de fbf, A et B des fbf: si $\Gamma, A \vdash B$ alors $\Gamma \vdash A \supset B$. ($\Gamma, A \vdash B$ est une façon simplifiée d'écrire $\Gamma \cup \{A\} \vdash B$).

Le cas particulier $\Gamma = \emptyset$ donne:

si $A \vdash B$ alors $\vdash A \supset B$.

PREUVE: dans [100]. ■

(*) En fait, on devrait distinguer les théorèmes du langage des théorèmes sur le langage en appelant ces derniers d'une façon différente (ils sont appelés "propositions" dans [100]). Cependant, suivant l'usage le plus répandu, nous ne ferons pas cette distinction.

Ce théorème permet de réduire la notion de *déduction* à celle de *démonstration*. En effet nous aurons à *déduire* une *conséquence* B à partir des hypothèses A. Au lieu de faire ceci nous *démontrerons* le *théorème* $A \supset B$.

L'approche de la démonstration automatique que nous utiliserons étant sémantique, la version de ce théorème dont nous ferons usage est la suivante:

THEOREME: $A \models B$ ssi $\models A \supset B$.

PREUVE: dans [27], [73], [100].■

Comme nous l'avons déjà avancé les formules A auront pour nous la forme

$$H_1 \wedge H_2 \wedge \dots \wedge H_n \quad (n \geq 1)$$

donc

$$H_1 \wedge H_2 \wedge \dots \wedge H_n \models C \quad \text{ssi} \quad \models (H_1 \wedge H_2 \wedge \dots \wedge H_n) \supset C$$

où la formule

$(H_1 \wedge H_2 \wedge \dots \wedge H_n) \supset C$ est le *théorème*, H_i ($1 \leq i \leq n$) sont les *hypothèses* ou *prémises* et C est la *conclusion du théorème*.

Pour démontrer une formule, nous testerons l'insatisfaisabilité d'une autre formule:

THEOREME: $H_1 \wedge H_2 \wedge \dots \wedge H_n \models C$ ssi $H_1 \wedge H_2 \wedge \dots \wedge H_n \wedge \neg C$ est *insatisfaisable*.

PREUVE: dans [27], [73], [100].■

1.5 PROPRIETES DES SYSTEMES FORMELS

Les propriétés des systèmes formels introduites dans les définitions suivantes sont d'importance fondamentale dans l'étude de n'importe quel système axiomatico-déductif. Il faut remarquer qu'il existe de légères différences entre les définitions adoptées par les divers auteurs.

Δ Un système formel (ayant le symbole de négation) est dit *simplement consistant* ssi il n'y a pas de fbf A dans le système tel que l'on ait $\vdash A$ et $\vdash \neg A$. Il est dit *simplement inconsistent* dans le cas contraire, c'est à dire ssi il existe une fbf A dans le système tel qu'on a $\vdash A$ et $\vdash \neg A$.

Pour le calcul propositionnel et pour les systèmes qui nous intéressent, la définition suivante est équivalente:

Δ Un système formel est *simplement consistant* ssi il contient *au moins* une fbf non démontrable (autrement dit ssi *au plus* certaines fbf sont des théorèmes). Il est dit *simplement inconsistent* dans le cas contraire, c'est-à-dire ssi *toute* fbf est théorème.

On peut justifier intuitivement pour le calcul propositionnel cette équivalence en partant du fait qu'à partir d'une contradiction (A et $\neg A$) toute autre for-

mule est déductible: il suffit de voir dans la table de vérité (§1.2.1.1) que si l'antécédent d'une implication est F l'implication est évaluée à V; appliquer le théorème de la déduction (version sémantique) et la complétude du calcul propositionnel (voir la fin de ce paragraphe). Plus de détails peuvent être trouvés dans [72].

Il est intéressant de noter que cette deuxième définition est *plus générale* que la première parce qu'elle est applicable à des systèmes formels où le concept de négation n'existe pas.

Etant donné une propriété pour les fbf du système, alors:

Δ Un système est dit *consistant* relativement à la propriété ssi *seulement* les fbf qui ont cette propriété sont des théorèmes.∇

Si, par exemple, la propriété pour une fbf est "être valide " alors:

Δ Un système est dit *consistant* ssi seulement les fbf valides sont des théorèmes.∇

Une autre notion de consistance (la ω -consistance) sera présentée au §1.6.3.

Une autre question fondamentale est celle de la *complétude* d'un système formel. Les différents critères choisis comme réponse à la question informelle: " que veut-on pouvoir démontrer dans le système? " donneront lieu à des notions différentes de complétude.

Etant donné une propriété pour les fbf du système alors:

Δ Un système est *complet* relativement à la propriété si *toutes* les fbf qui ont cette propriété sont des théorèmes.∇

En choisissant une propriété "intéressante" une définition équivalente est la suivante:

Δ Un système est *complet* relativement à la propriété ssi *au moins* certaines fbf sont des théorèmes.∇

Si, par exemple, la propriété pour une fbf est "être valide", alors:

Δ Un système est *complet* ssi toute fbf valide est un théorème.∇

Cette notion est la plus utilisée pour les systèmes qui nous intéressent.

Nous donnons deux autres définitions de complétude citées souvent dans la bibliographie:

Δ Un système est *complet* ssi pour toute fbf A on a un (et seulement un) des cas suivants: $\vdash A$ ou $\vdash \neg A$.∇

Une propriété très forte (que les systèmes importants n'ont pas) est la *complétude de Post* (voir par exemple [40]):

Δ Un système est *complet* ssi l'incorporation comme axiome d'une fbf qui n'est pas démontrable rend le système inconsistant. ▽

Le calcul propositionnel est complet avec n'importe laquelle de ces trois définitions.

Une fois une théorie formelle définie, une question qui se pose naturellement est de savoir si l'on peut toujours répondre à la question: "la fbf A est-elle un théorème?" ou la question équivalente (dans les systèmes complets): "la fbf A est-elle valide?". Cette question est connue comme le *problème de la décision*.

Δ S'il existe un algorithme qui permet de décider si une fbf de la théorie est ou n'est pas un théorème, on dit que la théorie est *décidable* et l'algorithme est appelé une *procédure de décision*. ▽

Cette notion formelle de décidabilité est étroitement liée à la notion de fonction récursive ou calculable. En effet, le système est décidable ssi l'ensemble des théorèmes est *récursif* (c'est à dire si sa fonction caractéristique est calculable).

Un système qui n'est pas décidable est appelé *indécidable*.

Pour les systèmes indécidables une propriété moins forte peut être exigée:

Δ Si dans un système formel il existe une procédure qui permet de déterminer en un nombre fini de pas si une fbf A est un théorème, mais qui peut ne pas se terminer si A n'en est pas un (c'est à dire on ne sait pas si on n'a pas trouvé parce que effectivement A n'est pas un théorème ou parce qu'on n'a pas assez cherché) alors on dira que le système possède une *procédure de preuve* ou qu'il est *semidécidable*. ▽

Le calcul propositionnel est consistant, complet et décidable.

Bien que le calcul propositionnel permette de formaliser un nombre de problèmes et d'introduire les notions sur lesquelles s'appuieront des notions dans des langages plus complexes, il est insuffisant quand il ne s'agit pas d'exprimer des *propriétés*, mais des *relations* entre objets d'une façon générale (on peut voir des exemples très clairs dans [65]). En particulier, il est impuissant à exprimer les mathématiques en termes de logique (jusqu'au travaux de Frege et Peano, on aurait pu croire que la logique traditionnelle suffisait pour construire les mathématiques).

Nous étudions maintenant le calcul des Prédicats du premier ordre (l'adjectif *premier* est là pour le distinguer des logiques avec un pouvoir d'expression plus fort; logique de second ordre, logique faible de second ordre, logique de troisième ordre, et logiques d'ordre supérieur en général).

1.6 CALCUL DE PREDICATS DU PREMIER ORDRE

1.6.1 LANGAGE ET TERMINOLOGIE

A. LE VOCABULAIRE DU LANGAGE

- 1) () , (symboles de ponctuation)
- 2) \neg \wedge \vee \supset \equiv (connectifs logiques)
- 3) f_i^n avec $i \geq 1$ et $n \geq 0$ (symboles de fonction)

REMARQUE: l'ensemble de symboles de fonction peut être vide (voir § 1.6.2: Calcul de prédicats du premier ordre pur)

Conventions pratiques:

- i) Pour distinguer les différents symboles de fonction, nous écrirons f, g, h, \dots au lieu de $f_1^n, f_2^n, f_3^n \dots$
- ii) L'indice supérieur n sert à identifier l'arité de la fonction, nous l'omettrons; dans le cas où l'arité est zéro le symbole fonctionnel est appelé une *constante*, notée a, b, c, \dots
- 4) P_i^n avec $i \geq 1$ et $n \geq 0$ (symboles de prédicat)

Conventions pratiques:

- i) L'indice i sert à identifier le symbole de prédicat, nous ne l'utiliserons pas; pour distinguer les différents symboles de prédicat, nous écrirons: P, Q, R, \dots
- ii) L'indice supérieur n sert à identifier l'arité du prédicat, nous l'omettrons; dans le cas où l'arité est zéro, nous avons un *symbole propositionnel* (ou une *proposition*).
- 5) \forall appelé *quantificateur universel* et qui est lu dans ses occurrences: "pour tout" ou "quelque soit".
- 6) \exists appelé *quantificateur existentiel* et qui est lu dans ses occurrences: "il existe".
- 7) $u, v, \dots, z, u_1, u_2, \dots, v_1, v_2, \dots$ (symboles de variables)

B. LE LANGAGE

1) Les termes

- i) Les variables et les constantes sont des termes.
- ii) Si f_i^n est un symbole de fonction et t_1, \dots, t_n sont des termes, alors $f_i^n(t_1, \dots, t_n)$ est un terme.
- iii) Une suite de symboles du vocabulaire est un terme ssi elle obéit aux règles de formation 1i) et 1ii).

2) Les formules atomiques ou atomes

- i) Les propositions sont des formules atomiques.

alors $P^n(t_1, \dots, t_n)$ est une formule

Nous appellerons t_1, \dots, t_n les *arguments* du prédicat.

iii) Une expression est une formule atomique ssi elle obéit aux règles 2ii) et 2iii).

Dans l'utilisation de la logique on appelle souvent un prédicat unaire *propriété* et un prédicat n-aire; *relation n-aire*.

3) Les *formules bien formées* (fbf).

i) Une formule atomique est une fbf.

ii) Si A et B sont des fbf et x est une variable alors

$\neg A$ $A \supset B$ $(\forall x)A$ sont des fbf.

iii) Le quantificateur existentiel et les connectifs \wedge , \vee et \equiv sont introduits au moyen des définitions:

$(\exists x)A$	est défini	par	$\neg ((\forall x)(\neg A))$
$A \wedge B$	"	"	$\neg (A \supset \neg B)$
$A \vee B$	"	"	$(\neg A) \supset B$
$A \equiv B$	"	"	$(A \supset B) \wedge (B \supset A)$

La hiérarchie des connectifs et quantificateurs est la même que dans le calcul propositionnel (§ 1.2.1). " \exists " et " \forall " ont la même hiérarchie que " \neg ".

Il aurait été possible d'introduire d'abord $(\exists x)A$ et définir $(\forall x)A$ comme $\neg((\exists x)(\neg A))$.

Souvent, nous écrirons $\forall xA$ et $\exists xA$ à la place de $(\forall x)A$ et $(\exists x)A$.

iii) Une suite de symboles du vocabulaire est une fbf ssi elle obéit aux règles 3i) et 3ii).

REMARQUES:

1') L'ensemble des termes est le plus petit ensemble d'expressions contenant les variables et constantes et fermé pour la règle de formation 1ii).

3') L'ensemble des fbf est le plus petit ensemble d'expressions contenant les formules atomiques et fermé pour la règle de formation 3ii).

Pour faire le lien avec les langages de programmation, il est intéressant de noter que dans [120] les symboles de fonction sont appelés des *constructeurs*.

A Un terme qui ne contient pas de variables est appelé un *terme fermé*.

EXEMPLE:

$f(a, g(h(b, a), c))$

A La *profondeur* $h(t)$ d'un terme t est définie récursivement comme suit:

1. si t est une variable ou une constante alors

$$h(t) = 1$$

2. sinon ($t = f(t_1, \dots, t_n)$)

$$h(t) = 1 + \max \{h(t_1), \dots, h(t_n)\} . \nabla$$

La profondeur d'un terme est une mesure du niveau d'imbrication maximum.

Δ Dans les fbf $(\forall x)A$ et $(\exists x)A$, A est la *portée* des quantificateurs. ∇

Δ Une occurrence d'une variable x dans une fbf est dite *liée* ssi x apparaît après un quantificateur $(\forall x, \exists x)$ ou si elle est dans la portée d'un quantificateur; toute autre occurrence de x est dite *libre*. ∇

Δ Une variable est dite *liée* (*libre*) dans une fbf ssi elle a une occurrence liée (*libre*) dans la fbf. ∇

Une variable peut donc être libre et liée dans la même formule.

La notion de variable liée correspond à celle d'une variable telle que la fbf dans laquelle elle apparaît a un sens *indépendamment de cette variable*.

Δ Si A est une fbf et t un terme, t est dit *libre pour x dans A* ssi lorsqu'on remplace une occurrence libre de x dans A par t ; aucune variable de t ne devient liée. ∇

Dans l'approche sémantique du Calcul de Prédicats du 1^{er} ordre, nous pourrions facilement montrer que les formules contenant des variables libres ne sont pas "intéressantes".

Δ Une fbf ne contenant pas de variables libres est dite *fermée* (autrement dit toutes ses variables sont quantifiées).

Si elle n'est pas fermée elle est dite *ouverte*. ∇

Δ La *fermeture universelle*, ou plus simplement la *fermeture* d'une fbf A est définie comme la fbf fermée obtenue à partir de A en la préfixant avec des quantificateurs universels suivis des variables libres dans A . Si A est fermée sa fermeture universelle coïncide avec A . ∇

EXEMPLE:

La fermeture de la fbf : $P(y,u) \supset \neg (\exists y) Q(x,y,z)$ est

$$\forall x \forall y \forall u \forall z (P(y,u) \supset \neg (\exists y) Q(x,y,z))$$

Nous présentons maintenant les deux approches classiques du Calcul des Prédicats. La première, qui est historiquement la conséquence du logicisme, est une approche syntaxique et nous l'appellerons *méthode axiomatique*: elle correspond aux *systèmes de type hilbertien* [72]. On pourrait aussi parler de *système de type Frege* puisque ce fut Frege qui introduisit en 1879 [51] les systèmes formels et la théorie de la quantification. La deuxième approche est une approche sémantique.

1.6.2 APPROCHE SYNTAXIQUE

Δ Une théorie dans laquelle on introduit des quantificateurs (sur les variables individuelles) s'appelle une *théorie du premier ordre*. ▽

Δ Les axiomes d'une théorie du premier ordre se divisent en: *axiomes logiques* et *axiomes non-logiques* (ou *propres*). ▽

Les premiers sont les axiomes "inhérents" au langage et les deuxièmes sont les axiomes particuliers de la théorie dont on veut démontrer des théorèmes.

Δ Une théorie du premier ordre dans laquelle il n'y a pas d'axiomes propres est appelée un *Calcul des Prédicats du premier ordre*. ▽

Δ Un Calcul de Prédicats du 1^{er} ordre dans lequel il n'y a pas de symboles fonctionnels (dont les constantes sont un cas particulier) et tel que pour tout entier n il y a une infinité de symboles de prédicat d'arité n est appelé un *Calcul des Prédicats du premier ordre pur*. ▽

(Par exemple: $P(x,y)$ est une fbf de ce calcul mais $P(f(x),b)$ n'en est pas une).

Suivant [100], nous donnons les axiomes logiques et les règles d'inférence du Calcul des Prédicats du 1^{er} ordre (qui sont aussi ceux de toute théorie du premier ordre).

A. AXIOMES

Si A, B et C sont des fbf alors

- 1) $A \supset (B \supset A)$
- 2) $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$
- 3) $(\neg B \supset \neg A) \supset ((\neg B \supset A) \supset B)$
- 4) $(\forall x) A(x) \supset A(t)$ si $A(x)$ est une fbf et t un terme libre pour x dans $A(x)$.
- 5) $(\forall x) (A \supset B) \supset (A \supset (\forall x) B)$ si A ne contient pas d'occurrence libre de x .

On peut remplacer A, B et C par n'importe quelle fbf (ce sont des *métavariab*les représentant des fbf). Chaque fbf 1) à 5) représente donc une *infinité d'axiomes* c'est pourquoi on les appelle *schéma d'axiomes*.

B. REGLES D'INFERENCE

- 1) Modus ponens : de A et $A \supset B$ on conclut B .
- 2) Généralisation: de A on conclut $(\forall x) A$.

REMARQUE:

Au § 1.2 nous avons différé la présentation d'une théorie formelle axiomatique pour le calcul propositionnel, nous le faisons ici: comme connectifs propositionnels on garde seulement la négation et l'implication (les autres connectifs

peuvent être introduits à l'aide des définitions); les fbf sont celles utilisant ces deux connectifs; les axiomes sont 1) à 3) ci-dessus; la seule règle d'inférence est le modus ponens.

Les définitions de démonstration, théorème et déduction sont les mêmes que nous avons données pour le calcul propositionnel; en ajoutant une hypothèse (voir [100]) l'énoncé du théorème de la déduction est le même.

1.6.3 APPROCHE SEMANTIQUE

Une autre approche (historiquement postérieure) qui ne fait usage ni d'axiomes ni de règles d'inférence - abandonnant donc la notion de formule démontrable - utilise la notion de *validité*. Ici la notion de validité repose sur des considérations relatives à la théorie des ensembles. Cette approche est appelée *théorie des modèles*. L'objectif est de pouvoir assigner à des fbf un *sens* et des *valeurs de vérité*.

Pour donner un sens à une fbf on choisit une *structure* et on fait correspondre aux symboles de la fbf certains objets de la structure. Le sens des symboles dans la fbf est celui des objets correspondants de la structure.

Pour assigner des valeurs de vérité la façon la plus naturelle est de donner le moyen d'assigner des valeurs de vérité aux formules atomiques (lesquelles pourront avoir la valeur V ou la valeur F, mais pas les deux) et d'utiliser les tables de vérité comme dans le calcul propositionnel.

Nous devons aussi définir une procédure d'évaluation des formules quantifiées (un traitement exhaustif du concept de vérité dans les langages formels est donné dans [151]).

Pour la formalisation de ces idées nous nous sommes inspirés de [6], [60] [89] et [132].

Il est utile de considérer des structures, c'est à dire des ensembles dans lesquels on définit des opérations et des relations (par rapport à [60] nous donnons des définitions légèrement simplifiées qui sont suffisantes pour nos besoins).

Δ Une *structure de premier ordre* ou plus simplement une *structure* \mathbb{M} est un triplet $\langle D; F, R \rangle$ où:

D: ensemble *non vide*.

$F: \{ f_1^{i_1}, f_2^{i_2}, \dots, f_n^{i_n}, \dots \}$

$f_j^{i_j}$ ($j = 1, 2, \dots$) est une *opération d'arité* i_j (*finie*) dans D (application de D^{i_j} dans D).

$$R: \{ r_1^{k_1}, r_2^{k_2}, \dots, r_n^{k_n}, \dots \}$$

$r_j^{k_j}$ ($j=1,2,\dots$) est une relation d'arité k_j (finie) dans D.

On peut remarquer deux cas particuliers intéressants:

- a) s'il n'y a pas de relations -on note $\langle D; F, \emptyset \rangle$ ou $\langle D; F \rangle$ - \mathbb{M} est une algèbre.
- b) s'il n'y a pas d'opérations -on note $\langle D; \emptyset, R \rangle$ ou $\langle D; R \rangle$ - \mathbb{M} est un système relationnel \forall .

Une telle structure est parfois appelée *structure dans la théorie des ensembles* parce qu'on pourrait aussi considérer des structures où D n'est pas un ensemble [6].

Dans la définition de la sémantique des fbf du calcul de prédicats du premier ordre, D est appelé le *domaine* ou *univers* ou *univers du discours*.

Δ Une assignation ξ d'une fbf A à une structure $\mathbb{M} = \langle D; F, R \rangle$ est un triplet d'applications:

$$\xi = \langle \xi_1, \xi_2, \xi_3 \rangle$$

où

$$\begin{aligned} \xi_1 &: F_A \longrightarrow F \\ \xi_2 &: P_A \longrightarrow R \cup D^* \\ \xi_3 &: T_A \longrightarrow D \end{aligned}$$

F_A : ensemble de symboles de fonction de A.

P_A : ensemble de symboles de prédicat et de symboles propositionnels de A.

T_A : ensemble de termes de A.

$$D^* = D^0 \cup D^1 \cup \dots \cup D^n \cup \dots$$

REMARQUE: On aurait pu considérer seulement R comme le codomaine de ξ_2 ; on a ajouté D^* simplement pour pouvoir traiter d'une façon "élégante" la sémantique des symboles propositionnels, qui peuvent intervenir dans des fbf du calcul de prédicats (voir § 1.6.1).

ξ_1 : Comme il est habituel nous considérons $D^0 = \emptyset$ et une constante comme une fonction $\emptyset \rightarrow D$.

La valeur $\xi_1(a)$, si a est une constante, est parfois appelée un *élément distingué* de D [89] et aussi la *dénotation* de a [20].

ξ_2 :

i) si P est un symbole de prédicat

$$\xi_2(P) \in R$$

ii) sinon (P est un symbole propositionnel)

$$\xi_2(P) = \emptyset \quad \text{ou} \quad \xi_2(P) = D^*$$

ξ_3 :

i) si t est une variable

$$\xi_3(t) \in D$$

(on peut considérer ξ_3 comme assignant le sens $\xi_3(t)$ à la variable t).

ii) sinon ($t = f(t_1, \dots, t_n)$)

$$\xi_3(t) = \xi_1(f)(\xi_3(t_1), \dots, \xi_3(t_n))$$

Cette notation correspond, évidemment, à la valeur de la fonction $\xi_1(f)$ appliquée aux valeurs des arguments $\xi_3(t_1), \dots, \xi_3(t_n)$.

Δ Une structure \mathfrak{M} est *adéquate* pour une fbf A ssi il existe une assignation de A à \mathfrak{M} .

La phrase "une structure \mathfrak{M} de A " suppose que \mathfrak{M} est adéquate pour A .

Δ Une *interprétation* I pour une fbf A est un triplet

$$I = \langle A, \mathfrak{M}, \xi \rangle$$

où \mathfrak{M} est une structure et ξ une assignation de A à \mathfrak{M} .

Δ Soit S un ensemble de fbf. Si I est une interprétation pour chaque formule de S , alors I est une *interprétation de S* .

Si A est une fbf, \mathfrak{M} une structure adéquate pour A , ξ et ξ' deux assignations de A à \mathfrak{M} telles qu'elles coïncident sauf, peut être, dans la valeur assignée à la variable libre v : $\xi'_3(v) = a$ (c'est à dire on peut avoir dans ξ : $\xi_3(v) = a$ ou $\xi_3(v) \neq a$), alors nous noterons ξ' comme $\xi \left(\frac{a}{v} \right)$.

Δ Une *évaluation* d'une fbf pour une interprétation donnée pour cette fbf est une fonction (partielle)

$$E : L \times I \longrightarrow \{V, F\}$$

L : ensemble des fbf du calcul de prédicats.

I : ensembles des interprétations.

Evidemment $E(A, I)$ n'est définie que si I est une interprétation pour A .

Toutes les formes possibles de A sont 1 à 5 ci-dessous et la définition de

$E(A, I)$ est la suivante:

1) A : P (P : symbole propositionnel)

$$E(A, I) = V \quad \text{ssi} \quad \xi_2(P) = D^*$$

2) A : $P(t_1, \dots, t_n)$

$$E(A, I) = V \quad \text{ssi} \quad \langle \xi_3(t_1), \dots, \xi_3(t_n) \rangle \in \xi_2(P)$$

($\xi_2(P)$ représente l'*extension* du prédicat)

3) A : $\neg B$; $B \wedge C$; $B \vee C$; $B \supset C$; $B \equiv C$

où B et C dénotent des fbf

(voir définition fonctions de vérité § 1.2.1.1)

- a) $E(A, I) = V$ ssi $H_{\neg} (E(B, I)) = V$
 b) $E(A, I) = V$ ssi $H_{\wedge} (E(B, I), E(C, I)) = V$
 c) $E(A, I) = V$ ssi $H_{\vee} (E(B, I), E(C, I)) = V$
 d) $E(A, I) = V$ ssi $H_{\supset} (E(B, I), E(C, I)) = V$
 e) $E(A, I) = V$ ssi $H_{\equiv} (E(B, I), E(C, I)) = V$

4) A: $\exists x B$ où B est une fbf

$E(A, I) = V$ ssi il existe une interprétation $I' = \langle B, \mathbb{M}, \xi' \rangle$ avec $\xi'_x = \xi_x^{(a)}$ telle que $E(B, I') = V$

5) A: $\forall x B$ où B est une fbf

$E(A, I) = V$ ssi pour toute interprétation $I' = \langle B, \mathbb{M}, \xi' \rangle$ avec $\xi'_x = \xi_x^{(y)}$ pour tout $y \in D$ on a:
 $E(B, I') = V$

6) Si $E(A, I) \neq V$ alors $E(A, I) = F$

REMARQUE: La règle 4) d'évaluation correspond à la généralisation de l'évaluation d'une *disjonction* de fbf dans le calcul propositionnel. Pour illustrer ceci considérons la fbf $\exists x P(x)$ et une interprétation I avec domaine fini D. On peut assigner à x successivement tous les éléments de D (disons x_1, x_2, \dots, x_n). Pour chacune de ces assignations P(x) peut être évaluée à V ou à F (avec la règle 2). Si l'on forme la disjonction $P(x_1) \vee P(x_2) \vee \dots \vee P(x_n)$ alors elle sera évaluée à V, ssi il y a au moins un disjunct qui est évalué à V. La quantification existentielle permet d'étendre cette notion à des interprétations avec domaine infini.

Un raisonnement analogue montre que la règle 5) correspond à une généralisation de l'évaluation d'une *conjonction* dans le calcul propositionnel.

Pour une interprétation donnée, une fbf fermée représente une proposition qui est vraie ou fausse, tandis qu'une fbf ouverte représente une *relation* dans le domaine et peut avoir la valeur V pour certaines valeurs assignées aux variables libres dans le domaine et la valeur F pour d'autres valeurs assignées aux mêmes variables dans le domaine. C'est dans ce sens que nous avons dit au § 1.6.1 que les formules ouvertes n'étaient pas "intéressantes".

Il faut noter aussi qu'en ce qui concerne interprétation et évaluation il n'y a pas jusqu'ici de différence entre constante et occurrence libre d'une variable.

D'après la sémantique décrite ci-dessus, on peut dire que les *termes* dénotent des *individus* (de l'univers du discours) et les *fbf* dénotent des *valeurs de vérité*.

La question de la validité d'une fbf se ramène à celle de la validité d'une fbf fermée. Si l'on veut pouvoir assigner une valeur de vérité à une fbf contenant des variables libres il faut introduire la définition suivante [89].

Δ Une fbf est évaluée à V dans une interprétation ssi sa fermeture est évaluée à $V . \nabla$

EXEMPLE:

$(\forall x) P(x) \supset P(y)$ est valide ssi $(\forall y) ((\forall x) P(x) \supset P(y))$ est valide.

REMARQUE: Si l'on remplace dans la définition d'interprétation "A est une fbf" par "A est un langage" (un système formel) nous avons la définition d'interprétation pour un langage. Mais il est évident qu'il nous faut une notion qui nous permette de considérer une infinité de fbf:

Δ Un *modèle d'une théorie formelle* est un modèle des axiomes. ∇

Intuitivement, on voit que le modèle satisfera toutes les fbf déduites dans le système si la (les) règle (s) d'inférence "transmet" ("transmettent") la satisfaisabilité. Au Chapitre 2 nous dirons d'une telle règle d'inférence qu'elle est *correcte*.

Les définitions d'équivalence, satisfaisabilité, validité et insatisfaisabilité sont les mêmes que pour le calcul propositionnel.

Puisqu'il n'y a pas de limitation sur la cardinalité des domaines, on a une infinité d'interprétations possibles pour une fbf. Pour tester la validité d'une fbf, nous ne pouvons donc plus utiliser la procédure de décision fournie par les tables de vérité. Si on regarde les règles d'évaluation d'une fbf on conclut facilement que la seule chose qui importe est la *cardinalité* du domaine et non pas les *noms* que l'on donne à ses éléments.

Pour les univers finis la définition suivante est parfois utilisé:

Δ Une fbf est dite *n-valide* ssi elle est évaluée à V dans toutes les interprétations avec des univers de cardinalité $n . \nabla$

Nous examinons quelques exemples d'évaluation des fbf du calcul de prédicats du 1^{er} ordre.

EXEMPLE 1: Soit la fbf:

$$(\forall x) (P(x) \supset Q (f(x), a))$$

nous allons l'évaluer pour une certaine interprétation définie comme suit:

STRUCTURE **m**:

1) Domaine:

$$D = \{ 1, 2 \}$$

2) Ensemble de fonctions:

(bien que ce ne soit pas nécessaire F peut être ici l'ensemble de toutes les fonctions de $D \rightarrow D$ puisque le domaine est de faible cardinalité).

a) $f_1 : \{ \langle 1,1 \rangle , \langle 2,1 \rangle \}$

b) $f_2 : \{ \langle 1,1 \rangle , \langle 2,2 \rangle \}$

c) $f_3 : \{ \langle 1,2 \rangle , \langle 2,1 \rangle \}$

d) $f_4 : \{ \langle 1,2 \rangle , \langle 2,2 \rangle \}$

3) Ensemble de relations:

$R_1 : \{ \langle 1 \rangle \}$

$R_2 : \{ \langle 1,1 \rangle , \langle 1,2 \rangle \}$

$R_3 : \{ \langle 1,1 \rangle , \langle 1,2 \rangle , \langle 2,1 \rangle \}$

ASSIGNATION ξ :

$\xi_1 (a) = 2$

$\xi_1 (f) = f_4$

$\xi_2 (P) = R_1$

$\xi_2 (Q) = R_3$

$\xi_3 (x) = 2$

$\xi_3 (f(x)) = f_4(2) = 2$

EVALUATION (avec interprétation $I = \langle A, \text{III}, \xi \rangle$):

$P(x)$ évalué à F : $\langle 2 \rangle \notin R_1$

Il n'est pas nécessaire alors d'évaluer le conséquent de l'implication.

$P(x) \supset Q(f(x), a)$ est donc évaluée à V dans l'interprétation I .

La seule autre assignation possible qui diffère au plus de la première dans la valeur assignée à x est ξ' avec:

$\xi'_3 (x) = 1$

EVALUATION (avec interprétation $I' = \langle A, \text{III}, \xi' \rangle$)

$P(x)$ évalué à V : $\langle 1 \rangle \in R_1$

$Q(f(x), a)$ évalué à F : $\langle 2,2 \rangle \notin R_3$

$P(x) \supset Q(f(x), a)$ est donc évaluée à F dans l'interprétation I' .

Finalement la règle 5) d'évaluation nous permet d'évaluer la formule du départ:

$(\forall x) (P(x) \supset Q(f(x), a))$ est évaluée à F dans l'interprétation I .

EXEMPLE 2: Soit la fbf:

$(\exists x) P(x) \supset (\forall x) P(x)$

Elle est 1-valide. En effet il est facile de vérifier que pour les univers de cardinalité 1 l'évaluation de $\exists x P(x)$ et de $\forall x P(x)$ coïncident, donc l'antécédent et le conséquent de l'implication sont évalués à la même valeur de vérité, donc l'implication est toujours V .

Il est clair qu'elle n'est pas valide.

EXEMPLE 3: Soit la fbf :

$$(\forall x)((\neg P(x,x) \wedge (\forall y) (\forall z) ((P(x,y) \wedge P(y,z)) \supset P(x,z)) \wedge (\exists w) P(x,w))$$

Elle n'est pas satisfaisable dans les univers de cardinalité finie, mais si l'on interprète $P(x,y)$ comme " $x < y$ " alors elle est satisfaisable dans l'univers des entiers [115] (elle nie la réflexivité et affirme la transitivité de " $<$ " et l'existence d'un entier plus grand qu'un entier donné).

Un théorème très important dans la voie de la mécanisation de l'approche sémantique est le suivant:

THEOREME (de Löwenheim ou de Skolem-Löwenheim [90]):

Si une fbf du calcul des prédicats est satisfaisable, alors elle est satisfaisable dans une interprétation avec domaine dénombrable.

PREUVE: Dans [20], [100].■

Skolem a postérieurement donné une version plus puissante du même théorème dans laquelle il est dit que si une fbf est satisfaisable dans une interprétation I avec domaine D alors elle est satisfaisable dans une interprétation I' avec domaine dénombrable D' . D' est un sous-ensemble de D et I' peut être extraite de I (des précisions peuvent être trouvées dans [20]).

Des résultats plus récents permettent de remplacer dans l'énoncé du théorème "domaine dénombrable" par des sous-ensembles très particuliers de l'ensemble des nombres naturels.

Nous faisons remarquer ici que la méthode employée par Löwenheim pour démontrer son théorème a été à l'origine de la théorie de Herbrand dont nous parlerons au § 1.7.2.

Comme conclusion des paragraphes 1.6.2 et 1.6.3 nous retenons les questions fondamentales qui se posent dans les deux approches du calcul de prédicats:

" Soit A une fbf; peut-on savoir si":

$$\begin{array}{l} \text{approche syntaxique} \\ \text{(axiomatique)} \\ \\ \text{approche sémantique} \end{array} \left\{ \begin{array}{l} A \text{ est démontrable?} \\ A \text{ n'est pas démontrable?} \\ \\ A \text{ est valide?} \\ A \text{ n'est pas valide?} \end{array} \right.$$

Le rapport entre ces propriétés est établi par le théorème suivant:

THEOREME (de Complétude de Gödel[57]): *Dans tout calcul de prédicats du premier ordre une fbf est un théorème ssi elle est valide.*

PREUVE: dans [20] , [100].■

REMARQUE: Dans [20] on donne le nom de "théorème de complétude " à la partie "si" du théorème et "théorème de correction" (nous avons traduit "soundness" par "correction") à la converse. L'intérêt de cette séparation (exclusivement dans un souci d'homogénéité de l'exposé) sera évident au Chapitre 2 quand nous introduirons les définitions de correction et complétude pour les méthodes de preuve.

Comme l'indique le théorème suivant le calcul de prédicats du premier ordre est "libre de contradiction":

THEOREME : *Le calcul de prédicats du premier ordre est (simplement) consistant.*

PREUVE: Dans [100].■

Avant de parler des problèmes de décidabilité nous donnons une définition de consistance que nous n'avons pas présentée au § 1.5.

Δ Un système \mathcal{S} dans lequel on peut représenter les nombres naturels est ω -consistant [56] si dans \mathcal{S} il n'y a pas de variable x et de formule $A(x)$ telle que l'on ait:

$$a) \vdash A(0), \vdash A(1), \vdash A(2), \dots$$

et

$$b) \vdash \neg \forall x A(x)$$

sinon \mathcal{S} est dit ω -inconsistant. ∇

La ω -consistance implique la consistance simple ([73], [100]).

1.6.4 PROBLEMES DE DECIDABILITE DANS LE CALCUL DES PREDICATS DU PREMIER ORDRE

1.6.4.1 INDECIDABILITE DU CALCUL DES PREDICATS

Une autre question fondamentale (en particulier quand on veut mécaniser la démonstration de théorèmes) est: " existe-t-il une procédure de décision pour le calcul de prédicats du premier ordre?".

Une réponse négative a été donnée par Church en 1936.

THEOREME (de Church [30]): *Le calcul de prédicats du premier ordre est indécidable (on dit aussi: récursivement indécidable).*

PREUVE: Dans [100].■

En utilisant une codification, par exemple une gödelisation (application bijective de l'ensemble des fbf dans les nombres naturels), on peut considérer le codomaine de cette codification à la place de l'ensemble de fbf. Le problème de la décision est alors équivalent à se demander si la fonction caractéristique de cet ensemble d'entiers est récursive.

Un énoncé équivalent pour le théorème de Church serait donc:

L'ensemble des théorèmes du calcul de prédicats du premier ordre est récursivement énumérable mais non récursif

Il existe pourtant une *procédure de preuve* ou de *sémidécision* (voir §1.5), l'énumération en est une, bien qu'elle ne soit pas praticable. Une façon évidente d'énumérer les théorèmes est : en partant des axiomes appliquer toutes les règles d'inférence de toutes les façons possibles à chaque théorème obtenu (une façon plus formelle de faire ceci utiliserait une gödelisation des fbf et des démonstrations).

Le calcul de prédicats du premier ordre est donc (simplement) consistant, complet et sémidécidable.

Etant donnée la réponse négative pour le problème de la décision on peut l'énoncer d'une façon moins forte comme suit: " Peut-on décider pour une fbf quelconque si elle est n-valide pour tout n fini?".

Trakhtenbrot montra en 1950 (cité dans [46]) qu'on ne peut pas non plus trouver une solution pour cette deuxième forme du problème de la décision.

1.6.4.2 DECIDABILITE POUR CERTAINES CLASSES DE FORMULES

L' inexistence d'une procédure de décision dans le cas général n'empêche pas l'existence d'une procédure de décision pour certaines classes de formules (procédure de décision dans le sens le plus fort).

Nous donnons d'abord quelques définitions:

Δ Une fbf A est dite sous *forme normale prénexe* ssi A est de la forme:

$$(Q_1 x_1) \dots (Q_n x_n) M$$

où Q_i ($1 \leq i \leq n$) est soit \forall , soit \exists et M est une fbf qui ne contient aucun quantificateur.

$(Q_1 x_1) \dots (Q_n x_n)$ est appelé le *préfixe* et M la *matrice* de A. ▽

Une propriété bien connue est la suivante:

Il existe une procédure effective pour transformer toute fbf dans une fbf sous forme normale prénexe équivalente (voir par exemple [100]).

Une telle procédure sera implicitement donnée dans la transformation d'une formule sous forme clausale (§ 1.7.1.2).

Dans ce qui suit nous parlerons de préfixe et de matrice d'une fbf, en omettant de préciser à chaque fois que ladite fbf est sous forme normale prénexe.

Δ Une fbf est dite sous *forme normale de Skolem* ssi le préfixe est de la forme: $\exists x_1 \dots \exists x_n \forall y_1 \dots \forall y_m$ (c'est à dire ssi tous les quantificateurs existentiels précèdent tous les quantificateurs universels).

Dans tout calcul de prédicats de 1^{er} ordre pur il existe une procédure effective pour transformer toute fbf A en une autre en forme normale de Skolem B, telle que A est valide ssi B est valide (voir par exemple [100]).

Par abus de langage nous appellerons les atomes "atomes positifs" et les atomes précédés du signe " \neg " atomes négatifs. Le mot "atome" identifiera les atomes positifs et les atomes négatifs.

Δ La matrice d'une fbf est une *formule de Horn* ssi elle est en forme normale conjonctive et chaque conjoint contient au plus un atome positif [66], [132]. ∇

REMARQUE: Les formules de ce type (avec élimination des quantificateurs existentiels) sont celles admises dans le langage PROLOG ([75], [125]).

Δ La matrice d'une fbf est *Krom* ssi elle est en forme normale conjonctive et chaque conjoint est une disjonction de deux atomes. ∇

Une étude exhaustive du problème de la décision est faite dans [1] et [46]; nous nous bornerons ici à énumérer quelques classes décidables.

Dans un traitement systématique du problème de la décision, les formules sont groupées en fonction de leurs structure logique. On peut classer les fbf pour lesquelles le problème de la décision a une réponse positive d'après [1]:

- i) Le préfixe.
- ii) La structure de la matrice.
- iii) Le nombre d'occurrences et l'arité des symboles de prédicat.

Dans la liste suivante ne sont considérées que des fbf appartenant au Calcul de Prédicats du 1^{er} ordre pur (mais la plupart d'entre elles peuvent être transformées en des formules contenant des symboles fonctionnels par skolemisation - voir § 1.7.1.1).

Il existe une *procédure de décision* pour les classes de formules suivantes:

- 1) Avec préfixe de la forme: $\exists x_1 \dots \exists x_m \forall y_1 \dots \forall y_n$ (classe de Bernays - Schönfinkel).
- 2) Avec préfixe de la forme: $\forall y_1 \forall y_2 \exists x_1 \dots \exists x_n$ (classe de Gödel - Kalmar - Schütte).
- 3) La matrice est une conjonction d'atomes (classe de Herbrand).
- 4) Formules en forme normale de Skolem dont la matrice est Krom (classe de Maslov).
- 5) La matrice est en forme normale conjonctive et contient au plus deux occurrences de chaque symbole de prédicat.
- 6) Le préfixe est de la forme: $\forall y_1 \exists x \forall y_2$ et la matrice est une formule de Horn (classe de Horn).
- 7) La matrice est une formule de Horn et le nombre d'occurrences de chaque atome négatif est de 1 au plus.
- 8) La matrice est une formule de Horn et le nombre d'occurrences de chaque atome positif est de 1 au plus.

9) La matrice contient seulement des atomes à 1 argument (classe monadique).

Sur le problème de la décision on peut consulter également [29], [31].

1.7 MECANISATION DE LA DEMONSTRATION DES THÉORÈMES

La plupart des procédures de démonstration automatique, et en particulier celle que nous avons choisie d'implanter, s'appliquent à des formules sous une forme normale ou standard obtenue (par simple omission des quantificateurs universels) à partir de leur *forme normale conjonctive de Skolem* [89], ou *forme normale fonctionnelle* [115] (cette forme ne doit pas être confondue avec la forme normale de Skolem du § 1.6.4.2):

$$(\forall x_1) \dots (\forall x_n) M$$

où x_i ($1 \leq i \leq n$) sont *toutes* les variables dans M (la fbf en question est donc *fermée*). M est en forme normale conjonctive.

Cette forme normale est appelée, dans la terminologie de la démonstration automatique *forme clausale* et il existe un algorithme pour le passage d'une fbf quelconque à une autre (non équivalente, mais qui préserve la validité) sous cette forme.

1.7.1 TRANSFORMATION SOUS FORME CLAUSALE

Nous donnerons les règles de transformation sur une formule que nous prendrons comme exemple. Cette formule du Calcul de Prédicats du premier ordre exprime un théorème de la théorie des groupes. Puis, en utilisant un algorithme de transformation nous obtiendrons sa forme normale.

Considérons le théorème de la théorie des groupes: "*tout élément d'un groupe a un inverse à droite*". Nous écrivons une des fbf qui représente cet énoncé en écrivant d'une part les axiomes propres (ou prémisses) et d'autre part la conclusion du théorème (qui devra être une conséquence logique des axiomes -voir § 1.3, § 1.6.2, § 1.6.3 -).

En général, dans la pratique, pour formuler un théorème à démontrer la démarche générale est la suivante: on considère la structure mathématique ou une structure qui est censée représenter le "monde réel", dont on veut exprimer certaines propriétés, et l'on suit la démarche *inverse* à celle d'assigner une sémantique à une fbf, ce qui correspond à la description d'un système *abstrait* [72]. La structure mathématique du départ (ou celle représentant le "monde réel ") constituent une *représentation* [72] ou un *modèle* du système abstrait (ici modèle a sa signification logique, et ne doit pas être confondue avec la notion de modèle utilisée par exemple, en simulation).

Le fait de montrer que la formule est un théorème revient à montrer que toutes les structures qui sont, via une interprétation, un modèle des axiomes, satisfont via la même interprétation la propriété énoncée dans la conclusion du théorème.

Pour notre exemple nous écrivons:

$P(x,y,z)$ dénote $x \circ y = z$ où " \circ " est l'opération binaire dans le groupe

A1) $\forall x \forall y \exists z P(x,y,z)$ (l'opération est interne)

A2) $\forall x \forall y \forall z \forall u \forall v \forall w ((P(x,y,u) \wedge P(y,z,v)) \supset (P(x,v,w) \equiv P(u,z,w)))$
(associativité)

A3) $\exists x (\forall y P(x,y,y) \wedge \forall y \exists z P(z,y,x))$ (existence de l'identité à gauche et de l'inverse à gauche)

T) $\exists x (\forall y P(x,y,y) \wedge \forall y \exists z P(y,z,x))$ (conclusion du théorème)

Pour démontrer le théorème $A1 \wedge A2 \wedge A3 \supset T$ (noter qu'il s'agit d'une formule fermée) nous montrerons l'insatisfaisabilité de la formule suivante:

$A1 \wedge A2 \wedge A3 \wedge \neg T$ (voir les 2 derniers théorèmes du § 1.4), et au lieu de parler de *démonstration* de la première nous parlerons de *réfutation* de la deuxième. La deuxième formule sera transformée dans une autre, en forme standard, en utilisant un algorithme connu sous le nom de *mise sous forme clausale*. Cette forme standard a été introduite en 1960 par Davis et Putnam [42] et elle est largement utilisée depuis. La forme clausale est une formule fermée, en forme normale conjonctive, sans quantificateurs existentiels. Elle se prête très bien à la mécanisation des démonstrations basées sur le théorème de Herbrand (§1.7.2) et en particulier elle est utilisée dans la méthode de résolution (voir Chap.2) et de Prawitz (voir Chap.3). En outre, la formulation des théorèmes dans plusieurs disciplines, conduit naturellement à des représentations sous forme d'ensemble de clauses. Des exemples d'utilisation dans la résolution de problèmes (en général), dans les bases de données, dans la dérivation des programmes à partir des spécifications, entre autres, peuvent être trouvés en [75].

L'algorithme de mise sous forme clausale transforme une fbf A en une autre fbf A' en préservant la satisfaisabilité (insatisfaisabilité). Il faut noter qu'en général A et A' ne sont pas équivalentes. La cause de cette non équivalence a son origine dans un pas de l'algorithme appelé *skolemisation*, qui a pour but d'éliminer les quantificateurs existentiels et que nous présentons d'abord.

1.7.1.1 SKOLEMISATION

L'idée intuitive de la skolemisation est simple. Soit la fbf en forme normale préfixe:

$$\forall x \exists y P(x,y)$$

si elle est satisfaisable, alors il existe une interprétation I avec domaine D , tel que pour tout $a \in D$ il y a *au moins* un $b \in D$ avec $\langle a, b \rangle \in \xi_2(P)$ (voir § 1.6.3). C'est à dire pour chaque "a" on peut choisir un "b" qui *dépend* de "a". On peut donc choisir une *fonction* f telle que:

$$y = f(x)$$

En général f n'est évidemment pas unique, elle dépend du choix particulier de y , parmi tous les y possibles. Cette fonction est appelée *fonction de Skolem*.

La formule initiale peut donc être transformée en une autre qui contient une fonction de Skolem:

$$\forall x P(x, f(x))$$

Les fonctions de Skolem peuvent être des fonctions à zéro ou plusieurs arguments, leur arité étant toujours égale au nombre de quantificateurs universels précédant le quantificateur existentiel éliminé. Par exemple:

$$\exists x \forall y \forall z \exists u \exists v P(x, y, z, u, v)$$

serait transformée en:

$$\forall y \forall z P(a, y, z, f(y, z), g(y, z))$$

Il est naturel de se poser la question: "est-ce qu'on ne change rien en ce qui concerne la validité d'une formule en y introduisant des fonctions de Skolem?". La réponse est donnée dans le lemme suivant:

LEMME: Soit A' une fbf obtenue par skolemisation à partir d'une fbf A . A est satisfaisable (insatisfaisable) ssi A' est satisfaisable (insatisfaisable); autrement dit la skolemisation préserve la propriété d'une formule d'avoir ou de ne pas avoir de modèle.

PREUVE:

Sans perte de généralité on peut supposer A de la forme:

$$A: \forall x_1 \dots \forall x_n \exists y P(x_1, \dots, x_n, y)$$

A' aura donc la forme:

$A': \forall x_1 \dots \forall x_n P(x_1, \dots, x_n, f(x_1, \dots, x_n))$ où f est un symbole fonctionnel qui n'est pas dans A .

1) Si A est satisfaisable, alors elle est évaluée à V dans une interprétation $I = \langle A, \mathbf{m}, \xi \rangle$ avec $\mathbf{m} = \langle D, F, R \rangle$ et $\xi = \langle \xi_1, \xi_2, \xi_3 \rangle$.

C'est à dire pour tout n -uplet $\langle a_1, \dots, a_n \rangle$ dans D^n , il existe au moins un élément de D (disons "b") tel que $\langle a_1, \dots, a_n, b \rangle \in \xi_2(P)$ (voir règles d'évaluation 2,4 et 5 § 1.6.3).

Considérons l'interprétation I' pour A' :

$$I' = \langle A', \mathbf{m}', \xi' \rangle \text{ avec } \mathbf{m}' = \langle D, F', R \rangle \text{ et } \xi' = \langle \xi'_1, \xi_2, \xi_3 \rangle$$

$$F' = F \cup \{ f_* \}$$

- f_* est une fonction de $D^n \rightarrow D$ définie de la façon suivante:
 $f_*(a_1, \dots, a_n) = b$ ssi pour le $n+1$ -uplet $\langle a_1, \dots, a_n, b \rangle$, A est satisfaisable (on fait ici usage implicite de l'axiome de choix puisqu'on admet que l'on peut toujours préciser les valeurs de la fonction f_* et nous n'avons fait aucune hypothèse sur D).

$$\text{domaine}(\xi'_1) = \text{domaine}(\xi_1) \cup \{f\}$$

$$\xi'_1(f) = f_*$$

I' est un modèle de A' . En effet: d'après la définition de f_* et du fait que A est satisfaisable, pour tout n -uplet $\langle a_1, \dots, a_n \rangle \in D^n$

$$\langle a_1, \dots, a_n, f_*(a_1, \dots, a_n) \rangle \in \xi_2(P)$$

donc I' satisfait A' .

2) La converse est démontrée d'une façon tout à fait analogue.

L'existence des fonctions de Skolem pose certains problèmes. On peut utiliser une approche non constructive comme l'on vient de faire; mais une approche constructive est aussi possible (remarque faite dans [42]).

Si A est satisfaisable alors elle l'est dans une interprétation avec domaine dénombrable (théorème de Skolem - Löwenheim § 1.6.3), on peut donc énumérer les n -uplets $\langle x_1, x_2, \dots, x_n \rangle$ dans D^n et assigner au symbole de fonction de Skolem " f " $f_* : D^n \rightarrow D$ ($\xi_1(f) = f_*$) définie de la façon suivante:

$$f_*(x_1, \dots, x_n) = \mu_y$$

où μ_y est le plus petit y tel que pour le

$$n+1\text{-uplet } \langle x_1, x_2, \dots, x_n, y \rangle, A \text{ est évaluée à } V. \blacksquare$$

REMARQUE: Comme nous l'avons déjà avancé, une fbf A et celle obtenue à partir d'elle par skolemisation A' , ne sont pas en général équivalentes, comme le montre l'exemple suivant:

EXEMPLE:

$$A: \exists x P(x)$$

$$A': P(a)$$

Nous évaluons $A' \supset A$ et $A \supset A'$ dans une interprétation définie comme suit:

STRUCTURE:

1) Domaine:

$$D = \{1, 2\}$$

2) Ensemble de relations:

$$R_1 = \{ \langle 2 \rangle \}$$

ASSIGNATION:

$$\xi_1 (a) = 1$$

$$\xi_2 (P) = R_1$$

$$\xi_3 (x) = 2$$

Pour évaluer nous utilisons les règles 2,3 et 4 du § 1.6.3

EVALUATION:

$P(a)$ est évaluée à F : $\langle 1 \rangle \notin R_1$

$P(x)$ est évaluée à V : $\langle 2 \rangle \in R_1$

d'après la règle 4 $\exists x P(x)$ est évaluée à V .

d'après la règle 3 on évalue:

$P(a) \supset \exists x P(x)$ à V

$\exists x P(x) \supset P(a)$ à F

L'explication dans le cas général est que

$A' \supset A$ est valide

tandis que $A \supset A'$ n'est pas V en général

Soit l'implication:

$$\forall x P(x, f(x)) \supset \forall x \exists y P(x, y)$$

il est facile de se convaincre intuitivement que chaque fois que l'antécédent est V alors le conséquent l'est aussi.

La converse:

$$\forall x \exists y P(x, y) \supset \forall x P(x, f(x))$$

peut facilement être falsifiée en exhibant une interprétation dans laquelle la signification de P et de f n'ont pas de rapport: en effet l'appartenance de $\langle x, y \rangle$ à $\xi_2(P)$ ne nous assure pas que

$$\xi_1(f)(\xi_3(x)) = y$$

Des considérations (qui font intervenir la logique de second ordre) extrêmement intéressantes à propos de ce sujet sont faites dans [120].

La règle de skolemisation est la suivante:

Dans une fbf en forme prénexe éliminer tout quantificateur existentiel et le nom de variable correspondante ($\exists x_i$) du préfixe. Toute occurrence de x_i dans la matrice est remplacée par un terme. Ce terme est un symbole fonctionnel n'apparaissant pas dans la formule, il a comme arité le nombre de quantificateurs universels précédant le quantificateur existentiel éliminé et comme arguments les variables quantifiées universellement correspondantes. Dans le cas où $\exists x_i$ n'est précédé d'aucun quantificateur universel, x_i est remplacée dans la matrice par un symbole de constante.

1.7.1.2 ALGORITHME DE MISE D'UNE FBF SOUS FORME CLAUSALE

Considérons le théorème à démontrer comme une formule fermée écrite comme la conjonction des axiomes et de la négation de la conclusion du théorème.

Dans le tableau suivant nous donnons les transformations à appliquer à *chacun des conjoints*, dans l'ordre et itérativement, jusqu'à ce qu'aucune autre application ne soit possible. A chaque pas les propriétés de commutativité et d'associativité des connectifs " \vee " et " \wedge " sont applicables. Les pas 4, 8 et 9 sont facultatifs.

Pas n°	Chaque occurrence de:	La remplacer par:	But
1	$A \supset B$ $A \equiv B$	$\neg A \vee B$ $(A \vee \neg B) \wedge (\neg A \vee B)$	Elimination de \supset et \equiv
2	$\neg \forall x A$ $\neg \exists x A$ $\neg (A \wedge B)$ $\neg (A \vee B)$ $\neg \neg A$	$\exists x \neg A$ $\forall x \neg A$ $\neg A \vee \neg B$ $\neg A \wedge \neg B$ A	Déplacer les \neg le plus à l'intérieur possible (on arrive à ce que les \neg précédent une formule atomique).
3	une variable liée dont le nom est partagé par deux quantificateurs.	une nouvelle variable (dans l'un des cas) dans le quantificateur et dans l'occurrence liée correspondante.	Il n'y a pas deux variables liées partageant le même nom.
4	$\exists x (A \vee B)$ $\forall x (A \vee B)$ $\exists x (A \wedge B)$ $\forall x (A \wedge B)$ $\forall x (A \wedge B)$ $\exists x (A \vee B)$ $\forall x \forall y (A \vee B)$ $\exists x \exists y (A \wedge B)$	$(A \vee \exists x B)$ (1) ou $(\exists x A \vee B)$ (2) $(A \vee \forall x B)$ (1) ou $(\forall x A \vee B)$ (2) $(A \wedge \exists x B)$ (1) ou $(\exists x A \wedge B)$ (2) $(A \wedge \forall x B)$ (1) ou $(\forall x A \wedge B)$ (2) $\forall x A \wedge \forall x B$ (3) $\exists x A \vee \exists x B$ (3) $\forall y \forall x (A \vee B)$ (4) $\exists y \exists x (A \wedge B)$ (4)	Déplacer les quantificateurs vers la droite. Permet dans certains cas de minimiser l'arité des fonctions de Skolem introduites dans le pas 5 (en réduisant le nombre de quantificateurs universels qui ont des quantificateurs existentiels dans leur portée). Ceci a des conséquences pratiques parce qu'on peut réduire ainsi la complexité de la recherche d'une réfutation.

(1): si x est liée dans A
(2): si x est liée dans B
(3): si x est libre dans A et dans B .
(4): si x est liée dans A ou dans B et y est libre dans A et dans B

Pas n°	Chaque occurrence de:	La remplacer par:	But
5	une variable quantifiée existentiellement.	une fonction de Skolem et éliminer le quantificateur existentiel du préfixe.	Élimination des quantificateurs existentiels par skolemisation.
6	un quantificateur universel.	son déplacement tout à fait à gauche de la formule.	Transformation de la formule en forme prénexé (sans quantificateurs existentiels, puisqu'ils ont été éliminés au pas 5).
7	$A \vee (B \wedge C)$ $(B \wedge C) \vee A$	$(A \vee B) \wedge (A \vee C)$ $(B \vee A) \wedge (C \vee A)$	Transformation de la matrice en forme normale conjonctive.
8	$F \vee G \vee F$ dans la matrice. $F \vee G \vee \neg F$ dans la matrice.	$F \vee G$ élimination du conjoint.	Simplification.
<p>Une fois les pas 1 à 8 effectués sur tous les conjoints, appliquer le pas 9. On dit qu'un conjoint F "subsume" un autre conjoint G si tout disjoint de F est (aux noms des variables près) dans G. Exemple $P(x_1) \vee Q(x_1)$ subsume $P(x_2) \vee Q(x_2) \vee R(a, x_2)$.</p>			
9	un conjoint G subsumé par un conjoint F .	élimination du conjoint G .	Simplification.

EXEMPLE: Nous appliquons l'algorithme au théorème sur le théorème des groupes énoncé au § 1.7.1.

1) TRANSFORMATION DE A1:

Pas 5: $\forall x \forall y P(x, y, f(x, y))$

A'1: $\forall x \forall y P(x, y, f(x, y))$

2) TRANSFORMATION DE A2:

Pas 1: $\forall x \forall y \forall z \forall u \forall v \forall w (\neg(P(x, y, u) \wedge P(y, z, v)) \vee ((P(x, v, w) \vee \neg P(u, z, w)) \wedge (\neg P(x, v, w) \vee P(u, z, w))))$

Pas 2: $\forall x \forall y \forall z \forall u \forall v \forall w ((\neg P(x, y, u) \vee \neg P(y, z, v)) \vee ((P(x, v, w) \vee \neg P(u, z, w)) \wedge (\neg P(x, v, w) \vee P(u, z, w))))$

DISTRIBUTIVITE DE "v":

$$\forall x \forall y \forall z \forall u \forall v \forall w ((\neg P(x,y,u) \vee \neg P(y,z,v)) \vee (P(x,v,w) \vee \neg P(u,z,w))) \wedge ((\neg P(x,y,u) \vee \neg P(y,z,v)) \vee (\neg P(x,v,w) \vee P(u,z,w)))$$

ASSOCIATIVITE DE "v":

$$A'2: \forall x \forall y \forall z \forall u \forall v \forall w ((\neg P(x,y,u) \vee \neg P(y,z,v) \vee P(u,z,w) \vee P(x,v,w)) \wedge (\neg P(x,y,u) \vee \neg P(y,z,v) \vee \neg P(x,v,w) \vee P(u,z,w)))$$

3) TRANSFORMATION DE A3:

$$\text{Pas 3: } \exists x (\forall y P(x,y,y) \wedge \forall u \exists z P(z,u,x))$$

$$\text{Pas 5: } \exists x (\forall y P(x,y,y) \wedge \forall u P(g(u),u,x))$$

$$\forall y P(a,y,y) \wedge \forall u P(g(u),u,a)$$

$$A'3: \forall y P(a,y,y) \wedge \forall u P(g(u),u,a)$$

4) TRANSFORMATION DE A4: $\neg T$

$$\neg (\exists x (\forall y P(x,y,y) \wedge \forall y \exists z P(y,z,x)))$$

$$\text{Pas 2: } \forall x (\neg \forall y P(x,y,y) \vee \neg \forall y \exists z P(y,z,x))$$

$$\forall x (\exists y \neg P(x,y,y) \vee \exists y \forall z \neg P(y,z,x))$$

$$\text{Pas 3: } \forall x (\exists y \neg P(x,y,y) \vee \exists u \forall z \neg P(u,z,x))$$

$$\text{Pas 5: } \forall x (\neg P(x,h(x),h(x)) \vee \forall z \neg P(k(x),z,x))$$

$$\text{Pas 6: } \forall x \forall z (\neg P(x,h(x),h(x)) \vee \neg P(k(x),z,x))$$

$$T' : \forall x \forall z (\neg P(x,h(x),h(x)) \vee \neg P(k(x),z,x))$$

Nous n'avons pas eu la possibilité d'appliquer le pas 4 dans la transformation de notre formule, mais en voici un exemple d'application [93]:

Soit la formule $\forall u \exists y \forall x \exists v (P(x,y) \wedge Q(u,v))$.

Si l'on n'applique pas le pas 4 la formule transformée obtenue est:

$$\forall u \forall x (P(x,f(u)) \wedge Q(u,g(u,x))).$$

Si l'on applique le pas 4 on obtient:

$$\exists y \forall x P(x,y) \wedge \forall u \exists v Q(u,v)$$

$$\forall x P(x,a) \wedge \forall u Q(u,f(u)).$$

C'est à dire après application du pas 6:

$$\forall x \forall u (P(x,a) \wedge Q(u,f(u))).$$

L'application de ce pas d'essai de minimisation du nombre d'arguments des fonctions de Skolem n'assure évidemment pas qu'aucune minimisation ne soit encore possible (un algorithme plus complexe serait nécessaire pour cela). On retiendra comme conclusion qu'il peut y avoir pour une même formule plus d'une forme

normale ou standard.

1.7.1.3 TERMINOLOGIE POUR LA DEMONSTRATION AUTOMATIQUE

Δ Un atome ou sa négation s'appelle un *littéral*. ▽

Δ Si l est un littéral, le littéral l' est son *littéral complémentaire* ssi il ne diffère du premier que dans le symbole de négation (l et l' sont dits des *littéraux complémentaires*). Parfois on note le complémentaire d'un littéral l par l^c [89]. ▽

Δ D'un littéral qui est un atome on dit qu'il est un *littéral positif*, d'un littéral qui n'est pas un atome on dit qu'il est un *littéral négatif*. ▽

Δ Une *clause* est une disjonction de littéraux (on appelle aussi clause un *ensemble de littéraux*). La différence entre les deux significations sera faite par le contexte d'utilisation. ▽

Δ Un *ensemble de clauses* S est un ensemble qui représente la *conjonction* de toutes les clauses de S . ▽

Δ La *clause vide*, notée \square , est une clause qui ne contient aucun littéral. ▽

L'algorithme de mise sous forme clausale d'une formule permet de voir facilement que:

Toutes les variables de toutes les clauses sont quantifiées universellement et il n'y a pas deux clauses contenant un même nom de variable.

THEOREME: Toute fbf A peut être mise en forme clausale, donnant une fbf A' (ou ce qui est équivalent: un ensemble de clauses S). A' est satisfaisable (insatisfaisable) ssi A l'est.

PREUVE: Dans [27], [89], [93]. ■

La forme clausale du théorème de la théorie des groupes est la suivante: (on peut renommer les variables de même nom que celles d'une clause différente si cela prête à confusion):

$$P(x,y,f(x,y))$$

$$\neg P(x,y,u) \neg P(y,z,v) \neg P(u,z,w) \quad P(x,v,w)$$

$$\neg P(x,y,u) \neg P(y,z,v) \neg P(x,v,w) \quad P(u,z,w)$$

$$P(a,y,y)$$

$$P(g(u),u,a)$$

$$\neg P(x,h(x),h(x)) \neg P(k(x),z,x)$$

où il y a des " \vee " implicites entre les littéraux dans chaque ligne, et des "

implicites entre les clauses constituées par les lignes.

REMARQUE 1: D'un point de vue théorique il n'y a pas à faire de différence entre une clause provenant des axiomes et une clause provenant de la négation du théorème. Cependant une *stratégie de recherche* (voir Chapitre 2) les traitera *différemment*.

REMARQUE 2: Par transformation en forme clausale d'une fbf on obtient toujours un ensemble *fini* de clauses (évident d'après l'algorithme de transformation).

REMARQUE 3: La forme clausale n'est pas toujours le plus adéquate (dans le sens de l'efficacité) ni la plus naturelle pour trouver une démonstration d'un théorème. En outre, l'utilisation de la distributivité du " \vee " (pas 7 de l'algorithme de transformation) cause une prolifération de littéraux.

Ces difficultés se présentent aussi pour des théorèmes de la logique d'ordre supérieur. En effet, certains théorèmes dans ces logiques peuvent facilement être traduits en des théorèmes de la logique du premier ordre et en exiger seulement les méthodes [3]. L'utilisation combinée de la forme clausale et d'une forme dans laquelle on permet des occurrences des quantificateurs non nécessairement au début de la formule, ou des connectifs autres que " \vee " et " \wedge ", est parfois souhaitable ([3], [75]).

1.7.2 LA METHODE DE HERBRAND

Nous abordons maintenant l'exposé de la méthode qui est à l'origine de la plupart des réalisations de démonstrateurs automatiques.

Puisqu'on sait qu'il n'y a pas de procédure de décision, nous pouvons essayer de trouver une procédure de preuve. Dans l'approche syntaxique, elle est facile à imaginer: on énumère les théorèmes (§ 1.3); si l'on arrive à la formule dont on veut savoir si elle est un théorème on arrête, sinon on continue (on fait évidemment abstraction de la complexité d'une telle procédure et de l'impossibilité pratique de sa mise en œuvre dans le cas général). Mais une procédure est moins facile à imaginer dans l'approche sémantique parce que la définition de validité exige que la formule testée soit vraie pour *toutes* les interprétations ou, ce qui est équivalent, sa négation fausse pour *toutes* les interprétations, donc on n'a pas en apparence une réponse (au moins partielle) à la question: "quand arrête-t-on?".

En fait la réponse sera donnée par la méthode de Herbrand, qui réduit le problème à des *tests d'insatisfaisabilité utilisant les règles du calcul propositionnel*.

Nous donnons auparavant les définitions et résultats nécessaires à l'énoncé et à la démonstration du théorème de Herbrand.

THEOREME (de Compacité): Un ensemble S de fbf fermées est insatisfaisable ssi il existe un sous-ensemble fini S_0 de S insatisfaisable.

Un énoncé équivalent est évidemment:

Un ensemble de S de fbf fermées est satisfaisable ssi tout sousensemble fini S_i de S est satisfaisable.

PREUVE: Dans [6], [20], [89].■

Le théorème de compacité doit son nom au fait qu'il implique la compacité d'un certain espace topologique [60].

Ce théorème est appelé aussi *loi de la conjonction infinie* [115] et correspond au lemme de l'infini de König : Dans tout arbre orienté infini (c'est à dire avec une infinité de nœuds) dans lequel tout nœud n'a qu'un nombre fini de nœuds descendants il y a un chemin infini à partir de la racine.

Δ Soit S un ensemble de clauses, l'univers de Herbrand de S , noté $H(S)$ est défini comme suit:

. H_0 est l'ensemble de constantes apparaissant dans des clauses de S .

Si cet ensemble est vide H_0 contient un seul élément: une constante que nous appellerons conventionnellement " a".

. $H_{i+1} = \{ f_j^n (t_1, \dots, t_n) \} \cup H_i$ ($i \geq 0, j \geq 0, n \geq 1$)
où f_j^n est un symbole fonctionnel dans les clauses de S et t_i ($1 \leq i \leq n$) sont des éléments de H_i .

. $H(S) = H_\infty$

$H_i (i \geq 0)$ est appelé l'ensemble de termes constants de niveau i de S .

Les éléments de l'univers de Herbrand sont des termes fermés appelés *termes de Herbrand de S* (ou plus simplement, si l'on sous-entend l'ensemble de clauses S : *termes de Herbrand*).

L'univers de Herbrand va être l'univers "privilégié", dans ce sens qu'il suffira de s'intéresser à lui comme seul univers possible.

Soit S un ensemble fini de clauses (ce qui est le cas le plus courant) alors:

a) si dans S il n'y a pas de symbole fonctionnel d'arité plus grand que zéro $H(S)$ est fini.

b) autrement $H(S)$ est infini dénombrable.

EXEMPLE 1:

$$S = \{ P(x,y), \neg Q(z) \vee P(y,z) \}$$

REMARQUE: Cette façon de représenter les clauses est parfois utilisée (par exemple dans [89]); on sous-entend un symbole " \vee " entre deux littéraux non sé-

parés par une " ,".

Une autre représentation [27] indique explicitement le " v " :

$$S = \{ P(x,y), \neg Q(z) \vee P(y,z) \}$$

Nous utiliserons indistinctement l'une ou l'autre de ces représentations.

$$H(S) = \{ a \}$$

EXEMPLE 2:

$$S = \{ P(x,a), Q(f(b)) \}$$

$$H(S) = \{ a, b, f(a), f(b), f(f(a)), f(f(b)), \dots \}$$

Δ Un littéral dont les termes sont des termes de Herbrand est appelé un *littéral constant* (ou *c-littéral*) ou un *cas particulier* d'un littéral de S. ∇

Δ Une *instance constante* d'une clause de S (ou *c-clause*) est une clause de S où toutes les variables ont été remplacées par des termes de Herbrand (autrement dit où tous les littéraux sont des littéraux constants). ∇

Nous appellerons c-littéral aussi des symboles propositionnels et c-clauses aussi à des clauses dont les littéraux sont des symboles propositionnels.

EXEMPLE:

$$S = \{ P(x,y), \neg Q(f(x),b) P(a,z) \}$$

Des instances constantes de clauses de S (parmi toutes les possibles) sont:

$$P(a,b)$$

$$P(f(f(a)),f(b))$$

$$\neg Q(f(a),b) \vee P(a,f(b))$$

$$\neg Q(f(f(b)),f(a)) \vee P(a,b)$$

Δ Soit S un ensemble de clauses, l'ensemble de toutes les instances constantes de tous les atomes des clauses de S est appelé la *base de Herbrand de S* ou *l'ensemble d'atomes de S*, et il est noté $B(S)$. ∇

EXEMPLE:

$$S = \{ P(x) \vee Q(y), \neg P(f(a)) \vee \neg Q(b) \}$$

L'univers de Herbrand de S:

$$H(S) = \{ a, b, f(a), f(b), f(f(a)), f(f(b)), \dots \}$$

La base de Herbrand de S:

$$B(S) = \{ P(a), Q(a), P(b), Q(b), P(f(a)), \dots \}$$

Une autre notion très importante et sur laquelle nous reviendrons au Chapitre 2 est la suivante:

Δ Soit S un ensemble de clauses et P un ensemble de termes de Herbrand de S; on appelle *saturation de S sur P*, notée $SAT_P(S)$, l'ensemble de toutes les c-clauses obtenu en remplaçant les variables des clauses de S par des termes dans

P (les occurrences de la même variable dans une clause étant remplacées par le même terme). \forall

A Soit S un ensemble de c-clauses. Un ensemble M de littéraux de clauses de S est dit un *c-modèle* de S ssi il ne contient pas de paire de littéraux complémentaires et pour tout C dans S: $C \cap M \neq \emptyset$.

En général, si S est un ensemble de clauses, on dit que M est un c-modèle de S ssi il est un c-modèle de $SAT_H(S)$. \forall

Le lemme suivant correspond à l'idée intuitive suivante: comme pour le calcul propositionnel (§ 1.2.1) on peut penser un c-modèle M comme un ensemble de littéraux (éventuellement infini) qui seront évalués à V.

Comme chaque clause contient au moins un littéral de M alors toutes les clauses seront évaluées à V et leur conjonction sera donc aussi évaluée à V.

LEMME: Si A est une fbf et S l'ensemble de clauses correspondant, alors A est satisfaisable ssi $SAT_H(S)$ a un c-modèle.

PREUVE: Elle est basée sur l'idée intuitive avancée ci-dessus. A partir d'un c-modèle de $SAT_H(S)$ on peut construire une interprétation qui satisfait A et viceversa; à partir d'une interprétation qui satisfait A, on peut construire un c-modèle de $SAT_H(S)$ (les détails de la démonstration dans [89]).■

Le théorème suivant fournit une procédure de preuve pour le calcul de prédicat du premier ordre, en ramenant le problème de la validité (insatisfaisabilité) d'une fbf dans le dit calcul à un test de validité (insatisfaisabilité) d'un ensemble de c-clauses qui peut donc être évalué avec les règles du calcul propositionnel.

THEOREME (de Herbrand [64]): Soit A une fbf fermée, S l'ensemble de clauses correspondant. A est insatisfaisable ssi il existe un ensemble fini de c-clauses (c'est à dire un ensemble fini $T \subseteq SAT_H(S)$) insatisfaisable.

(On pourrait aussi dire "...ssi il existe un ensemble fini de c-clauses dont la conjonction est évaluée à faux").

PREUVE:

a) Supposons existe $T/T \subseteq SAT_H(S)$, T fini et insatisfaisable. $SAT_H(S)$ n'a pas de c-modèle M, parce que s'il en avait un il y aurait dans M au moins un littéral de chaque clause de $SAT_H(S)$, donc au moins un littéral de chaque clause de T et en évaluant à V chacun des littéraux de M et de T on aurait un modèle de T, contredisant l'hypothèse. $SAT_H(S)$ n'a donc pas de c-modèle: par lemme antérieur, A est insatisfaisable.

b) Pour prouver la converse nous prouvons la contraposition équivalente.

Supposons que pour tout R fini ($R \subseteq SAT_H(S)$) R est satisfaisable, alors

par théorème de compacité. $SAT_{H(S)}(S)$ est satisfaisable. Si $SAT_{H(S)}(S)$ est satisfaisable alors on évalue à V au moins un littéral de chacune de ses clauses, on peut construire l'ensemble de littéraux M avec tous ces littéraux (M est donc un c -modèle de $SAT_{H(S)}(S)$).

Si $SAT_{H(S)}(S)$ a un c -modèle alors par lemme antérieur A est satisfaisable. ■

Les deux lemmes suivantes donnent deux propriétés importantes des ensembles de clauses insatisfaisables.

LEMME: *Si l'on ajoute des clauses à un ensemble de clauses insatisfaisable alors on obtient un ensemble de clauses insatisfaisable.*

PREUVE: Soient S l'ensemble de clauses insatisfaisable et S' l'ensemble de clauses élargi. D'après le th. de Herbrand il existe un ensemble fini de c -clauses de S insatisfaisable: T . Il suffit de considérer un ensemble fini de c -clauses quelconques de $S'-S$: T' . L'ensemble $\{T, T'\}$ est clairement insatisfaisable (propriété de la conjonction: $T \wedge T'$ est évaluée à F si T ou T' le sont). Par théorème de Herbrand S' est donc insatisfaisable. ■

Cette propriété étant triviale il est plus intéressant de chercher des sous-ensembles insatisfaisables d'un ensemble insatisfaisable.

Δ Un ensemble de clauses S est *minimalement insatisfaisable* ssi S est insatisfaisable et il n'existe pas de sous-ensemble propre de S insatisfaisable. ▽

LEMME: *Un ensemble de clauses minimalement insatisfaisable est fini.*

PREUVE: Supposons S infini, alors par théorème de compacité il existe T fini insatisfaisable, $T \subset S$. Ce qui contredit la propriété de S d'être minimalement insatisfaisable. ■

Du premier de ces deux derniers lemmes et de la notion de minimalement insatisfaisable on peut aisément conclure que tout ensemble de clauses insatisfaisable contient un sous-ensemble minimalement insatisfaisable, mais *il n'est pas unique en général* (exemple: $S = \{A, B, \neg A, \neg B\}$).

Comme nous l'avons avancé au § 1.7.1 dans les méthodes basées sur le théorème de Herbrand, si l'on veut prouver le théorème:

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \supset B$$

(où les A_i ($1 \leq i \leq n$) et B sont des fbf fermées), l'approche la plus fréquente consiste à passer à la forme clausale et à montrer l'insatisfaisabilité de la conjonction:

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \wedge \neg B$$

et les méthodes de preuve utilisant cette approche sont appelées *procédures de réfutation*.

Il serait équivalent de montrer la validité de la disjonction:

$$\neg A_1 \vee \neg A_2 \vee \dots \vee \neg A_n \vee B$$

Les procédures utilisant cette autre approche sont appelées *procédures de validation*. Cette approche est celle utilisée dans [97]. Un exemple d'utilisation des procédures de validation peut être trouvé dans [89].

1.8 POURQUOI CE CHAPITRE ?

Nous avons présenté dans ce chapitre les notions de logique que nous considérons comme indispensables pour comprendre la suite, ainsi que la terminologie spécifique et les principes sur lesquels sont basés les méthodes présentées dans les chapitres suivants. Nous dirons quelques mots pour justifier la place accordée au premier de ces points (la justification du deuxième point n'étant évidemment pas nécessaire).

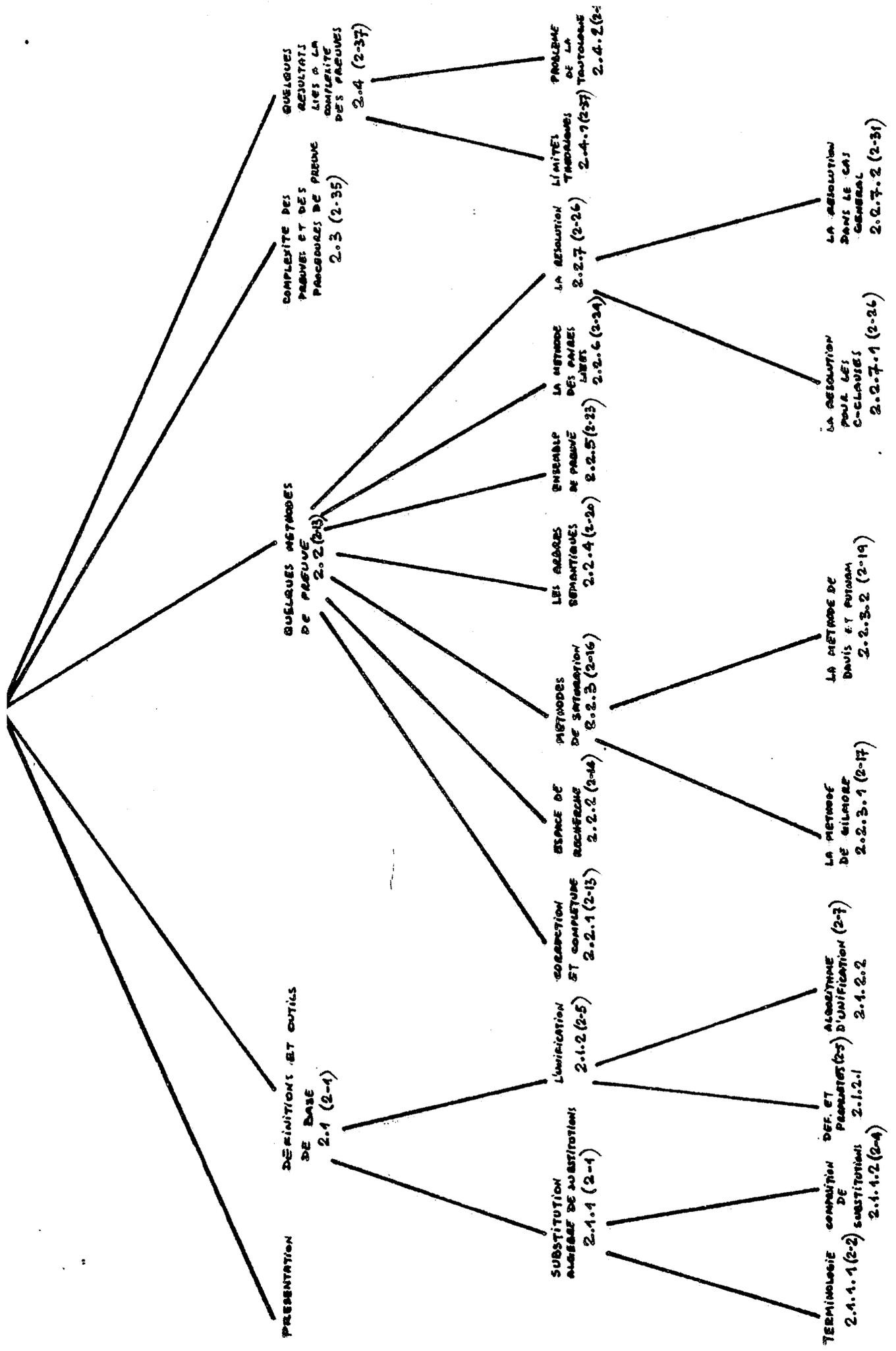
Il y a de bonnes raisons pour considérer la logique du 1^{er} ordre comme le langage de base des mathématiques. Mendelson [100] écrit: "Les théories de 1^{er} ordre suffisent pour exprimer les théories mathématiques connues, et en tout cas, la plupart des théories d'ordre supérieur peuvent être "traduites" d'une façon convenable en des théories du 1^{er} ordre" (il faudrait pour cela *explicitement* les suppositions mathématiques *implicites* - par exemple les propriétés des nombres naturels, au moyen d'axiomes exprimés dans la logique du 1^{er} ordre, ce qui est toujours possible).

Bien que ce point de vue théorique fasse l'objet d'un consensus parmi beaucoup de mathématiciens et logiciens, son utilisation dans la pratique mathématique peut être extrêmement encombrante et même contre-indiquée; il est bien connu que la logique du 1^{er} ordre ne permet pas d'exprimer, par exemple, un concept aussi couramment utilisé que celui de "bon ordre". Des logiciens ont développé des extensions du calcul de prédicats du 1^{er} ordre qui permettent d'absorber dans la logique certaines notions ou structures mathématiques très utilisées et qui ne peuvent pas être exprimées d'une façon simple en logique du 1^{er} ordre (le problème du pouvoir d'expression relatif de logiques d'ordres différents a donné lieu à un certain nombre de publications ces dernières années).

Nous finissons ce chapitre en rappelant l'importance croissante de l'utilisation de la logique en Intelligence Artificielle (dont la démonstration automatique fait partie) en citant l'opinion de N. Nilsson [103]: "La logique est le langage pour décrire le raisonnement mécanisé", à laquelle il ajoute le souhait que son utilisation se généralise, évitant ainsi les langages ad hoc, souvent inadéquats, utilisés par beaucoup de chercheurs.

CHAPITRE 2

OUTILS DE BASE EN DEMONSTRATION AUTOMATIQUE ET QUELQUES METHODES DE PREUVE



On peut situer le renouvellement de l'intérêt pour la mécanisation des procédures de test de validité de formules du Calcul de Prédicats dans les années 1960 avec les travaux de Wang [157], Prawitz et al. [113], Gilmore [55], Davis et Putnam [42]. Il faut citer aussi la méthode présentée en 1955 par Quine [116].

Les méthodes, ou procédures de preuve, sont des techniques permettant de détecter si une formule du Calcul de Prédicats est insatisfaisable (il est facile de vérifier que cette définition est équivalente à la définition de procédure de preuve (ou de sémidécision) donnée au § 1.5).

Nous parlerons de quelques méthodes de preuve après avoir présenté des outils de base.

Nous commencerons donc ce chapitre en parlant des notions de substitution et d'unification dont l'importance, bien que fondamentale en démonstration automatique, dépasse largement ce domaine.

Ensuite, nous parlerons de quelques procédures de preuve (basées sur la méthode de Herbrand) pour des formules du Calcul de Prédicats du premier ordre (ces procédures peuvent s'appliquer également à des formules du Calcul Propositionnel).

Nous avons choisi de parler de ces méthodes parce qu'elles représentent des points de repère dans l'histoire de la démonstration automatique et par leur rapport avec la méthode que nous avons implantée.

Dans la présentation des différentes méthodes, nous supposons que la formule à réfuter est sous forme clausale (§ 1.7.1.3); en fait, dans la méthode de Gilmore ce n'est pas exactement le cas, mais on peut le supposer sans changer en rien l'essence de la méthode.

Pour finir, nous parlerons de quelques résultats fondamentaux sur la complexité des preuves, sur le problème de la tautologie (en particulier des théorèmes de Cook) ainsi que sur des résultats plus récents sur la complexité moyenne des preuves.

2.1 OUTILS DE BASE : SUBSTITUTION ET UNIFICATION

2.1.1 SUBSTITUTION, ALGÈBRE DE SUBSTITUTIONS

Nous noterons (voir § 1.6.1 définition du langage du Calcul de Prédicats)

V : ensemble de variables.

T : ensemble de termes.

F : ensemble de symboles de fonction.

P : ensemble de symboles de prédicat.

Δ Une *substitution* est une application de V dans T .

On s'intéresse presque toujours à des substitutions qui coïncident avec la fonction identité sauf dans un nombre fini de points; nous adopterons donc comme dé-

finition:

Δ Une *substitution* est une application de V dans T , identité presque partout, c'est à dire sauf dans un ensemble fini de V [68]. ∇

Une définition, qui correspond à la restriction de la fonction à l'ensemble de variables pour lequel la fonction n'est pas l'identité est largement utilisée dans la littérature sur la démonstration automatique (voir par exemple [27], [89], [121]).

Dans la définition que nous avons adoptée, le fait qu'une substitution soit une *fonction totale* sur V nous permettra de définir naturellement la composition de substitutions (§ 2.1.1.1).

NOTATION: Nous noterons, selon l'usage généralisé, les substitutions avec des lettres grecques.

La représentation utilisée pour une substitution σ dans le cas le plus courant (celui où la substitution diffère de l'identité dans un nombre fini de points) est:

$$\sigma = \{ t_1 \rightarrow x_1, t_2 \rightarrow x_2, \dots, t_n \rightarrow x_n \}$$

ou

$$\sigma = \{ t_1 / x_1, t_2 / x_2, \dots, t_n / x_n \}$$

ou encore

$$\sigma = \{ \langle x_1, t_1 \rangle, \langle x_2, t_2 \rangle, \dots, \langle x_n, t_n \rangle \}$$

$$x_i \in V \quad t_i \in T \quad x_i \neq t_i \quad (1 \leq i \leq n)$$

Avec la définition adoptée, on sous-entend:

$$\sigma(x_j) = x_j \quad \forall x_j \neq x_i \quad (1 \leq i \leq n)$$

Nous utiliserons indistinctement les trois représentations ci-dessus.

2.1.1.1 TERMINOLOGIE ET EXTENSION DE LA DEFINITION DE SUBSTITUTION

Δ Les éléments $t_i \rightarrow x_i$ (ou t_i / x_i ou $\langle x_i, t_i \rangle$) s'appellent *composants de substitution*. La variable x_i est appelée *nom du composant* et le terme t_i est appelé *terme composant*. ∇

Δ Soit σ une substitution; $D_\sigma = \{ x \in V / \sigma(x) \neq x \}$. Si pour tout $x \in D_\sigma$ $\sigma(x) \in V$ et la restriction de σ à D_σ est injective, alors σ est dite une *substitution de renommage des variables de D_σ* . ∇

Δ Si tous les termes composants d'une substitution sont des termes fermés, la substitution est dite *sans variable*. ∇

Δ La substitution identité partout est appelée la *substitution identité* ou parfois la *substitution vide*, et elle est notée ϵ ("vide" parce que la substitution est représentée par l'ensemble vide). ∇

On étend le domaine d'application des substitutions à l'ensemble des termes, avec la définition suivante:

Δ Soit σ une substitution, $t \in T$

1) Si t est une variable ($t = x$) alors $\sigma(t) = \sigma(x)$

2) Si t est une constante ($t = a$) alors $\sigma(t) = a$

3) Si t n'est ni variable ni constante ($t = f^n(t_1, \dots, t_n)$; $n \geq 1$) alors

$$\sigma(t) = f^n(\sigma(t_1), \dots, \sigma(t_n)) \quad \forall f^n \in F \quad (n \geq 1).$$

Une fois cette définition adoptée, on peut étendre d'une façon naturelle le domaine d'application à:

Δ . Un ensemble de termes:

$\sigma(\{t_1, \dots, t_n, \dots\}) = \{\sigma t_1, \dots, \sigma t_n, \dots\}$

. Un littéral:

si $L = P$ $\sigma(L) = P$

si $L = \neg P$ $\sigma(L) = \neg P$

si $L = P(t_1, \dots, t_n)$ $\sigma(L) = P(\sigma t_1, \dots, \sigma t_n)$

si $L = \neg P(t_1, \dots, t_n)$ $\sigma(L) = \neg P(\sigma t_1, \dots, \sigma t_n)$

. Un ensemble de littéraux (en particulier une clause):

$C = \{L_1, \dots, L_n, \dots\}$

$\sigma(C) = \{\sigma L_1, \dots, \sigma L_n, \dots\}$

. Un ensemble de clauses:

$S = \{C_1, \dots, C_n, \dots\}$

$\sigma(S) = \{\sigma C_1, \dots, \sigma C_n, \dots\} . \forall$

Nous noterons indistinctement $\sigma(t)$, $\sigma(L)$, $\sigma(C)$, $\sigma(S)$ ou σt , σL , σC , σS .

Δ $\sigma(L)$, $\sigma(C)$ et $\sigma(S)$ s'appellent des *instantiations* de L , C et S par σ ou des *instances* de L , C , S . \forall

Δ Soit L un littéral et σ une substitution de renommage de variables apparaissant dans L , σL est dite une *variante* (ou une *copie*) de L . \forall

Δ Si $C = \{L_1, L_2, \dots, L_n\}$ est une clause, $C' = \{L'_1, L'_2, \dots, L'_n\}$ est une *variante* (ou une *copie*) de C ssi il existe L'_i ($1 \leq i \leq n$) tel que L' est une variante de L_i . \forall

Un concept très important utilisant la notion de substitution est le suivant:

Δ Etant donné deux clauses A et B , on dit que A *subsume* B [27] ou que A θ -*subsume* B [89] ssi: $\text{card}(A) \leq \text{card}(B)$ et il existe une substitution θ tel que $\theta A \subseteq B$. \forall

REMARQUE: Nous avons confondu les définitions de subsomption et θ -subsomption, ce qui suffit pour nos besoins. Dans [89] la subsomption est un critère plus fort que la θ -subsomption.

2.1.1.2 COMPOSITION DE SUBSTITUTIONS

Une fois qu'on a étendu le domaine d'application des substitutions à l'ensemble des termes, le domaine et le co-domaine des substitutions coïncident; on peut donc considérer leur composition.

La définition de composition de deux substitutions est évidemment celle de composition de deux fonctions, c'est-à-dire:

$$\Delta \text{ Etant donné deux substitutions } \lambda \text{ et } \theta$$

$$\lambda : T \rightarrow T \text{ (en fait } \lambda : V \rightarrow T; \quad V \subset T \text{)}$$

$$\theta : T \rightarrow T \text{ (en fait } \theta : V \rightarrow T; \quad V \subset T \text{)}$$

la composition de λ avec θ , notée $\lambda \circ \theta$, est définie comme suit:

$$\forall t \in T \text{ (en fait } \forall t \in V \text{)} \quad (\lambda \circ \theta)(t) = \lambda(\theta(t))$$

C'est-à-dire pour obtenir le terme résultat de $(\lambda \circ \theta)(t)$ on applique d'abord θ , puis on applique λ au résultat \forall .

CONVENTION PRATIQUE: Parfois nous écrivons $\lambda\theta$ à la place de $\lambda \circ \theta$.

EXEMPLE:

$$\theta = \{ a \rightarrow x, f(z) \rightarrow y, x \rightarrow z \}$$

$$\lambda = \{ b \rightarrow x, c \rightarrow z, d \rightarrow u \}$$

(pour toute autre variable que celles apparaissant explicitement, θ et λ coïncident avec l'identité - voir § 2.1.1-)

$$\lambda \circ \theta = \{ a \rightarrow x, f(c) \rightarrow y, b \rightarrow z, d \rightarrow u \}$$

$$(\lambda \circ \theta)(v) = v \quad \forall v \in V - \{ x, y, z, u \}$$

En appliquant les définitions concernant les substitutions et les résultats connus sur la composition des fonctions, la démonstration du lemme suivant est triviale.

LEMME: L'ensemble des substitutions avec l'opération de composition est un monoïde. C'est-à-dire, si θ, λ, σ sont des substitutions quelconques:

- 1) L'opération de composition de substitutions est une opération interne.
- 2) $\varepsilon \circ \theta = \theta \circ \varepsilon = \theta$ (existence d'élément neutre: ε , la substitution identité partout).
- 3) $(\theta \circ \lambda) \circ \sigma = \theta \circ (\lambda \circ \sigma)$ (associativité).

PREUVE: En appliquant les résultats de la théorie des fonctions (voir par ex. [89], [120]).

On peut se demander si l'ensemble de substitutions n'a pas une structure plus riche, par exemple existence de substitution inverse d'une substitution ou la commutativité de la composition. La réponse est négative dans le cas général (voir remarques ci-dessous):

Δ On dira qu'une substitution θ est *invertible* ssi il existe une substitution λ -notée θ^{-1} et appelée *inverse* - tel que $\theta \circ \theta^{-1} = \varepsilon$ (cette notion correspond évidemment à la notion d'inverse à droite). ▽

REMARQUE 1: En général, l'inverse n'existe pas.

EXEMPLE: $\sigma = \{ a \rightarrow x \}$

Puisque "a" ne peut pas être un nom de composant, σ^{-1} n'existe pas.

REMARQUE 2: En général, la composition des substitutions n'est pas commutative.

EXEMPLE:

$$\theta = \{ f(x) \rightarrow x \}$$

$$\lambda = \{ g(x) \rightarrow x \}$$

$$\theta \circ \lambda = \{ g(f(x)) \rightarrow x \}$$

$$\lambda \circ \theta = \{ f(g(x)) \rightarrow x \}$$

C'est-à-dire $\theta \circ \lambda \neq \lambda \circ \theta$

L'exemple trivial de substitution qui commute avec toute autre substitution est celui de la substitution identité ε .

2.1.2 L'UNIFICATION

2.1.2.1 DEFINITION ET PROPRIETES

Comme nous le verrons au § 2.2, une limitation inhérente aux méthodes dites de saturation est d'engendrer exhaustivement des c-clauses, pour tester leur contradiction en appliquant le théorème de Herbrand (§ 1.7.2). En 1960 Prawitz [110] suggéra de n'engendrer que les clauses contenant des littéraux complémentaires; pour ce faire, il fallait "égaliser" les arguments des atomes correspondants. Ce concept fut précisé par Robinson en 1965 [121] avec sa définition d'unificateur le plus général; il donna aussi un algorithme de calcul.

Δ Etant donné l'équation dans les termes

$$(a) \quad t_1 = t_2$$

on appelle *unificateur* des termes t_1 et t_2 toute *solution* de l'équation, c'est-à-dire toute substitution σ , telle que:

$$(b) \quad \sigma t_1 = \sigma t_2 \quad .\nabla$$

REMARQUE: Dans l'équation, les variables sont prises au *sens conditionnel* [73]; elles ne sont pas liées (§ 1.6.1). Dans (a) donc le signe "=" ne correspond pas à l'identité, tandis que dans (b) il correspond à l'identité.

L'équation établit une *propriété* des termes et la (les) solution(s) donne(nt) les valeurs des variables des termes qui satisfont cette propriété (c'est-à-dire pour lesquelles la propriété est évaluée à V (voir §1.6.3)).

Un cas particulier, et plus simple, du problème de l'unification est celui du filtrage ("pattern matching")

Δ Etant donné deux termes t_1 et t_2 , le problème du *filtrage* de t_1 à partir de t_2 est de trouver la substitution σ , telle que $t_2 = \sigma t_1$.

On dit que t_1 *schématise* t_2 , et on note $t_1 \leq t_2$ (la relation \leq est un préordre) [68]. ∇

Dans le problème de l'unification, on se pose naturellement celui d'existence et d'unicité des solutions. Il est évident qu'il y a des équations qui n'ont pas de solution (exemple $a = b$). Mais on peut se demander ce qui se passe quand il en existe (au moins) une.

EXEMPLE:

$$x = f(g(y))$$

a comme solution

$$\sigma = \{ \langle x, f(g(a)) \rangle, \langle y, a \rangle \}$$

mais aussi

$$\sigma = \{ \langle x, f(g(b)) \rangle, \langle y, b \rangle \}$$

et d'une façon plus générale:

$$\sigma = \{ \langle x, f(g(y)) \rangle \}$$

Cette dernière peut engendrer toutes les solutions de l'équation (dans le cas qui nous occupe, une infinité): elles sont obtenues en remplaçant "y" par un terme "t" et en ajoutant le doublet " $\langle y, t \rangle$ " à σ .

On veut donc en général connaître non seulement si deux termes sont unifiables mais aussi l'ensemble de ses unificateurs (et évidemment la façon de les engendrer). Ceci peut être expliqué en introduisant un ordre parmi les solutions.

Δ Soit $t_1 = t_2$ une équation dans les termes et soit S l'ensemble de ses solutions.

Si $S \neq \emptyset$ alors on peut introduire un ordre parmi les éléments de S :
 $\sigma \in S, \theta \in S \quad \sigma \leq \theta$ ssi il existe une substitution $\lambda / \theta = \lambda \circ \sigma$ (" \leq est un préordre).

Le plus petit élément de S avec ce préordre s'appelle *l'unificateur le plus général* ou *l'unificateur principal* de t_1 et t_2 . ∇

Divers algorithmes ont été proposés pour le calcul de l'unificateur principal ([120] et [121], [123], [9], [68], [96], [105]).

L'algorithme d'unification est d'une importance capitale dans les démonstrateurs des théorèmes. Ses applications dépassent cependant largement le cadre de la démonstration automatique et parmi les domaines d'application nous pouvons citer: algèbre, calcul symbolique, langages de programmation, bases de données, systèmes de réécriture ([69], [119]).

Le mécanisme d'unification n'est pas restreint aux langages du premier ordre (voir par exemple [68]), mais pour nos besoins il est suffisant de considérer l'unification dans ces langages.

Nous exposons l'algorithme d'unification de Huet [68] qui est celui choisi pour l'implantation du démonstrateur (les critères qui nous ont amené à ce choix seront donnés au chapitre 4), et nous le faisons suivre d'un exemple d'application.

Nous donnons une description détaillée de l'algorithme, très proche d'un codage en Pascal.

2.1.2.2 ALGORITHME D'UNIFICATION POUR LA LOGIQUE DU PREMIER ORDRE

L'algorithme qui détecte l'existence de p.g.u. comporte deux étapes. La première étape construit des classes d'équivalence contenant les termes qui doivent être égaux. Si aucune impossibilité à cette construction ne se présente, la deuxième étape construit un graphe dont les nœuds sont les classes d'équivalence, puis il teste l'existence des cycles dans le graphe. Le p.g.u existe si le graphe ne contient pas de cycle. Si l'on veut le connaître, on peut le construire effectivement à partir des classes d'équivalence et de l'environnement.

Comme l'algorithme explore récursivement les sous-termes des termes pour construire les classes d'équivalence, il est pratique (mais non nécessaire) d'avoir une notation pour pouvoir décider si deux tels termes sont dans la même classe. L'algorithme commence donc avec un prétraitement qui réduit à 1 au plus la profondeur de tout sous-terme (voir § 1.6.1). Ce prétraitement crée un ensemble de *variables de travail* (notées w_1) et un *environnement* ρ qui est une substitution (en supposant que les variables de travail font partie de l'ensemble de variables).

EXEMPLE: Le terme $f(f(y,a), f(z,f(b,z)))$ sera noté après prétraitement w_1 , avec l'environnement:

$$\rho : \{ \langle w_1, f(w_2, w_3) \rangle, \langle w_2, f(y, w_4) \rangle, \langle w_3, f(z, w_5) \rangle, \langle w_4, a \rangle, \langle w_5, f(w_6 z) \rangle, \langle w_6, b \rangle \}$$

NOTATION: Dans ce qui suit, nous noterons:

x_i : variables apparaissant dans les termes à unifier

w_j : variables de travail

v_k : soit un x_i , soit un w_j (les v_k sont appelées "variables" dans l'algorithme)

\bar{v}_k : représentant de la classe d'équivalence à laquelle appartient v_k

$[\bar{v}_k]$: classe d'équivalence dont le représentant est \bar{v}_k .

Dans la description que nous faisons par la suite de l'algorithme, la fonction principale est UNIFIABLE, elle appelle EQCLASSES et NONCYCLE; ces deux dernières utilisent en commun l'ensemble de classes C^* et l'environnement ρ .

PRETRAIT et EQCLASSES utilisent en commun epvu (identificateur de l'ensemble de paires de variables à unifier).

Les données d'entrée sont deux termes à unifier, la sortie est la valeur *vrai* si les termes sont unifiables, et la valeur *faux* s'ils ne le sont pas.

La procédure PGU construit effectivement le p.g.u. dans le cas où il existe, en utilisant C^* et ρ .

fonction EQCLASSES: booléen;

* Elle construit les classes d'équivalence et délivre la valeur vrai ou faux selon qu'elle a pu ou non terminer cette étape de l'algorithme; l'ensemble de classes d'équivalence est noté C^* *

début

finir \leftarrow faux;

résultat \leftarrow vrai;

tantque non finir

faire

tantque epvu $\neq \emptyset$ et résultat * epvu initialisée dans PRETRAIT *

faire

choisir un $\langle v_1, v_1' \rangle \in \text{epvu}$;

$\langle v_1, v_2 \rangle \leftarrow \langle v_1, v_1' \rangle$;

epvu \leftarrow epvu - $\{ \langle v_1, v_1' \rangle \}$;

si $\bar{v}_1 \neq \bar{v}_2$ * c'est à dire si v_1' et v_2 n'étaient déjà dans la même classe, les classes d'équivalence sont initialisées dans UNIFIABLE *

alors

si v_1 et v_2 sont des variables de travail alors

* soient $\rho(v_1) = f(v_{1_1}, \dots, v_{1_n})$ et $\rho(v_2) = f'(v'_{j_1}, \dots, v'_{j_p})$ *

si $f \neq f'$ alors

finir \leftarrow vrai;

résultat \leftarrow faux

sinon * $p = n$ *

pour $k = 1$ jusqu'a n

faire

epvu \leftarrow epvu $\cup \{ \langle v_{1_k}, v'_{1_k} \rangle \}$

* v_{1_k} : i-ème argument de f , correspondant à $\rho(v_1)$;

v'_{1_k} : i-ème argument de f' , correspondant à $\rho(v_2)$ *

ffaire

fsi

fsi;

* Union de classes d'équivalence (comprend le cas où v_1 ou v_2 n'est pas une variable de travail);*

si non finir alors

si \bar{v}_1 est une variable de travail alors

$[\bar{v}_1] \leftarrow [\bar{v}_1] \cup [\bar{v}_2]$

$C^* \leftarrow C^* - [\bar{v}_2]$

sinon

$[\bar{v}_2] \leftarrow [\bar{v}_2] \cup [\bar{v}_1]$

$C^* \leftarrow C^* - [\bar{v}_1]$

fsi

fsi

ffaire

ffaire

finir \leftarrow vrai

ffaire

eqclasses \leftarrow resultat

fin * EQCLASSES * ;

La procédure suivante est destinée à refuser comme composant de substitution (§ 2.1.1.1) ceux de la forme $\langle x, f(\dots, x, \dots) \rangle$.

fonction NONCYCLE: booléen;

début

Construire un graphe G dont les nœuds appartiennent à C^* et tel qu'il y a un arc de $[\bar{v}_1]$ vers $[\bar{v}_2]$ ssi (a), (b) et (c) sont vérifiées:

(a) \bar{v}_1 est une variable de travail;

(b) $\rho(\bar{v}_1) = f(v_{i_1}, \dots, v_{i_n})$;

(c) $\exists k \quad 1 \leq k \leq n \quad / \quad v_{i_k} \in [\bar{v}_2]$

si G ne contient pas de cycle alors noncycle \leftarrow vrai

sinon noncycle \leftarrow faux;

fsi

fin * NONCYCLE * ;

fonction UNIFIABLE (t_1, t_2) : booléen;

* Elle détermine si t_1 et t_2 sont unifiables *

début

PRETRAIT (t_1, t_2) ; * réduction à 1 de la profondeur des sous-termes et initialisation de epvu avec $\langle w_1, w_2 \rangle$, où w_1 et w_2 sont les codes de t_1 et t_2 respectivement *

Pour toute variable v dans les termes ou dans l'environnement, construire la classe $[\bar{v}]$ avec $\bar{v} = v$; * initialisation des classes d'équivalence *

si EQCLASSES alors unifiable \leftarrow NONCYCLE

sinon unifiable \leftarrow faux

fsi

fin * UNIFIABLE * ;

Si l'on veut non seulement prouver l'existence du p.g.u. mais aussi le construire effectivement on peut employer la procédure PGU, qui utilise UNIFIABLE et SUBST. On définit d'abord SUBST.

fonction SUBST (v: variable): terme composant;

début

si \bar{v} n'est pas une variable de travail alors

subst \leftarrow \bar{v}

sinon

* Soit $\rho(\bar{v}) = f^n(v_1, \dots, v_n)$ *

si $n=0$ alors

subst \leftarrow $\rho(\bar{v})$ * v est une constante *

sinon

subst \leftarrow $f^n(\text{subst}(v_1), \dots, \text{subst}(v_n))$

fsi

fsi

fin * SUBST * ;

procedure PGU (t_1, t_2);

début

PGU \leftarrow \emptyset ;

si UNIFIABLE (t_1, t_2) alors

pour toute variable x_1 faire

PGU \leftarrow PGU \cup { $\langle x_1, \text{SUBST}(x_1) \rangle$ }

ffaire

sinon

écrire ('non unifiables')

fsi

fin * PGU * ;

L'existence d'un unificateur principal est une des propriétés fondamentales de la logique du premier ordre; elle est exprimée habituellement sous forme de théorème:

THEOREME: (d'unification): Etant donné l'équation dans les termes $t_1 = t_2$, il existe une procédure de décision pour l'existence d'une solution et un algorithme de calcul de l'unificateur principal. Cet unificateur principal est unique (à un renommage des variables près).

PREUVE: Dans [68]. Pour l'algorithme de Robinson, voir [27], [89], [120], [121]; pour les autres algorithmes, les références citées au § 2.1.2.1. ■

Nous donnons maintenant un exemple du fonctionnement de l'algorithme.

EXEMPLE: Soit l'équation

$$f(g(y), h(x, y)) = f(g(z), h(z, k(a, l(b))))$$

1. PRETRAIT transformera cette équation en :

$$w_1 = w_2$$

Avec l'environnement:

$$\rho = \{ \langle w_1, f(w_3, w_4) \rangle, \langle w_2, f(w_5, w_6) \rangle, \langle w_3, g(y) \rangle, \langle w_4, h(x, y) \rangle, \\ \langle w_5, g(z) \rangle, \langle w_6, h(z, w_7) \rangle, \langle w_7, k(w_8, w_9) \rangle, \langle w_8, a \rangle, \langle w_9, l(w_{10}) \rangle, \\ \langle w_{10}, b \rangle \}$$

Et elle initialisera epvu à:

$$\text{epvu} = \{ \langle w_1, w_2 \rangle \}$$

2. EQCLASSES donne les ensembles de classes d'équivalence suivants (nous donnons aussi les états de l'ensemble de paires de variables à unifier (epvu)).

Etats de l'ensemble de paires
de variables à unifier:

Ensemble de classes d'équivalence:

$$C_0^* = \{ \{ \bar{w}_1 \}, \{ \bar{w}_2 \}, \dots, \{ \bar{w}_{10} \}, \{ \bar{x} \}, \{ \bar{y} \}, \{ \bar{z} \} \}$$

$$\text{epvu} = \{ \langle w_1, w_2 \rangle \}$$

$$C_1^* = \{ \{ \bar{w}_1, w_2 \} \} \cup C_0^* - \{ \{ \bar{w}_1 \}, \{ \bar{w}_2 \} \}$$

$$\text{epvu} = \{ \langle w_3, w_5 \rangle, \langle w_4, w_6 \rangle \}$$

$$C_2^* = C_1^* \cup \{ \{ \bar{w}_3, w_5 \} \} - \{ \{ \bar{w}_3 \}, \{ \bar{w}_5 \} \}$$

$$\text{epvu} = \{ \langle w_4, w_6 \rangle, \langle y, z \rangle \}$$

$$C_3^* = C_2^* \cup \{ \{ \bar{w}_4, w_6 \} \} - \{ \{ \bar{w}_4 \}, \{ \bar{w}_6 \} \}$$

$$\text{epvu} = \{ \langle y, z \rangle, \langle x, z \rangle, \langle y, w_7 \rangle \}$$

$$C_4^* = C_3^* \cup \{ \{ \bar{y}, z \} \} - \{ \{ \bar{y} \}, \{ \bar{z} \} \}$$

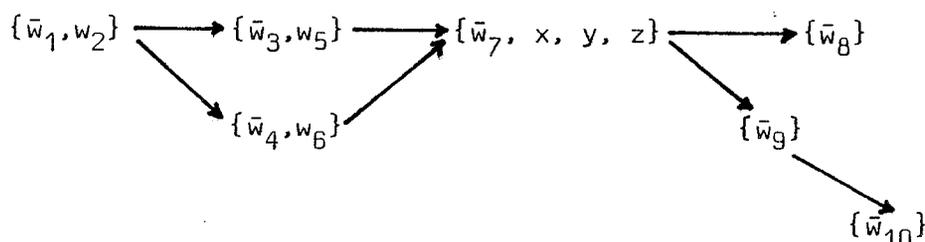
$$C_5^* = C_4^* \cup \{ \{ \bar{y}, x, z \} \} - \{ \{ \bar{y}, z \}, \{ \bar{x} \} \}$$

$$C_6^* = C_5^* \cup \{ \{ \bar{w}_7, y, x, z \} \} - \{ \{ \bar{y}, x, z \}, \{ \bar{w}_7 \} \}$$

$$C^* \text{ final} = \{ \{ \bar{w}_1, w_2 \}, \{ \bar{w}_3, w_5 \}, \{ \bar{w}_4, w_6 \}, \{ \bar{w}_7, y, x, z \}, \{ \bar{w}_8 \}, \{ \bar{w}_9 \}, \{ \bar{w}_{10} \} \}$$

Valeur retournée par EQCLASSES: *vrai*

3. NONCYCLE construit le graphe suivant:



Qui n'a pas de cycle, donc:

Valeur retournée par NONCYCLE: *vrai*

Valeur retournée par UNIFIABLE: *vrai*

4. PGU construit l'unificateur le plus général;

$$\sigma = \{ \langle x, k(a, l(b)) \rangle, \langle y, k(a, l(b)) \rangle, \langle z, k(a, l(b)) \rangle \}$$

2.2 QUELQUES METHODES DE PREUVE ET CONCEPTS ASSOCIÉS

2.2.1 CORRECTION ("SOUNDNESS") ET COMPLETUDE DES METHODES DE PREUVE. CORRECTION DES REGLES D'INFERENCE.

Le problème fondamental de la recherche automatique de la preuve de la validité ou de l'insatisfaisabilité d'une fbf par une méthode quelconque, est le suivant:

"Utiliser une procédure, qui pour *toute* fbf valide (insatisfaisable) - et *seulement* dans ce cas - trouvera que la fbf est valide (insatisfaisable)".

S'il s'agit d'une méthode basée sur le théorème de Herbrand, alors on dira:

"Utiliser une procédure, qui pour *tout* ensemble insatisfaisable de clauses - et *seulement* dans ce cas - trouvera un ensemble fini de c-clauses insatisfaisables".

La terminologie utilisée pour exprimer ces notions est la suivante:

Δ Soit:

P la proposition: "La procédure classifie comme valide (insatisfaisable) une fbf",

et Q la proposition: "La fbf testée est valide (insatisfaisable)".

Une méthode qui satisfait:

(1) si P alors Q

est dite *correcte* ("sound" ou "correct" [75]).

Une méthode qui satisfait:

(2) si Q alors P

est dite *complète*. ∇

On s'intéresse en général à des méthodes *correctes et complètes*.

Dans les systèmes corrects et complets, la notion sémantique d'insatisfaisabilité (validité) coïncide avec la notion syntaxique de réfutation (démonstration).

Jusqu'ici les notions de théorie formelle et de règle d'inférence étaient liées à un traitement syntaxique de la logique.

Les méthodes de preuve testant l'insatisfaisabilité d'un ensemble de clauses, bien que fondées sur une notion *sémantique* (le théorème de Herbrand) font un traitement *syntaxique* des clauses: elles manipulent des formules sans avoir besoin d'attribuer un sens quelconque aux symboles qui la composent, ni d'effectuer des tests de vérité. C'est pourquoi à partir de maintenant nous parlerons

de règles d'inférence (et de théorie formelle), pour des méthodes basées sur le théorème de Herbrand.

Si l'on veut mettre en rapport les notions d'insatisfaisabilité et de réfutation il est naturel d'exiger des règles d'inférence qu'elles vérifient certaines propriétés:

Δ On dit d'une règle d'inférence qu'elle est *correcte* ("sound") ssi elle transforme des fbf valides dans des fbf valides. ∇

EXEMPLE: Le "modus ponens" est une règle correcte: Si A et $A \supset B$ sont des tautologies, alors il est facile de vérifier par application des tables de vérité que B est aussi une tautologie.

2.2.2 ESPACE DE RECHERCHE, STRATEGIE, HEURISTIQUE.

Nous examinons brièvement ces trois concepts fondamentaux. Nous en faisons une présentation orientée à la démonstration automatique, mais son extension aux autres domaines de l'Intelligence Artificielle est immédiate.

Le concept d'espace de recherche que nous définissons en premier correspond à celui d'ensemble de toutes les conséquences (§ 1.3).

Si P est une procédure de preuve, Γ l'ensemble des règles d'inférence de P et $\Gamma(S)$, $\Gamma^2(S)$, $\Gamma^3(S)$, ... le résultat de l'application de toutes les règles d'inférence de Γ respectivement à S, $\Gamma(S)$, $\Gamma^2(S)$, ... alors:

Δ Etant donné un ensemble de clauses S, *l'espace de recherche de S*, noté $\Gamma(S)$, est défini comme suit:

$$\Gamma(S) = \bigcup_{i \in I} \Gamma^i(S) \quad I = 0, 1, 2, \dots, n, \dots \quad (\Gamma^0(S) = S). \nabla$$

REMARQUE 1: L'espace de recherche de S contient toutes les réfutations de S.

REMARQUE 2: L'espace de recherche d'un ensemble de clauses peut être fini ou infini.

REMARQUE 3: L'espace de recherche est généralement représenté comme un arbre (ou un graphe). C'est pourquoi on parle indistinctement "d'espace de recherche" ou "d'arbre de recherche".

Si l'on examine les nœuds de l'arbre de recherche d'une façon aveugle (exhaustive) alors on peut voir intuitivement que cette recherche n'est pas adéquate pour la démonstration de théorèmes complexes (une recherche exhaustive, d'ailleurs, sera inévitablement faite en temps exponentiel).

On est donc amené à essayer d'éviter cette "explosion combinatoire".

Ainsi une fois la notion d'espace de recherche introduite, la question se pose de *comment* l'explorer (ou ce qui est équivalent comment le construire) d'une façon systématique (*non exhaustive*). Ceci conduit aux autres deux con-

cepts; celui de stratégie et celui d'heuristique. Ces deux concepts étant une réponse à la question: "Est-il possible de trouver une réfutation en explorant (en construisant) un sous-arbre de l'arbre de recherche?".

Il est classique de considérer deux possibilités de parcours (construction) "extrêmes":

1) En largeur d'abord ("breadth-first"): explorer tous les nœuds d'un niveau de l'arbre avant d'explorer tous les nœuds du niveau suivant.

On suppose ici, évidemment, que tout nœud dans l'arbre est racine d'un nombre fini de sous-arbres.

2) En profondeur d'abord ("depth-first"): explorer une seule branche de l'arbre. Si l'on arrive à une impasse on est obligé de faire un retour arrière ("backtrack").

Nous empruntons les deux définitions suivantes à Loveland [89] (un traitement plus abstrait de la notion de stratégie peut être trouvé dans [77]).

Δ Une *restriction* d'une procédure P est une procédure qui engendre un sous-ensemble des déductions de P pour tous les ensembles de clauses et un *sous-ensemble propre* pour au moins un ensemble de clauses. ∇

Δ Un *raffinement* d'une procédure P est soit une restriction de P , soit une procédure qui réordonne l'ordre de développement des déductions associées à P . Un raffinement qui réordonne l'ordre de développement est appelé une *stratégie*. ∇

Une stratégie de parcours étant fixée, on veut s'assurer qu'en l'appliquant à un ensemble de clauses insatisfaisable on trouve effectivement une réfutation pour cet ensemble; on introduit donc, en accord avec la définition de complétude de § 2.2.1, la définition de complétude pour une stratégie.

Δ Une *stratégie* pour une méthode de preuve est dite *complète* ssi en l'appliquant à tout ensemble de clauses S insatisfaisable on trouve une réfutation de S . ∇

Une autre façon d'éviter l'explosion combinatoire est d'utiliser des heuristiques (heuristique = "aide à la découverte"). Les heuristiques sont des critères (en général empiriques) qui permettent d'explorer un sous-arbre de l'arbre de recherche.

A différence des stratégies complètes, une heuristique ne garantit pas qu'en l'appliquant à un ensemble de clauses insatisfaisable on en trouvera une réfutation. Cependant, l'utilisation des heuristiques peut être très utile dans la démonstration de théorèmes compliqués.

2.2.3 METHODES DE SATURATION

Les méthodes ainsi appelées dans [121] et [124] appliquent directement le théorème de Herbrand (§ 1.7.2). La recherche d'un ensemble de c-clauses insatisfaisable est faite à partir d'une *énumération* de l'univers de Herbrand correspondant aux clauses dont on veut montrer l'insatisfaisabilité.

Nous donnons d'abord quelques définitions qui nous aideront à formaliser l'exposé.

Δ L'ensemble de variables d'un terme t , noté $EV(t)$, est défini récursivement de la façon suivante:

$$EV(x) = \{x\} \quad \forall x \in V$$

$$EV(a) = \emptyset \quad \forall a \in F$$

$$EV(f^n(t_1, \dots, t_n)) = \bigcup_{i=1}^n EV(t_i) \quad \forall f^n \in F, \forall$$

Δ L'ensemble des variables d'un littéral, d'une clause et d'un ensemble de clauses est défini de la façon suivante. Pour:

a) un littéral

. si $L: P$ (symbole propositionnel)

$$EV(L) = \emptyset$$

. si $L: P(t_1, \dots, t_n)$ ou $L: \neg P(t_1, \dots, t_n)$ $\forall P \in P$

$$EV(L) = \bigcup_{i=1}^n EV(t_i)$$

b) une clause

$$C: \{L_1, \dots, L_n\}$$

$$EV(C) = \bigcup_{i=1}^n EV(L_i)$$

c) un ensemble de clauses

$$S: \{C_1, \dots, C_n, \dots\}$$

$$EV(S) = \bigcup_{i \in I} EV(C_i) \quad I = 1, 2, \dots, n, \dots \quad \forall$$

NOTATION: Soit V un ensemble de variables ($V \subseteq V$), nous noterons σ_V la restriction de la substitution σ à V . Quand nous voudrions mettre en évidence que P ($P \subseteq T$) est le codomaine de σ , nous écrirons σ_V^P .

En particulier, pour un ensemble de clauses S , nous nous intéresserons à des sous-ensembles de T qui sont aussi des sous-ensembles de $H(S)$ (univers de Herbrand de S).

Δ Soit $\sigma_{EV(S)}^P$, si $P \subseteq H(S)$ nous dirons que $\sigma_{EV(S)}^P$ est une *h-substitution* pour l'ensemble de variables $EV(S)$. \forall

Une h -substitution est donc une substitution sans variables (§ 2.1.1.1).

L'ensemble de toutes les h -substitutions $\sigma_{EV(S)}^P$ sera noté $\Sigma_{EV(S)}^P$.

Si P est fini et si l'on considère un ensemble fini de clauses (ou un ensemble de clauses avec un nombre fini de variables différentes), alors:

$$\text{card} \left(\Sigma_{EV(S)}^P \right) = (\text{card } P)^{\text{card} (EV(S))}$$

Nous pouvons définir la saturation d'un ensemble de clauses par un ensemble de termes (voir § 1.7.2), en utilisant les notions définies dans ce chapitre:

Δ La *saturation* d'un ensemble de clauses S par un ensemble de termes P ($P \subseteq H(S)$) est définie comme suit:

$$\text{SAT}_P(S) = \{ \sigma \circ S \quad / \quad \sigma \in \Sigma_{EV(S)}^P \}$$

Nous voyons maintenant les procédures de Gilmore et de Davis et Putnam, qui sont des *procédures de saturation*.

Le principe de ces méthodes est très simple:

" Pour tester l'insatisfaisabilité d'un ensemble (fini) de clauses S , on énumère les substitutions de $\Sigma_{EV(S)}^{H(S)} : \sigma_1, \sigma_2, \dots, \sigma_1, \dots$ et l'on évalue avec les règles du calcul propositionnel la valeur de vérité de $\sigma_1 S, \sigma_1 S \wedge \sigma_2 S, \dots, \sigma_1 S \wedge \sigma_2 S \wedge \dots \wedge \sigma_n S$ jusqu'à ce que le résultat de l'évaluation soit F , ou que l'on décide d'arrêter".

2.2.3.1 LA METHODE DE GILMORE

Elle correspond à une application directe du théorème de Herbrand. Le programme, présenté dans [55] fournit aussi une procédure de décision pour une sous-classe de formules du Calcul de Prédicats. Bien que les formules acceptées par le programme de Gilmore ne soient pas sous forme clausale, nous pouvons considérer qu'elles le sont sans rien changer à la méthode. Nous appelons S l'ensemble de clauses dont on veut tester l'insatisfaisabilité.

On peut décrire la méthode comme suit:

procédure GILMORE;

début

donner une façon d'énumérer $\Sigma_{EV(S)}^{H(S)}: \sigma_1, \sigma_2, \dots;$

théorème \leftarrow faux;

non-théorème \leftarrow faux;

arrêt \leftarrow faux; * représente un critère d'arrêt quelconque, autre que la propriété de l'ensemble de clauses d'être ou de ne pas être un théorème *

$i \leftarrow 1;$

$P_0 \leftarrow$ vrai

tantque (non théorème et non arrêt)

faire

$P_i \leftarrow \sigma_i S \wedge P_{i-1};$

(1) développer la forme normale disjonctive de P_i ;

(2) évaluer (avec les règles du Calcul Propositionnel) P_i ;

si $P_i =$ faux alors théorème \leftarrow vrai

sinon

(3) si procédure de décision applicable alors

non-théorème \leftarrow vrai

arrêt \leftarrow vrai

sinon

si critère d'arrêt se vérifie alors

arrêt \leftarrow vrai

ssi;

ssi;

ssi;

$i \leftarrow i + 1;$

ffaire;

fin *GILMORE* ;

Certaines optimisations (utilisant essentiellement un codage) sont décrites dans [55].

L'inefficacité de cette méthode, due à l'explosion du nombre de disjoints engendrés, a fait que l'on ne s'intéresse à elle que comme référence historique. Cependant dans un travail récent [62], une méthode est présentée qui est une reformulation de l'approche de Gilmore, et qui lui est équivalente pour le cas propositionnel.

La correction et la complétude de cette méthode découlent de l'énoncé du théorème de Herbrand.

La méthode suivante évite le développement sous forme normale disjonctive de la formule, en utilisant des règles qui permettent de *simplifier l'évaluation de la valeur de vérité* des ensembles de clauses testés.

2.2.3.2 LA METHODE DE DAVIS ET PUTNAM

Elle comporte les 5 règles suivantes (dans l'article où elle a été publiée pour la première fois [42], elle ne comportait que les règles 1, 2 et 3).

Appliquées à un ensemble (non vide) de c -clauses S , elles permettent selon les cas de :

- a) décider si S est insatisfaisable.
- b) décider si S est satisfaisable.
- c) transformer S dans un autre ensemble S' (ou dans 2 autres ensembles S'' et S'''), en préservant la satisfaisabilité.

La règle 0 est appliquée une seule fois avant de commencer la méthode proprement dite. Il est facile de voir que d'autres tautologies ne peuvent pas être engendrées par l'application de la méthode.

Pour les règles 0, 1b, 2, 3: si l'ensemble S' est vide, alors S est satisfaisable sinon S est insatisfaisable ssi S' l'est.

Pour la règle 4: S est insatisfaisable ssi S'' et S''' le sont.

REGLE 0:

Éliminer les clauses qui contiennent des tautologies.

REGLE 1:

a) Si S contient deux clauses unitaires avec des littéraux complémentaires, alors S est insatisfaisable.

b) (règle de la clause unitaire): Si a) ne s'applique pas et si S contient une clause unitaire $\{L\}$ (L : positif ou négatif), éliminer toutes les clauses contenant L et éliminer toutes les occurrences de L^c dans les clauses restantes.

REGLE 2 (du littéral pur):

Si un littéral est présent mais son complémentaire ne l'est pas, alors toutes les clauses le contenant peuvent être éliminées.

REGLE 3 (de la division):

S'il existe des clauses non unitaires contenant des littéraux complémentaires (disons L et L'), remplacer S par les ensembles S'' et S''' . S'' est un ensemble où toutes les clauses contenant L ont été éliminées ainsi que toutes les occurrences de L' . S''' est un ensemble de clauses, où toutes les clauses contenant L' ont été éliminées ainsi que toutes les occurrences de L .

REGLE 4 (de subsumption):

Éliminer de S toute clause contenue dans une autre clause.

Δ Un littéral d'une clause qui n'a pas de complémentaire dans les autres clauses de l'ensemble est appelé un *littéral pur*. ▽

La règle 2 est souvent appelée "*principe de pureté*" [121].

Ces règles donnent une procédure correcte et complète ([27],[42],[89]).

L'algorithme de preuve utilisant la méthode de Davis et Putnam peut être obtenu en remplaçant dans la procédure Gilmore (§ 2.2.3.1) la ligne (1) par: (1'): appliquer les règles 1,2,3 et 4, dans cet ordre, itérativement et jusqu'à ce que chaque règle ne soit plus applicable.

Et la ligne (2) par:

(2') *si* l'on ne peut pas conclure à l'insatisfaisabilité (ou la satisfaisabilité) de P_i

alors évaluer P_i (avec les règles du Calcul Propositionnel);

La ligne (3) correspond à l'obtention d'un ensemble vide.

La règle 3 remplace un ensemble de clauses par deux autres ensembles et son application cause une croissance exponentielle des ensembles de clauses à tester

On voit que l'on *simplifie* le test de contradiction, mais on continue à tester d'une façon *exhaustive* (par énumération) les ensembles de c-clauses.

Plus récemment, Shostak [135] a donné une façon d'incorporer l'unification à cette méthode, évitant ainsi l'explosion combinatoire de clauses engendrées. On peut aussi trouver dans [27] une généralisation de la méthode de Davis et Putnam à des ensembles de clauses (c'est-à-dire qu'on n'exige plus de c-clauses) ainsi qu'une généralisation de la règle de division.

2.2.4 LES ARBRES SEMANTIQUES

Ils ont été présentés par Robinson [122], modifiés et utilisés dans [79]. Cette façon d'organiser la recherche d'un ensemble fini de c-clauses insatisfaisable apporte des éclaircissements sur la notion de preuve.

CONVENTION PRATIQUE: Dans le reste de ce chapitre et dans les chapitres suivants nous écrirons indistinctement $\neg P(\dots)$ ou $\bar{P}(\dots)$ au lieu de $\neg P(\dots)$.

Δ Soit S un ensemble de clauses, un *arbre sémantique associé à S* est une arborescence binaire, orientée, et où les arcs sont étiquetés par des atomes (ou la négation des atomes) de la base de Herbrand de S . L'étiquetage est fait de la façon suivante:

- . On donne une façon d'énumérer la base de Herbrand de S : $(A_1, A_2, \dots, A_n, \dots)$
- . Pour un nœud au niveau i ($i \geq 1$), son fils gauche est étiqueté A_i et son fils droit $\neg A_i$ ($i = 1$ correspond à la racine). ▽

REMARQUE: Dans un arbre sémantique, un chemin issu de la racine ne peut donc pas contenir deux littéraux complémentaires.

Il est évident que plusieurs arbres sémantiques peuvent être associés à un même ensemble de clauses S , dépendant de l'énumération faite de la base de Herbrand de S .

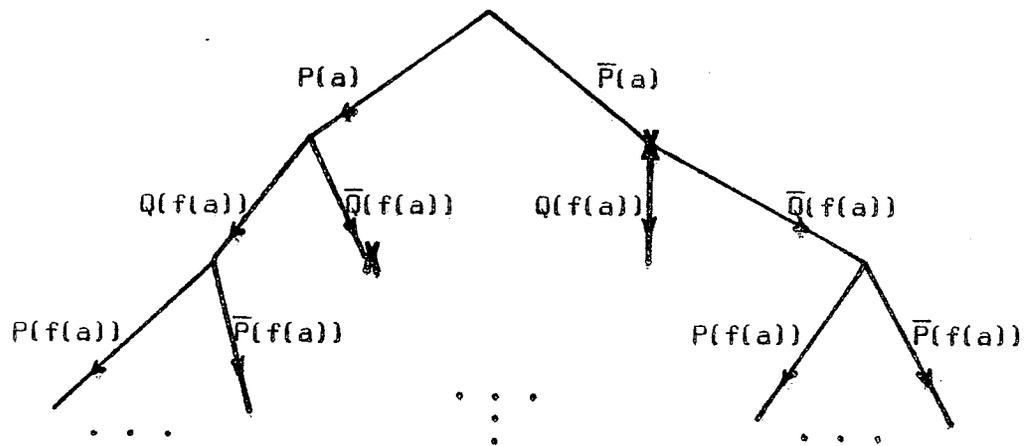
EXEMPLE:

$$S = \{ P(x), Q(f(x)) \}$$

$$H(S) = \{ a, f(a), f(f(a)), \dots \}$$

$$B(S) = \{ P(a), Q(f(a)), P(f(a)), Q(f(f(a))), \dots \}$$

L'arbre sémantique correspondant à cette énumération est :



On parle habituellement de "l'arbre sémantique de S ". Comme l'on sait, en général il y en a plusieurs, leur différence résidant dans l'ordre d'étiquetage des arcs. Cette différence n'a aucune importance dans le lemme suivant.

LEMME: Un ensemble de clauses S est satisfaisable ssi l'arbre sémantique pour S contient un chemin issu de la racine dont l'ensemble d'étiquettes est un c -modèle.

PREUVE:

- 1) La partie "si" du lemme découle directement de la définition de c -modèle (§ 1.7.2) et du premier lemme du § 1.7.2.
- 2) La partie "seulement si" peut être prouvée de la façon suivante: d'après le premier lemme du § 1.7.2 $SAT_{H(S)}(S)$ a un c -modèle; il faut montrer que ce c -modèle coïncide avec l'ensemble d'étiquettes d'un chemin issu de la racine.

On peut construire ce chemin en étiquetant les arcs avec les c -littéraux du c -modèle et "en complétant" l'étiquetage avec les c -littéraux éventuellement manquant dans le c -modèle (il est évident que cette construction respecte la définition d'arbre sémantique). ■

On a trouvé un c -modèle pour un ensemble de clauses S , quand un chemin issu de la racine a ses arcs étiquetés avec au moins une instance constante d'un littéral de chaque clause. Pour qu'une branche de l'arbre ne soit pas un c -modèle,

il faut alors qu'elle contienne un c-littéral complémentaire des instances constantes de chaque littéral d'une clause. D'après la définition d'arbre sémantique cette branche évaluera toujours à **F** la clause en question.

Δ Une branche qui ne peut pas être un c-modèle s'appelle une *branche d'échec*, et le nœud de la branche le moins profond dans l'arbre pour lequel cette propriété est vraie s'appelle un *nœud d'échec*. ▽

Exemple de nœuds d'échec: ceux repérés par une "X" dans le dessin ci-dessus. Si S est insatisfaisable, il n'existe pas de c-modèle pour S , chaque branche de l'arbre sémantique associé contient donc un nœud d'échec, un tel arbre s'appelle un *arbre sémantique fermé*.

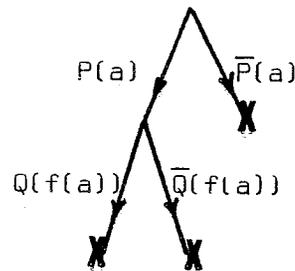
EXEMPLE:

Soit $S = \{ P(x), -P(x) \vee Q(f(x)), -Q(f(a)) \}$

la base de Herbrand de S :

$B(S) = \{ P(a), Q(f(a)), P(f(a)), Q(f(f(a))), \dots \}$

le suivant est un arbre fermé pour S :



Nous pouvons donner maintenant une autre version du théorème de Herbrand:

THEOREME (de Herbrand): *Un ensemble fini de clauses est insatisfaisable ssi tout arbre sémantique associé à S est fermé.*

PREUVE: Découle immédiatement de la version du théorème donnée au Chapitre 1.

Une autre preuve utilisant le lemme de König peut-être trouvée dans [27]. ■

Il est clair que dans un arbre sémantique toute branche issue de la racine est une interprétation de $SAT_H(S)(S)$.

Ici, il est utile de considérer la question suivante: "Quel est le rapport entre la notion (sémantique) d'interprétation définie au § 1.6.3 et celle considérée dans les arbres sémantiques?". (La deuxième notion est d'ailleurs implicite dans le théorème de Herbrand).

Dans [27] on peut trouver une définition précise de ce que l'on appelle "h-interprétations": interprétations dont le domaine est l'univers de Herbrand, et où une façon équivalente aux arbres sémantiques pour définir l'extension des prédicats est utilisée.

Il y est montré qu'a toute interprétation on peut faire correspondre une h-interprétation tel que si la première satisfait un ensemble de clauses S alors

la deuxième satisfait aussi S.

2.2.5 ENSEMBLE DE PREUVE

Pour un ensemble fini de clauses insatisfaisable S, il serait intéressant de disposer d'un ensemble fini $P (P \subseteq H(S))$, tel que l'on soit assuré (par exemple par un oracle) qu'il suffirait d'utiliser des substitutions dans $\Sigma_{EV(S)}^P$ pour obtenir un ensemble de c-clauses insatisfaisable (ceci limiterait la croissance explosive de c-clauses engendrées par énumération). Cette idée a été utilisée dans un démonstrateur [124] où l'utilisateur jouait le rôle de l'oracle. Nous citons ce travail parce qu'il a été l'un des premiers à notre connaissance à reconnaître les possibilités de l'interaction utilisateur-programme.

Les définitions suivantes formalisent la notion évoquée ci-dessus:

Δ Si S est un ensemble de clauses, $P \subseteq H(S)$, P fini, est un ensemble d'insatisfaisabilité pour S ssi $SAT_P(S)$ est insatisfaisable. ∇

Δ Si P est minimal, c'est-à-dire tout autre ensemble d'insatisfaisabilité P' est tel que $P' \supseteq P$, alors P est dit un ensemble de preuve. ∇

Soit un ensemble de clauses $S = \{C_1, C_2, \dots, C_n\}$; si l'on suppose

- a) l'ensemble de preuve P connu
- b) $\text{card}(EV(C_i)) = c_i \quad 1 \leq i \leq n$
- c) $\text{card}(P) = p$

alors le nombre total d'instances constantes (avec des termes dans P) d'une clause $C_i (1 \leq i \leq n)$ est égal au nombre total d'applications possibles de $EV(C_i)$ dans P; c'est-à-dire $p^{c_i} (1 \leq i \leq n)$, et le nombre total de clauses constantes que l'on peut engendrer avant de trouver un ensemble fini de c-clauses insatisfaisable peut atteindre:

$$\sum_{i=1}^n p^{c_i}$$

Ceci permet de se faire une idée des limites pratiques des procédures de saturation.

REMARQUE: Soit un ensemble fini de clauses S, on peut engendrer $SAT_{H(S)}(S)$ de la façon suivante: on commence par S et on y ajoute successivement des variantes de clauses de S (ces variantes ne contenant pas de variable dans S). Obtenant $S_1, S_2, \dots, S_n, \dots$, on applique des substitutions dans $\Sigma_{EV(S)}^{H(S)}, \Sigma_{EV(S_1-S)}^{H(S)}$,

$$\Sigma_{EV(S_2-S_1)}^{H(S)}, \dots, \Sigma_{EV(S_n-S_{n-1})}^{H(S)}, \dots$$

Cette façon de faire pour la recherche d'un ensemble fini de c-clauses insatisfaisable est la plus répandue, et quand on a réussi à obtenir un tel ensemble on

dit que l'on a utilisé, par exemple, i_1 variantes de C_1 , i_2 variantes de C_2, \dots, i_n variantes de C_n .

En correspondance avec cette idée, nous pouvons introduire une notion bien plus forte que celle d'ensemble de preuve; nous l'utiliserons au Chapitre 3:

Δ Soit S un ensemble de clauses insatisfaisable, un ensemble de clauses T est dit un *ensemble S-insatisfaisable par substitution* ssi:

- i) $\forall C_i \in T$ soit $C_i \in S$, soit $\exists C_j \in S / C_i$ est une variante de C_j .
- ii) Si $C_i, C_j \in T$ alors $EV(C_i) \cap EV(C_j) = \phi \quad \forall i, j$
- iii) $\exists \sigma \in \Sigma_{EV(T)}^{H(S)} / \sigma \circ T$ est contradictoire.

σ est dite une *substitution de réfutation* pour T . ∇

Il est facile de vérifier que ce qui est *important* pour prouver l'insatisfaisabilité est l'existence, dans des clauses différentes, de littéraux qui peuvent être rendus *complémentaires* par instantiation. En 1960 Prawitz [110] suggéra qu'au lieu d'engendrer des c-clauses par énumération, on engendre seulement les c-clauses correspondant à des clauses qui contiendraient (après l'instanciation correspondante de la clause) un littéral complémentaire dans une autre clause. Ceci permettrait, par exemple dans le développement sous forme normale disjonctive pour évaluer la formule, d'évaluer à F le disjoint contenant ces deux littéraux complémentaires. Cette idée très simple fut un pas en avant très important pour rendre réalisable en pratique les méthodes de démonstration automatique.

Cette idée est le principe fondamental de la méthode du "linked conjunct" [41] que nous traduirons par *paires liées*. Ce sera la dernière des méthodes ayant précédé la résolution; dont nous parlerons dans ce chapitre.

2.2.6 LA METHODE DES PAIRES LIEES ("LINKED CONJUNCT")

Cette méthode utilise l'idée de Prawitz citée ci-dessus, et elle évalue la valeur de vérité d'une formule sous forme clausale à l'aide des règles de Davis et Putnam.

Δ Un ensemble de c-clauses $S = \{ C_1, C_2, \dots, C_n \}$ est dit un *ensemble de paires liées*, ssi pour tout littéral L dans une clause C_i ($1 \leq i \leq n$) il existe un littéral complémentaire L^c dans une clause C_j ($1 \leq j \leq n$) $i \neq j$.

Autrement dit: si aucune clause ne contient un littéral pur.

L'occurrence de $L^c(L)$ est dite un *associé* ("mate") pour l'occurrence de $L(L^c)$. ∇

La même idée sous une forme légèrement différente a été utilisée dans un travail plus récent [4].

THEOREME: Si $S = \{ C_1, \dots, C_n \}$ est un ensemble de c -clauses insatisfaisable alors il existe S' insatisfaisable, tel que $S' \subseteq S$ et S' est un ensemble de paires liées.

PREUVE:

Si S est un ensemble de paires liées, il n'y a rien à démontrer.

Si S contient une clause avec un littéral pur L_j , alors nous allons prouver qu'elle peut être éliminée en préservant l'insatisfaisabilité:

Supposons $L_j \in C_i$, L_j pur

$C_i: \{ L_1, \dots, L_j, \dots, L_{1+p} \}$

nous construisons $S' = S - \{ C_i \}$

(nous supposons que m est le nombre de littéraux de S , donc $n = m-p$ est le nombre de littéraux de S'),

Supposons S' satisfaisable, alors S' a un modèle $M = \{ L_1, L_2, \dots, L_n \}$ où $L_i \neq L_j$ $1 \leq i \leq n$

Puisqu'une clause est évaluée à V si l'un de ses littéraux est évalué à V , nous pouvons construire un modèle pour S' en utilisant M :

$$S' = M \cup \{ L_j, L_1, \dots, L_{1+p} \}$$

La valeur de vérité assignée à L_1, \dots, L_{1+p} est indifférente.

Donc S' est satisfaisable, contredisant l'hypothèse. S' est donc insatisfaisable.

Le même raisonnement se généralise évidemment à toutes les clauses contenant un littéral pur.

S' ne peut pas être vide, parce que ceci signifierait que toutes les clauses contiennent au moins un littéral pur, donc on pourrait construire un modèle pour S qui contiendrait tous ces littéraux purs (évalués à V), ce qui contredit l'hypothèse d'insatisfaisabilité de S .

Cette preuve est aussi la justification de la règle 2 de Davis et Putnam (principe de pureté). ■

Davis propose alors de tester (à l'aide des règles de Davis et Putnam) l'insatisfaisabilité *seulement* quand on a obtenu un ensemble de paires liées.

L'idée de ne travailler qu'avec des ensembles de paires liées a été utilisée ultérieurement dans la méthode des graphes de connexion [75], [76].

Dans la méthode des graphes de connexion (dont nous parlerons aussi au chapitre 4) les clauses sont gardées dans un graphe, où les nœuds d'un même arc sont des littéraux qui peuvent être rendus complémentaires (qui sont des "associés") et chaque arc est étiqueté par le p.g.u des arguments des littéraux reliés par l'arc. Les clauses contenant des littéraux purs sont éliminées, ainsi que tous les arcs qui y avaient leurs origines (ce processus est répété jusqu'à ce qu'il n'y ait

plus de littéraux purs dans les clauses).

La méthode des paires liées a été implantée bien après sa publication au Courant Institute [162] (une implantation non publiée y est citée), l'implantation ayant comme but, entre autres, une comparaison avec les méthodes de résolution [121] et de "model-élimination" [86],[89]. Dans la présentation qui est faite de la méthode dans [41], il n'y a aucune indication (sauf peut-être des remarques heuristiques) sur la manière d'obtenir un nouvel ensemble de paires liées à partir d'un autre. Dans [162], des règles d'inférence sont données pour ce faire (elles utilisent l'algorithmique d'unification).

La méthode utilisée dans l'implantation a deux phases:

- 1) Phase de recherche (d'un ensemble de paires liées).
- 2) Phase de test d'insatisfaisabilité (avec les règles de Davis et Putnam; d'autres règles pourraient évidemment être appliquées).

Si le test d'insatisfaisabilité donne une réponse négative, on cherche un autre ensemble de paires liées que l'on teste à nouveau.

2.2.7 LA METHODE BASEE SUR LE PRINCIPE DE RESOLUTION

En considérant quelques exemples de théorèmes relativement simples à montrer (voir par ex.[124]) et l'évaluation faite au § 2.2.6 du nombre de clauses que l'on peut engendrer avant de trouver une réfutation, le besoin de principes de recherche plus efficaces que ceux présentés jusqu'ici apparaît clairement. Cette plus grande efficacité et une présentation très élégante de la logique du premier ordre sont contenues dans le principe de résolution. Ce principe publié en 1965 [121] fut à l'origine de très forts espoirs sur les possibilités de la mécanisation de la démonstration de théorèmes, et la plupart des démonstrateurs réalisés jusqu'à aujourd'hui sont basés sur lui. Nous présentons le principe de résolution tel qu'il a été exposé par Robinson; beaucoup de raffinements ont été conçus depuis son apparition (voir par ex:[27],[89]).

Il est classique de considérer d'abord la règle de résolution pour des ensembles de c-clauses, puis la résolution pour les ensembles de clauses en général.

2.2.7.1 LA RESOLUTION POUR LES ENSEMBLES DE C-CLAUSES

Dans ce cas elle n'apporte pas de nouvelle technique, puisqu'elle utilise un outil classique de la logique: la règle de *coupure* des systèmes de Gentzen ([54],[73],[100]). Elle est aussi le *dual* de la *règle du consensus* [114] et une extension de la *règle de la clause unitaire* de Davis et Putnam (règle 1b).

Nous présentons brièvement les séquences de Gentzen et la règle de coupure:

Δ Une *séquence* est une expression de la forme:

$$\Gamma \rightarrow \Delta \quad (\Gamma \text{ s'appelle l'antécédent et } \Delta \text{ le conséquent})$$

où Γ et Δ sont des ensembles de fbf.

Quand Γ et Δ sont finis, $\Gamma \rightarrow \Delta$ est appelée une *séquence finie* (seules les séquences finies nous intéressent ici). \forall

$$\text{Soit } \Gamma = \{ A_1, A_2, \dots, A_m \} \quad \Delta = \{ B_1, B_2, \dots, B_n \}$$

où A_i ($1 \leq i \leq m$) et B_j ($1 \leq j \leq n$) sont des fbf.

La séquence $\Gamma \rightarrow \Delta$ est une assertion sur une fbf; elle affirme la validité de la fbf: $A_1 \wedge A_2 \wedge \dots \wedge A_m \supset B_1 \vee B_2 \vee \dots \vee B_n$

$\Gamma \rightarrow \Delta$ signifie donc: "pour tout modèle de Γ au moins une des fbf B_i est évaluée à V dans le modèle". Ce qui est lu couramment comme: "si toutes les formules de Γ sont V , alors au moins une formule de Δ est V "; ou encore: "la conjonction des formules de Γ implique la disjonction de formules de Δ ".

La *règle de coupure* des systèmes de Gentzen (qui peut être considérée comme une généralisation du "modus ponens") est la suivante:

$$\frac{\Delta \rightarrow \Lambda, C \quad C, \Gamma \rightarrow \theta}{\Delta, \Gamma \rightarrow \Lambda, \theta} \quad \begin{array}{l} C: \text{ fbf} \\ \Delta, \Lambda, \Gamma, \theta: \text{ ensembles de fbf} \end{array}$$

les formules au-dessus de la ligne sont les *prémises* (il faut interpréter cette écriture comme la *conjonction* des prémisses); celles en dessous sont les *conclusions*. Les prémisses et les conclusions s'appellent aussi formules *supérieures* et *inférieures* respectivement [54].

On peut exprimer (voir par exemple [89], [121]) la méthode de résolution comme une théorie formelle pour le calcul de prédicats sans axiomes logiques et avec une seule règle d'inférence: la *règle de résolution*.

La théorie formelle pour la logique de c-clauses est spécifiée comme suit:

1) LANGUAGE:

a) Vocabulaire: symboles fonctionnels (incluant au moins un symbole de constante).

symboles propositionnels

symboles de prédicat

- { } () , .

(si l'on veut pouvoir indiquer explicitement que les clauses sont des disjonctions de littéraux, il faut inclure aussi le symbole " \vee ").

b) Formules bien formées: (i) les c-clauses.

(ii) la clause vide (notée " \square ").

- 2) AXIOMES: aucun.
 3) REGLE D'INFERENCE: la résolution (dans les c-clauses).

La règle d'inférence peut être symbolisée par:

$$\frac{A \vee \Lambda \quad - A \vee \theta}{\Lambda \vee \theta} \quad \text{ou} \quad \frac{\rightarrow A, \Lambda \quad A \rightarrow \theta}{\rightarrow \Lambda, \theta}$$

où A est un littéral positif et Λ et θ représentent des disjonctifs de littéraux.

La deuxième forme d'écriture de la règle d'inférence correspond à la forme d'écriture d'assertions dans PROLOG [75], [125].

On voit aisément que cette règle d'inférence correspond à la coupure (avec $\Delta = \phi$ et $\Gamma = \phi$).

Au § 1.3, on a défini une règle d'inférence comme une *relation* dans l'ensemble de fbf. En fait les règles d'inférence qui nous intéressent, et en particulier la règle de résolution, sont des *fonctions*. La définition de la règle de c-résolution peut donc être faite comme suit:

Δ Soit

L^* : ensemble de tous les c-littéraux

$C^* = P(L^*)$

$P(L^*)$: ensemble de parties finies de L^* .

(C^* représente donc l'ensemble de toutes les c-clauses)

La *c-résolvante de deux clauses* est une application (partielle):

$$R_C : (C^*)^2 \times (L^*)^2 \longrightarrow C^*$$

Si les conditions (a), (b) et (c) ci-dessous sont satisfaites:

(a) $C_i \in C^*$; $C_j \in C^*$

(b) $L_i \in C_i$; $L_j \in C_j$

(c) $L_i = L_j^c$ (c'est-à-dire L_i et L_j sont des littéraux complémentaires)

alors la définition de la fonction est la suivante:

$$R_C(C_i, C_j, L_i, L_j) = (C_i - \{L_i\}) \cup (C_j - \{L_j\})$$

si l'une quelconque des conditions n'est pas satisfaite, R_C n'est pas définie.

Avec la notation ensembliste pour les clauses, l'ensemble vide (correspondant à la clause vide) est aussi noté " \square ".

R_C est appelé *l'opérateur de c-résolution*. Quand on applique R_C à C_i et C_j (et la résolvante existe), on dit qu'on *résout* C_i et C_j . L_i et L_j sont appelés les *littéraux sur lesquels on résout*. C_i et C_j sont appelés les *clauses parents* de la résolvante obtenue.

REMARQUE 1: La c-résolvante de deux clauses est une conséquence logique (§ 1.2.1.1) de ces deux clauses (la preuve en est triviale, en appliquant les tables de vérité).

REMARQUE 2: L'ensemble de toutes les c-résolvantes de deux clauses est toujours fini (la preuve en est triviale; l'ensemble de littéraux de deux clauses est fini et la résolution n'ajoute pas de nouveaux littéraux).

EXEMPLE 1:

$$\begin{aligned}
 S &= \{ C_1, C_2, C_3, C_4 \} \\
 C_1 &: A \vee B \vee C \\
 C_2 &: \bar{C} \\
 C_3 &: \bar{A} \vee C \vee D \\
 C_4 &: C \\
 R_c(C_1, C_3, A, \bar{A}) &= B \vee C \vee D \\
 R_c(C_2, C_4, \bar{C}, C) &= \square
 \end{aligned}$$

(La clause vide est une conséquence de la contradiction $C \wedge \bar{C}$ présente dans l'ensemble de clauses).

EXEMPLE 2:

$$\begin{aligned}
 S &= \{ C_1, C_2 \} \\
 C_1 &: P(a, f(b)) \vee Q(c) \\
 C_2 &: -Q(c) \\
 R_c(C_1, C_2, Q(c), -Q(c)) &= P(a, f(b))
 \end{aligned}$$

Pour la définition précise de la méthode de résolution, il nous faut d'abord étendre le domaine d'application de l'opération de c-résolution à des *ensembles finis de c-clauses*.

Δ Soit

C^* : ensemble de toutes les c-clauses

$\mathcal{C} = P(C^*)$

$P(C^*)$: ensemble de parties finies de C^*

L'ensemble de résolvantes d'un ensemble de clauses est une application

$$R_c: \mathcal{C} \longrightarrow \mathcal{C}$$

définie comme suit:

$$R_c(S) = \{ R_c(C_i, C_j, L_i, L_j) \mid i \neq j; C_i \in S; C_j \in S; L_i \in C_i; L_j \in C_j; L_i = L_j \} \cup \square$$

Etant donné un ensemble fini de clauses S , la *méthode de résolution* engendre une suite ordonnée d'ensembles de clauses. Cette suite est définie de la façon suivante:

$R_c^0(S) = S$ (les clauses de S sont dites de *niveau 0*)

$R_c^{n+1}(S) = R_c^n(S) \cup R_c^n(R_c^n(S))$ (les clauses de $R_c^{n+1}(S) - R_c^n(S)$ sont dites de *niveau n+1*)

REMARQUE: Ceci correspond à la construction de l'ensemble de toutes les conséquences de S (voir § 1.3).

Il est facile de prouver que les propriétés 1 et 2 ci-dessous sont vérifiées pour tout ensemble fini de clauses.

1) $R_c^n(S) \subseteq R_c^{n+1}(S)$ pour $n = 0, 1, 2, \dots$ (d'après la définition de R_c)

2) $\exists p / R_c^{p+1}(S) = R_c^p(S)$, donc $\forall q > p \quad R_c^q(S) = R_c^p(S)$

(parce que le nombre de c-clauses que l'on peut construire avec des littéraux de S est fini, et que l'opération de c-résolution n'engendre pas de nouveaux littéraux).

La méthode de c-résolution qui s'applique à un ensemble S de c-clauses consiste à chercher la clause vide (" \square ") dans les ensembles engendrés. Si on la trouve, alors S est insatisfaisable. Si on ne la trouve pas (la procédure finit aussi dans ce cas puisqu'on arrive toujours à la stabilisation des ensembles de clauses engendrés - voir propriété 2 ci-dessus-), alors S est satisfaisable.

Une implantation de la méthode partant de l'ensemble S et engendrant $R_c^1(S)$, $R_c^2(S)$, ..., c'est-à-dire en "largeur d'abord", ne serait pas pratique; dans les réalisations de la méthode on cherche en "profondeur d'abord"; c'est-à-dire on fait ce que l'on appelle des CR-déductions:

Δ Etant donné un ensemble de c-clauses S et une c-clause A , une *CR-déduction de A à partir de S* (notée $S \stackrel{CR}{\vdash} A$) est une suite finie B_1, B_2, \dots, B_n de c-clauses telle que:

(1) $B_n = A$

(2) $\forall i \quad 1 \leq i \leq n$ soit:

a) $B_i \in S$

soit

b) $B_i = R_c(B_j, B_k, C_j, C_k)$ avec $j < i, k < i \quad \forall$

Δ Une *CR-réfutation* de S est une CR-déduction de \square à partir de $S \quad \forall$

Cette méthode est *correcte* et *complète*, comme l'assure le théorème suivant:

THEOREME (de c-résolution): Soit S un ensemble de c-clauses. S est insatisfaisable ssi $\exists n \geq 0$ tel que $\square \in R_c^n(S)$.

Autrement dit: Un ensemble de clauses S est insatisfaisable ssi $S \stackrel{CR}{\vdash} \square$.

PREUVE : Dans [89]. ■

REMARQUE: S peut être un ensemble *infini* de clauses (pour les détails de la démonstration voir [89]).

Dans les réalisations de démonstrateurs de théorèmes basés sur ce principe, on cherche une CR-réfutation de S.

On voit que cette définition de dérivation correspond à celle de conséquence (ou de déductible) donnée au § 1.3.

EXEMPLE: CR-réfutation d'un ensemble de c-clauses

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

$$C_1. \quad A \vee B \vee \bar{D}$$

$$C_2. \quad \bar{A} \vee C \vee \bar{D}$$

$$C_3. \quad \bar{B}$$

$$C_4. \quad \bar{C}$$

$$C_5. \quad D$$

$$C_6. \quad B \vee C \vee \bar{D} \quad \text{de } C_1 \text{ et } C_2$$

$$C_7. \quad C \vee \bar{D} \quad \text{de } C_3 \text{ et } C_6$$

$$C_8. \quad \bar{D} \quad \text{de } C_4 \text{ et } C_7$$

$$C_9. \quad \square \quad \text{de } C_5 \text{ et } C_8$$

La définition suivante nous sera utile:

Δ Etant donné un ensemble de littéraux C (card (C) ≥ 2), s'il existe une substitution σ tel que σ C est un singleton, alors nous dirons que *les littéraux de C sont unifiables* ou que *l'ensemble C de littéraux est unifiable*, ou plus brièvement que *C est unifiable*. ▽

REMARQUE: Il est évident que, pour qu'un ensemble de littéraux soit unifiable, ils doivent être tous positifs ou tous négatifs.

2.2.7.2 LA RESOLUTION DANS LE CAS GENERAL

A la différence des méthodes de saturation, elle *fusionne* les processus de substitution des variables et de test propositionnel.

Elle permet aussi de spécifier le Calcul de Prédicats comme un système formel sans axiomes logiques et avec une seule règle d'inférence.

Δ Soit

L^* : ensemble de tous les littéraux

$C^* = P(L^*)$

$P(L^*)$: ensemble de parties finies de L^*

La *résolvante de deux clauses* est une application (partielle)

$$\mathbf{R}: (C^*)^2 \times (L^*)^2 \longrightarrow C^*$$

Si les conditions (a), (b), (c) et (d) ci-dessous sont satisfaites:

(a) $C_i \in C^*$; $C_j \in C^*$; $L_i \in L^*$; $L_j \in L^*$

(b) L_i, L_j ensembles de littéraux, tel que tous les littéraux de L_i sont positifs (négatifs) et tous les littéraux de L_j sont négatifs (positifs)

$$L_i \subseteq C_i, L_j \subseteq C_j \quad (L_i \neq \emptyset, L_j \neq \emptyset)$$

(c) Si $EV(C_i) \cap EV(C_j) \neq \emptyset$ alors appliquer à C_i , ou à C_j ou à C_i et C_j une substitution de renommage η telle que:

$$EV(\eta C_i) \cap EV(\eta C_j) = \emptyset \quad \eta: \text{substitution de renommage ou identité partout}$$

c'est-à-dire après renommage C_i et C_j n'ont pas de variable en commun.

(d) Si l'on considère à la place de l'ensemble de littéraux négatifs l'ensemble de ses littéraux complémentaires, alors ηL_i et ηL_j sont unifiables avec p.g.u. σ .

Alors la définition de la fonction est la suivante:

$$\mathbf{R}(C_i, C_j, L_i, L_j) = \sigma \eta (C_i - \{L_i\}) \cup \sigma \eta (C_j - \{L_j\})$$

Si l'une des conditions n'est pas satisfaite, la fonction n'est pas définie.

En adoptant cette définition de résolvante les définitions de résolution et de résolution de niveau n sont les mêmes que celles données au § 2.2.7.1. On définit aussi d'une façon analogue à CR-dérivation et à CR-réfutation une R-dérivation et une R-réfutation. Dans la définition de l'opérateur \mathbf{R} il faut considérer *aussi* les résolvantes entre une clause et ses variantes.

Δ Si l'on prend L_i et L_j tels que $\text{card}(L_i) = \text{card}(L_j) = 1$, la définition ci-dessus est celle de *résolvante binaire de deux clauses* C_i et C_j . ∇

L'utilisation de la résolution binaire comme seule règle d'inférence ne donne pas une procédure de preuve complète (voir l'exemple après la définition de R-logique). Pour lui "redonner" la complétude, il faut ajouter la factorisation:

Δ Soit

L^* : ensemble de tous les littéraux

$$C^* = \mathcal{P}(L^*)$$

$\mathcal{P}(L^*)$: ensemble de parties finies de L^*

La factorisation d'une clause est une application (partielle).

$$\mathbf{F}: (C^*)^2 \longrightarrow C^*$$

si les conditions (a), (b) et (c) ci-dessous sont satisfaites

a) $C \in C^*$; $L \in C^*$

b) $L \subseteq C$;

c) L est unifiable avec p.g.u. σ

alors la définition de la fonction est la suivante:

$$F(C,L) = \sigma C$$

σC est dite un *facteur* de C .

Si l'on admet que $\text{card}(L) \geq 1$, et que la clause contenant un seul littéral est un ensemble unifiable, une clause est alors un facteur d'elle même). ∇

EXEMPLE:

$C: \{ P(x), P(f(y)), R(g(y)) \}$

$C': \{ P(f(y)), R(g(y)) \}$ est un facteur de C .

La logique du premier ordre peut être spécifiée comme système formel basé sur la résolution:

(1) LANGAGE:

a) Vocabulaire: variables

symboles fonctionnels

symboles propositionnels

symboles de prédicat

$\{ \} () ,$

(si l'on veut indiquer explicitement que les clauses sont des disjonctions de littéraux, il faut inclure aussi le symbole " \vee ").

b) Formules bien formées; (i) les clauses.

(ii) la clause vide (" \square ").

(2) AXIOMES: aucun.

(3) REGLE(S) D'INFERENCE:

soit a) résolution.

soit b_1) résolution binaire.

et

b_2) factorisation.

Dans les implantations de la résolution, on utilise généralement la résolution binaire et la factorisation.

La méthode de résolution est *correcte* et *complète*:

THEOREME: Soit S un ensemble de clauses, alors S est insatisfaisable ssi $S \vdash_R \square$.

PRELVE: Dans [89] . \blacksquare

Parfois il est possible de détecter un ensemble de clauses satisfaisable en appliquant la méthode de résolution:

THEOREME: Soit S un ensemble de clauses, si $\exists m (m \geq 0) / \square \notin R^m(S)$ et $R^{m+1}(S) = R^m(S)$ alors S est satisfaisable.

PREUVE: D'après la définition de \forall : $\forall n \geq m \quad R^n(S) = R^m(S)$.

Si $\square \notin R^m(S)$ alors la clause vide ne sera jamais engendrée et d'après la complétude de la résolution, S est satisfaisable. ■

EXEMPLE:

Soit l'ensemble de clauses:

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

$$C_1. \quad P(a) \quad P(u)$$

$$C_2. \quad -R(x) \quad S(a,x)$$

$$C_3. \quad -P(y) \quad -Q(z) \quad -S(y,z)$$

$$C_4. \quad R(f(a))$$

$$C_5. \quad Q(f(a))$$

$$C_6. \quad S(a,f(a)) \quad \text{de } C_2 \text{ et } C_4$$

$$C_7. \quad P(a) \quad \text{factorisation de } C_1$$

$$C_8. \quad -Q(z) \quad -S(a,z) \quad \text{de } C_3 \text{ et } C_7$$

$$C_9. \quad -S(a,f(a)) \quad \text{de } C_5 \text{ et } C_8$$

$$C_{10}. \quad \square \quad \text{de } C_6 \text{ et } C_9$$

S est donc insatisfaisable.

REMARQUE: La factorisation est *nécessaire* à la complétude de la résolution binaire. Voici un exemple très simple d'un ensemble de clauses insatisfaisable et pourtant on n'en obtient pas la clause vide [89].

EXEMPLE:

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

$$C_1. \quad P(x) \vee P(y)$$

$$C_2. \quad -P(u) \vee -P(v)$$

$$C_3. \quad P(y) \vee -P(v) \quad \text{de } C_1 \text{ et } C_2 \text{ avec } \sigma : \{ x \rightarrow u \}$$

$$C_4. \quad P(y) \vee P(z) \quad \text{de } C_1 \text{ et } C_3 \text{ avec } \eta : \{ z \rightarrow y \} \quad \eta C_1: P(x) \vee P(z)$$

$$\sigma = \{ v \rightarrow x \}$$

$$C_5. \quad -P(v) \vee -P(w) \quad \text{de } C_2 \text{ et } C_3 \text{ avec } \eta : \{ w \rightarrow v \} \quad \eta C_2: -P(u) \vee -P(w)$$

$$\sigma = \{ u \rightarrow y \}$$

$$C_6. \quad P(z) \vee -P(w) \quad \text{de } C_4 \text{ et } C_5 \quad \sigma = \{ y \rightarrow v \}$$

On voit facilement qu'on ne peut pas déduire des clauses de moins de 2 littéraux.

Si l'on utilise la résolution binaire et aussi la factorisation, on obtient:

C'_3 . $P(x)$ facteur de C_1 $\sigma = \{ x \rightarrow y \}$

C'_4 . $\neg P(u)$ facteur de C_2 $\sigma = \{ u \rightarrow v \}$

C'_5 . \square de C'_3 et C'_4 $\sigma = \{ x \rightarrow u \}$

Une définition de l'opérateur R utilisant la résolution binaire et de la factorisation peut être trouvée dans [89].

Plusieurs stratégies complètes existent pour la résolution (voir p. ex. [27] et [89]).

Il est intéressant de pouvoir *comparer* des preuves; pour ce faire, il faut être capable de *mesurer* les mêmes. Cette mesure est appelée la *complexité* de la preuve.

2.3 COMPLEXITE DES PREUVES ET DES PROCEDURES DE PREUVE

Dans ce paragraphe et dans le suivant, "ensemble de clauses" sera synonyme de "ensemble de c-clauses".

Nous concentrerons notre intérêt dans deux méthodes, celle de résolution et celle de Davis et Putnam, parce qu'elles sont suffisamment représentatives, et parce qu'une partie très importante de la littérature sur la complexité leur est consacrée. Une étude comparative de la complexité de diverses méthodes peut être trouvée dans [34] et [35].

La définition de déduction (ou réfutation) donnée au § 2.2.7.1 correspond à une définition *linéaire*, mais une définition équivalente sous forme d'arbre binaire est aussi possible [52]:

Δ Un *arbre de preuve* (par résolution) pour une clause C (utilisant S) est un arbre binaire A dont les nœuds sont étiquetés de la façon suivante:

- 1) C est la racine de A .
- 2) Toute feuille est une des clauses de S .
- 3) Toute clause d'un nœud intérieur est la résolvante de deux clauses étiquetant ses ancêtres. ∇

Δ Un *arbre de preuve* (ou de *réfutation*) par résolution pour un ensemble de clauses S est un arbre de résolution pour \square . ∇

Δ La *complexité* d'un *arbre de preuve par résolution* A , notée $N(A)$, est le nombre de clauses différentes apparaissant comme nœuds de A . (Si A correspond à une réfutation linéaire, alors $N(A)$ est le nombre de clauses de cette réfuta-

tion). \forall

EXEMPLE [52]:

Soit l'ensemble de clauses $S = \{ C_1, C_2, C_3, C_4 \}$

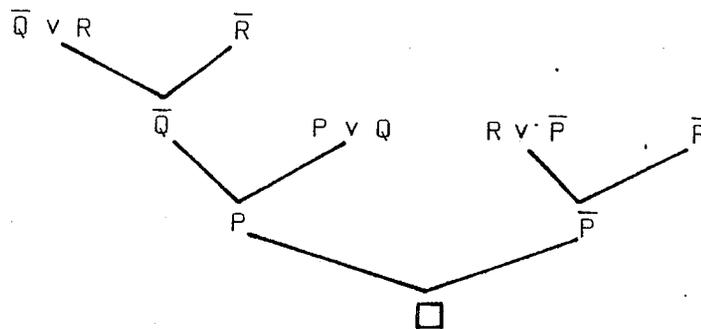
$C_1: \bar{R}$

$C_2: R \vee \bar{P}$

$C_3: P \vee Q$

$C_4: \bar{Q} \vee R$

l'arbre suivant est un arbre de preuve par résolution de S (de complexité 9)



La complexité d'une procédure de preuve est définie (en fonction des données correspondant au cas le plus défavorable) de la façon suivante [52].

Δ Si l'on note:

S : le nombre de littéraux différents dans les clauses de S
et

$A_S = \{ A \mid A \text{ est un arbre de preuve par résolution pour } S \}$

La complexité de la procédure de résolution est une fonction $Comp$, définie comme suit:

$$Comp(n) = \max_{|S|=n} \min_{A \in A_S} N(A). \forall$$

La complexité de la résolution n'est pas encore connue, mais il existe la conjecture suivante.

CONJECTURE: *La complexité de la résolution est exponentielle.*

REMARQUE: La méthode de Davis et Putnam qui est de complexité exponentielle ([34], [52]) peut être considérée comme une procédure qui engendre des preuves par résolution d'une certaine forme [34].

Une règle dite *d'extension* fut introduite par Tseitin en 1968 [153]; elle est utilisée en conjonction avec la résolution, donnant lieu à la *résolution étendue*.

La règle d'extension permet l'introduction de nouveaux littéraux et de nouvelles clauses, et avec certaines substitutions permet des raccourcissements pour la réfutation de certains ensembles de clauses; c'est pourquoi elle est citée surtout dans les travaux sur la complexité des méthodes de preuve (pour la résolution étendue on peut consulter aussi [34], [52]).

Pour des raisons pratiques évidentes, il est clair que même pour des théories décidables (comme, par exemple, le Calcul Propositionnel), le nombre de fbf dans une démonstration d'un théorème est de la plus grande importance. C'est pourquoi nous parlons maintenant brièvement du problème de la complexité des procédures de décision pour certaines théories, et des limites à la mise en œuvre des telles procédures. Nous parlerons ensuite de l'importance (qui va bien au-delà de la démonstration automatique) du problème consistant à déterminer, pour une fbf du Calcul Propositionnel, si elle est une tautologie. Comme nous l'avions annoncé au début nous finirons en considérant des résultats sur la complexité moyenne des procédures de Davis et Putnam et de résolution.

2.4 QUELQUES QUESTIONS EN RAPPORT AVEC LA COMPLEXITE DES PREUVES

2.4.1 LIMITES THEORIQUES (ET PRATIQUES) INTRINSEQUES A CERTAINES CLASSES DE PROBLEMES

Si l'on se place d'un point de vue purement formaliste, par exemple le programme de Hilbert, la preuve d'un théorème dans une théorie décidable est une question triviale puisqu'une méthode mécanique est applicable. (Pour une théorie semidécidable, il y aura en plus le problème de l'arrêt dans le cas où la formule testée n'est pas un théorème). Mais on peut légitimement se demander:

a) Quelle est la *faisabilité* d'une telle méthode?

et s'il en existe plusieurs:

b) Quelle est leur *efficacité relative* ?

Les problèmes a et b se présentent pour les algorithmes dans divers domaines, qu'on peut classer en deux grandes catégories: calcul numérique, et résolution des problèmes dans lesquels la recherche est faite dans un ensemble partiellement ordonné de solutions possibles. En particulier cette deuxième classe d'algorithmes est celle utilisée en Intelligence Artificielle.

Depuis quelques années, il y a eu un intérêt croissant pour l'étude de la complexité de calcul de théories décidables (ou de problèmes, dont on avait prouvé l'existence d'une solution). Un exposé des travaux sur les théories décidables peut être trouvé dans [117].

On a appelé *intrinsèquement complexes* ("intractables") les problèmes dont le nombre de pas, pour trouver une solution dans une infinité de cas, pouvait

être "énorme". Des exemples de ces problèmes dans [85], [147].

Dans [147], une intéressante spéculation est faite sur les milliards d'années que devrait tourner un programme sur un ordinateur hypothétique dont les limites seraient seulement celles imposées par les lois physiques et la dimension de l'univers pour résoudre certains problèmes. En tout ce qui suit, nous nous restreindrons aux problèmes de complexité dans la démonstration de théorèmes (avec d'éventuelles extensions à l'Intelligence Artificielle).

Dans [35], une définition rigoureuse de complexité d'une preuve "linéaire" est donnée. Elle s'adapte parfaitement aux problèmes sur la complexité des preuves.

En ce qui concerne le problème "a" évoqué ci-dessus; il a été montré que, pour certaines théories décidables, *n'importe quel algorithme* de décision exige un nombre de pas tellement élevé que la théorie devient *du point de vue pratique indécidable*. Nous donnons deux exemples:

1) Dans l'arithmétique de Presburger (arithmétique de l'addition des nombres naturels) qui est une théorie décidable, il a été montré ([48], [118]) que pour *tout* algorithme de décision, il existe des fbf A (dont le nombre de symboles est n) qui exigent de l'algorithme 2^{2^n} pas de calcul pour décider si A est ou n'est pas un théorème.

2) Soit le langage du second ordre connu comme S1S ("logique monadique faible du second ordre avec successeur"), qui permet d'écrire des assertions sur des ensembles d'entiers (quantification sur les ensembles, appartenance d'un élément à un ensemble) et qui est muni de la fonction successeur. Bien que ne permettant pas d'écrire des formules contenant la somme de deux variables, ce langage est assez puissant et permet, par exemple, d'exprimer le principe du bon ordre pour les entiers (dont nous avons vu au § 1.8 qu'on ne pouvait pas le faire tout au moins sans ajouter les axiomes sur les entiers, en logique du premier ordre):

$$\forall S ((\exists x) (x \in S) \supset (\exists z) (z \in S \wedge (\forall y) (y \in S \supset z \leq y)))$$

(S: ensemble d'entiers)

S1S est *décidable*, mais sa complexité est *superexponentielle*.

En ce qui concerne le problème "b" évoqué ci-dessus, la question se pose de comment mesurer la complexité des algorithmes. Une mesure de complexité pour les arbres de recherche (voir par ex. [143]) est la *profondeur minimale* à laquelle une solution est trouvée.

2.4.2 PROBLEME DE LA TAUTOLOGIE ET LA QUESTION $P \stackrel{?}{=} NP$

On appelle *problème de la satisfaisabilité* (ou de la *tautologie*) pour le Calcul Propositionnel le problème de savoir si une fbf de ce calcul a un modèle (si elle est une tautologie). Le problème de la satisfaisabilité (de la tautologie) correspond au problème de la décision dans le même calcul (c'est-à-dire savoir si une fbf donnée du calcul est un théorème) et il est extrêmement important de connaître la complexité des algorithmes de décision; son intérêt dépasse largement le domaine de la logique et de la démonstration automatique. Nous pouvons citer 2 raisons qui justifieraient à elles seules l'étude de ce problème:

- 1) *Le problème de la décision pour toute théorie formalisée est au moins aussi complexe que le problème de la décision du Calcul Propositionnel [117].*
- 2) *Le problème de la décision du Calcul Propositionnel est intimement lié au problème $P \stackrel{?}{=} NP$ (théorèmes de Cook ci-dessous).*

Il est bien connu que si l'on veut des algorithmes qui "puissent tourner", il faut éviter ceux dont la complexité est exponentielle (l'exponentielle croissant plus vite que n'importe quelle autre fonction), donc pour le problème de la satisfaisabilité, on a aussi cherché des algorithmes polynômiaux.

Il est possible de construire un programme *non déterministe* qui décide si une fbf A est satisfaisable dans un nombre de pas qui est une fonction polynomiale du nombre de symboles de A.

Une question qui se pose est de savoir s'il existe une procédure de décision *déterministe* polynomiale.

La réponse à cette question n'est pas connue.

Cependant, il existe des résultats extrêmement importants à ce propos; ces résultats mettent en rapport le problème de la tautologie avec le célèbre problème $P \stackrel{?}{=} NP$ (question d'un intérêt capital du point de vue de l'informatique théorique). Habituellement, on note P (respectivement NP) la classe des ensembles de chaînes de symboles, qui peuvent être reconnues par une machine de Turing *déterministe* (respectivement *non déterministe*) en temps borné par une fonction polynomiale du nombre de symboles de l'entrée.

Si l'on suppose une formalisation du Calcul Propositionnel avec un ensemble donné de connectifs (par exemple celui du § 1.2), et si l'on note TAUT l'ensemble de tautologies, alors Cook a démontré les théorèmes suivants:

THEOREME: $P = NP$ ssi TAUT est contenu dans P. (Autrement dit le problème de la tautologie est NP-complet).

PREUVE: Dans [33], [35]. ■

Δ Soit Σ l'alphabet fini considéré; Σ^* le monoïde libre engendré par Σ , L un ensemble de chaînes de symboles, $L \subseteq \Sigma^*$. Si L dans NP implique que $\Sigma^* - L$ est dans NP, alors on dit que NP est *fermé par complémentation*. ▽

THEOREME: NP est fermé par complémentation ssi TAUT est contenu dans NP.

PREUVE: Dans [33], [35]. ■

Il a été montré que beaucoup de problèmes combinatoires très compliqués sont réductibles au problème de la satisfaisabilité. Le problème de la satisfaisabilité est donc très complexe, ce qui renforce la plausibilité de la conjecture suivante:

CONJECTURE: $P \neq NP$.

Du point de vue de la démonstration automatique, une formulation plus intéressante est:

CONJECTURE: *Le calcul propositionnel est exponentiellement complexe.*

(comparer avec la conjecture du § 2.3)

Cette conjecture, unie au fait que beaucoup de recherches ont été faites pour améliorer l'efficacité des preuves, nous amènent à nous poser la question suivante:

Pour une procédure de preuve quelconque, peut-on éviter d'engendrer des formules sans rapport avec la preuve, c'est-à-dire en adoptant la terminologie du théorème de Herbrand: "Pour un ensemble de clauses S , peut-on assurer, quand on a trouvé un ensemble fini de c -clauses de S insatisfaisable, qu'il est *minimalement* insatisfaisable?"

Certains résultats de la théorie des fonctions récursives ([23], [99]) permettent de donner une *réponse négative à cette question*, c'est-à-dire: on ne peut pas assurer qu'on n'engendrera pas de formules inutiles à la preuve.

Malgré ces résultats qu'on peut qualifier de "négatifs" en ce qui concerne les possibilités pratiques des méthodes de preuve, d'autres résultats plus encourageants ont été publiés plus récemment. Ils portent sur la complexité *moyenne* de plusieurs méthodes (Davis et Putnam, résolution, graphes de réfutation [136]). Il y est montré [59] que la complexité *moyenne* en temps pour résoudre le problème de la tautologie pour ces méthodes, est *polynômiale*, et son ordre est:

$$O(r n^2)$$

où:

n : nombre de clauses

r : nombre de littéraux différents dans l'ensemble de clauses.

C'est-à-dire, si l'on appelle $CM(r,n)$ la complexité moyenne du test de tautologie pour des ensembles de n clauses avec r littéraux différents, alors il

existe une constante k telle que:

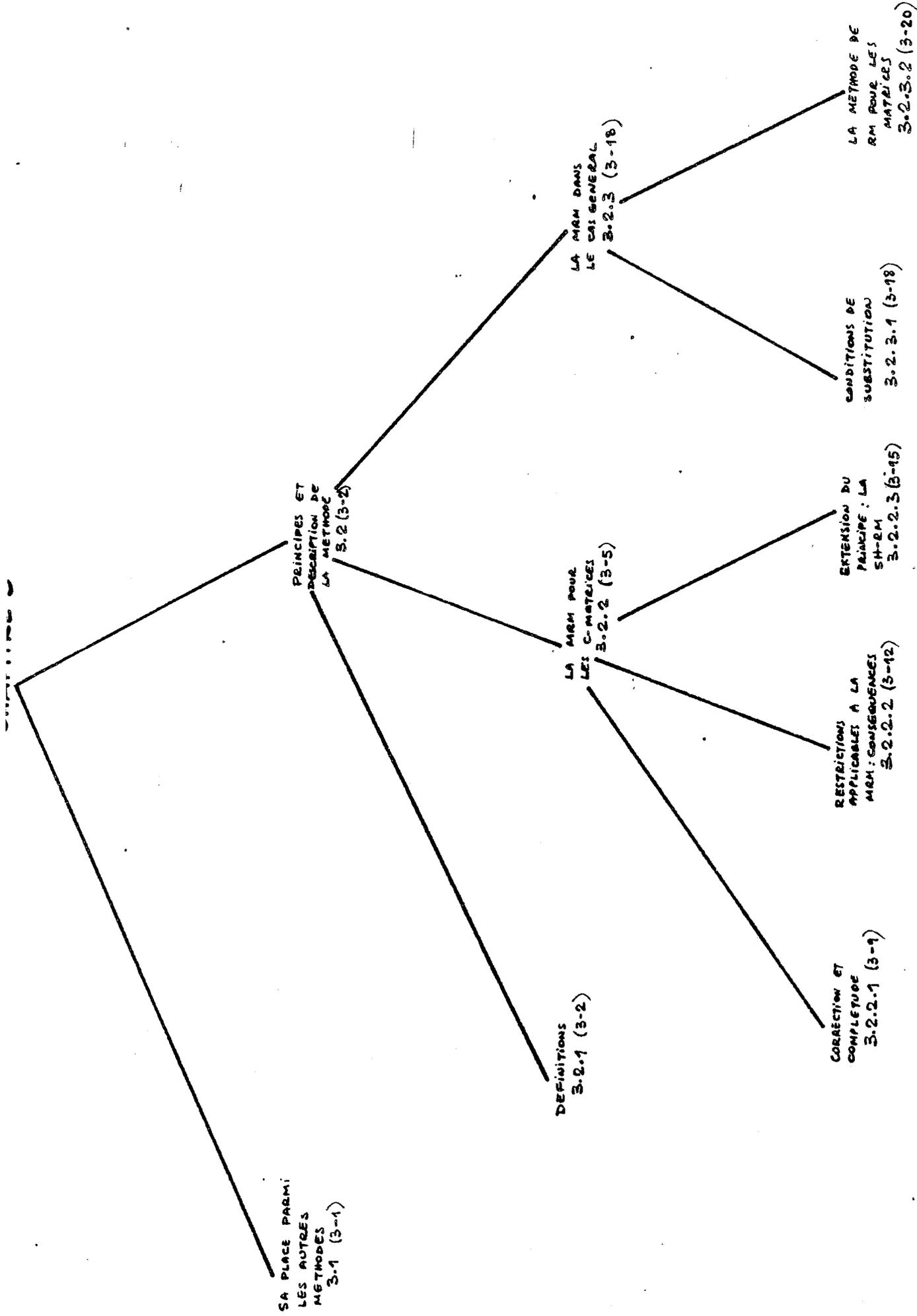
$$CM(r,n) \leq k \times r \times n^2$$

pour $\forall r, n \in \mathbb{N} - E$ ($E \subset \mathbb{N}$, E : fini) \mathbb{N} : les naturels

Comme le fait remarquer Bibel [12] ces conclusions permettent de considérer à nouveau la démonstration automatique comme un domaine prometteur.

CHAPITRE 3

LA METHODE DE REDUCTION MATRICIELLE



SA PLACE PARMI
LES AUTRES
METHODES
3-1 (3-1)

PRINCIPES ET
DESCRIPTION DE
LA METHODE
3-2 (3-2)

DEFINITIONS
3-2.1 (3-2)

LA MAM POUR
LES C-MATERIEES
3-2.2 (3-5)

LA MAM DANS
LE CAS GENERAL
3-2.3 (3-18)

CORRECTION ET
COMPLETION
3-2.2.1 (3-9)

RESTRICTIONS
APPLICABLES A LA
MAM : CONSEQUENCES
3-2.2.2 (3-12)

EXTENSION DU
PRINCIPE : LA
SH-ERM
3-2.2.3 (3-15)

CONDITIONS DE
SUBSTITUTION
3-2.3.1 (3-18)

LA METHODE DE
MAM POUR LES
MATERIEES
3-2.3.2 (3-20)

Avant de présenter la méthode de réduction matricielle, nous essaierons de montrer la place qu'elle a occupée parmi les autres méthodes, ce qui justifiera, à notre avis, le fait que nous nous y soyons intéressés.

3.1 SA PLACE PARMIS LES AUTRES METHODES

La méthode de réduction matricielle (MRM), due à Prawitz ([24], [27], [111], [112]) a été, depuis sa publication en 1969, l'objet de deux attitudes différentes:

1) D'abord elle a été considérée comme inintéressante et abandonnée en faveur, notamment, de la résolution.

Les critiques qui lui étaient adressées concernaient surtout une première version de la méthode [110]. Pour ces critiques, on peut consulter par exemple [132] où la méthode est disqualifiée essentiellement en fonction de la place mémoire exigée. La deuxième version de la méthode (qui est celle à laquelle nous nous sommes intéressés, et qui reçoit le nom de "réduction matricielle" (RM)), répond déjà en partie à ces critiques. Nous pensons que la technique de partage de structure proposée dans le Chapitre 4 donne une réponse définitive à l'objection sur l'importance de la place nécessaire pour une mise en œuvre de la méthode.

2) Plus récemment -approximativement à partir de 1976- il y a eu un regain d'intérêt pour la méthode, ou tout au moins pour des méthodes basées sur des principes similaires à celui de la réduction matricielle [3], [4], [11], [12], [13], [14], [28], [62], [129].

Nous pensons que ce renouveau d'intérêt a deux causes: la première étant la même que celle qui a été à l'origine de méthodes de démonstration n'utilisant pas la résolution [17], à savoir que la presque totalité des travaux avaient été dédiés à la résolution et que l'on avait négligé d'explorer d'autres voies.

La deuxième cause réside, d'une part dans la constatation qu'une des faiblesses de la résolution était sa *localité*, d'où l'intérêt pour des méthodes dites *globales* (dont la MRM), et d'autre part dans la possibilité d'appliquer des techniques de partage de structure, comme cela avait déjà été fait pour la résolution, à une méthode comme la MRM exigeant plus de place mémoire [129], [130].

REMARQUE: Le premier à notre connaissance à avoir classifié la RM comme "globale" et la résolution comme "locale" a été Maslov [97].

Une des façons de situer une méthode est de la comparer à d'autres, ce qui est réputé être une tâche difficile. Les travaux où l'on compare la MRM à d'autres méthodes sont à notre connaissance, et par ordre chronologique: [112],

[88], [62], [13].

Dans [112] des considérations très concises et très générales sont faites, en particulier, la conjecture appelée par Loveland [88] la "conjecture d'un pas en avant" ("one step ahead conjecture") , y a été émise.

Dans [88] il a été montré que des liens très étroits ("isomorphismes") existent entre la MRM, une forme de la résolution linéaire ([80], [89]) et la méthode d'élimination de modèles ([86], [89]).

Dans [62], une analogie est faite avec la méthode, due à Henschen, d'"expressions de recouvrement" ("covering expressions").

Dans [13], une vision unificatrice de plusieurs méthodes (résolution, SL-résolution, élimination de modèles, ...) à partir d'une méthode due à Bibel est donnée. En outre, plusieurs aspects importants de la MRM qui avaient été ignorés à l'époque de sa parution y sont signalés.

Dans [24], nous avons comparé brièvement la résolution de base et la MRM sur quelques points (voir aussi § 3.2.2.2 et § 3.2.3).

Nous considérons que les données et conclusions de ces travaux n'autorisent pas à conclure sur la valeur relative de la méthode par rapport par exemple à la résolution, ni sur ses possibilités.

En outre, elle utilise les concepts de "chemin" et de "matrice", concepts qui se sont révélés très fructueux et qui ont été étudiés et développés plus récemment par Bibel ([11], [12], [13], [14]).

3.2 PRINCIPES ET DESCRIPTION DE LA METHODE

Dans tout ce qui suit, nous abrègerons "forme normale disjonctive" par "fnd", et nous appellerons *matrices* les ensembles de clauses.

Comme pour la résolution, nous parlerons d'abord de la MRM pour les c-clauses, et ensuite de la MRM pour le cas général.

3.2.1 DEFINITIONS

Δ Soit M une matrice:

$$M = \{ C_1, C_2, \dots, C_m \}$$

C_i ($1 \leq i \leq m$) est une clause:

$$C_i = \{ L_1^i, L_2^i, \dots, L_{n_i}^i \}$$

un chemin P_K de M est un ensemble de m littéraux:

$$P_K = \{ L_j^i / L_j^i \in C_i \quad 1 \leq i \leq m, \quad 1 \leq j \leq n_i \text{ et } \forall L, L' \in P_K \text{ si } L \in C_1 \text{ et } L' \in C_1, \text{ alors } C_1 \neq C_1, \} \cdot \forall$$

(Un chemin de M correspond donc à l'ensemble de littéraux d'un disjunct de la fnd de M).

Nous dénotons $P_{\mathcal{A}}(M)$ l'ensemble de chemins de M .

D'après ces définitions il est évident que:

$N_M = \text{card}(P_{\mathcal{A}}(M)) = n_1 \times n_2 \times \dots \times n_m$ où $n_i = \text{card}(C_i)$ $1 \leq i \leq m$ et que la fnd de M est obtenue de la façon suivante:

$$\text{fnd}(M) = \bigvee_{k=1}^{N_M} \bigwedge_{i=1}^m L_{J_i}^i \in P_k$$

Δ Un chemin P_k d'une matrice est dit un *chemin contradictoire* ssi la conjonction des littéraux de P_k est contradictoire. ▽

L'équivalence avec la définition suivante plus simple est triviale:

Δ Un chemin P_k d'une matrice est dit *chemin contradictoire* ssi il contient deux littéraux complémentaires. ▽

Pour mieux apprécier le gain représenté par l'application de la règle de RM, nous exposons d'abord une méthode qui teste l'insatisfaisabilité d'une matrice en testant la contradiction de *tous* les chemins de la matrice (comparer avec la méthode de Gilmore § 2.2.3.1); cette méthode est une description détaillée de la version préliminaire de la MRM proposée par Prawitz dans [111].

Nous donnons une version non-déterministe; une version déterministe équivalente peut être trouvée dans [13].

fonction INSAT (M: matrice): booléen;

* détecte si une matrice $M = \{C_1, C_2, \dots, C_m\}$ avec $\text{card}(C_i) = n_i$ ($1 \leq i \leq m$) est insatisfaisable en testant l'existence dans tout chemin de M d'une paire de littéraux complémentaires *

début

Donner une façon d'énumérer tous les chemins de M;

$n \leftarrow n_1 \times n_2 \times \dots \times n_m$;

$i \leftarrow 1$;

$NP \leftarrow m \times (m-1)/2$; * nombre de paires de littéraux différents dans un chemin *

exismod \leftarrow faux ;

tantque ($i \leq n$) et (non exismod)

faire

$P_i \leftarrow$ i-ème chemin dans l'énumération

Donner une façon d'énumérer toutes les paires de P_n ;

cont \leftarrow faux ;

$k \leftarrow 1$;

tantque ($k \leq NP$) et (non cont)

faire

$\langle L_i, L_j \rangle \leftarrow$ k-ème paire dans P_n ;

si $L_i = L_j^c$ alors cont \leftarrow vrai

sinon $k \leftarrow k + 1$

fsi

ffaire

si $k > NP$ alors exismod \leftarrow vrai

sinon $i \leftarrow i + 1$

fsi

ffaire

insat \leftarrow non exismod ;

* si exismod alors P_n est un modèle de M *

fin * INSAT * ;

3.2.2 LA METHODE DE REDUCTION MATRICIELLE POUR LES C-CLAUSES

La méthode décrite dans le paragraphe précédent pourrait être qualifiée de "brutale" puisqu'aucune optimisation n'est faite.

Soit par exemple des deux matrices suivantes

$$\begin{array}{ccc} A & & A \ B \\ \bar{A} & \text{et} & \bar{A} \\ \Gamma & & \bar{B} \\ & & \Gamma \end{array}$$

(Γ dénote un ensemble de clauses)

Si n_{Γ} dénote le nombre de chemins de Γ , alors la boucle externe s'exécutera n_{Γ} fois dans un cas, et $2 \times n_{\Gamma}$ fois dans l'autre, bien qu'il soit clair que tous les chemins (dans le deux cas) sont contradictoires, sans qu'il soit nécessaire de les tester tous.

A partir de ces exemples, il est légitime de se demander si l'on ne pourrait pas trouver une façon de conclure à l'insatisfaisabilité de M sans tester tous ses chemins.

La MRM répond à cette question. Elle est fondée sur une propriété (a) et une observation (b).

(a) *Une matrice M est insatisfaisable ssi tout chemin de M contient une paire de littéraux complémentaires.*

(L'algorithme *insat* du § 3.2.1 utilise cette propriété).

(b) *Dans une matrice M , une paire de littéraux complémentaires peut être formée de deux littéraux appartenant à plus d'un chemin de M , dans ce cas cette paire rendra contradictoires tous les chemins la contenant. Si une telle paire est détectée, le nombre de tests nécessaires pour prouver la contradiction de tous les chemins de M diminue.*

On donnera un nom aux matrices dont l'insatisfaisabilité est "évidente".

Δ Une matrice avec deux clauses unitaires qui sont des littéraux complémentaires est dite une *matrice simplement contradictoire*. Une telle matrice est notée " $[]$ ". ∇

D'après la définition de chemin, il est trivial de vérifier que tous les chemins d'une matrice " $[]$ " sont contradictoires. Autrement dit, tous les disjoints de la *fn*d de M sont contradictoires.

CONVENTION PRATIQUE: Pour simplifier l'exposé de la méthode, nous admettrons qu'une matrice " $[]$ " soit remplacée par l'ensemble vide (l'ensemble vide ne représentant pas la matrice vide, mais *l'ensemble de chemins* qu'il faudrait encore tester pour détecter l'insatisfaisabilité de la matrice. Dans le cas d'une matrice " $[]$ " il ne reste évidemment plus de chemin à tester.

Nous définissons maintenant, avec une notation fonctionnelle, la règle de réduction matricielle.

Δ La règle de réduction matricielle est une application

$$\mathbf{M}_C: M^* \times (C^*)^2 \times (L^*)^2 \longrightarrow P_f(M^*)$$

où

L^* : ensemble de tous les c-littéraux.

$C^* = P_f(L^*)$: ensemble de toutes les parties finies de L^* (les clauses)

$M^* = P_f(C^*)$: ensemble de toutes les parties finies de C^* (les matrices)

Parmi les ensembles appartenant à $P_f(M^*)$ nous sommes pour l'instant intéressés seulement à ceux de cardinalité 2.

Si les conditions (a) - (b) et (c) ci-dessous sont satisfaites:

(a) $M \in M^*$

(b) $C_i \in M ; C_j \in M$

(c) $L_i \in C_i ; L_j \in C_j \quad L_i = L_j^c$ (L_i et L_j sont des littéraux complémentaires)

alors la valeur de \mathbf{M}_C est définie comme suit:

$$\mathbf{M}_C(M, C_i, C_j, L_i, L_j) = \{ X \cap M_G, Y \cap M_D \}$$

où

$$M_G = (M - \{C_i\} - \{C_j\}) \cup \{\{L_i\}\} \cup \{C_j - \{L_j\}\}$$

$$M_D = (M - \{C_i\} - \{C_j\}) \cup \{\{L_j\}\} \cup \{C_i - \{L_i\}\}$$

$$X = \begin{cases} \phi & \text{si } C_j = \{L_j\} \\ M_G & \text{sinon} \end{cases}$$

$$Y = \begin{cases} \phi & \text{si } C_i = \{L_i\} \\ M_D & \text{sinon} \end{cases}$$

si l'une quelconque des conditions (a), (b) ou (c) n'est pas satisfaite

alors \mathbf{M}_C n'est pas définie. ∇

Prawitz [111], [112] n'a pas présenté sa méthode avec une notation fonctionnelle, mais d'une façon informelle plus directe

$$M: \begin{array}{|l} P \\ \hline \bar{P} \\ \hline \Gamma \end{array} \quad \alpha \quad \beta$$

$$M_G: \begin{array}{|l} P \\ \hline \Gamma \end{array} \quad \beta$$

$$M_D: \begin{array}{|l} \bar{P} \\ \hline \Gamma \end{array} \quad \alpha$$

où α, β dénotent des ensembles de littéraux; Γ dénote un ensemble de clauses.

Le résultat de l'application de \mathbf{M}_C à M sur les littéraux P et \bar{P} est:

1) La paire $\langle M_G, M_D \rangle$ si $\alpha \neq \phi$ $\beta \neq \phi$

2) M_G (M_D) si $\alpha = \phi$ $\beta \neq \phi$ ($\alpha \neq \phi$ $\beta = \phi$)

3) M est simplement insatisfaisable si $\alpha = \beta = \phi$

Le lemme suivant exprime la propriété sur laquelle est basée la MRM.
(Nous insistons sur le fait que l'ensemble vide remplace une matrice simplement contradictoire). Nous appelons c-matrice un ensemble de c-clauses.

LEMME: Soit M une c-matrice et $M_C(M, C, C', L, L') = \{ X \cap M_G, Y \cap M_D \}$.

Tous les chemins de M sont contradictoires ssi tous les chemins de $X \cap M_G$ et tous les chemins de $Y \cap M_D$ sont contradictoires.

PREUVE: La preuve est très simple à faire si l'on considère que dire "tous les chemins de M sont contradictoires" équivaut à dire "l'ensemble de clauses M est insatisfaisable", et que dire " M contient (au moins) un chemin non contradictoire" équivaut à dire " M est satisfaisable".

Soit $M = \{ \{ L \} \cup \alpha \} \cup \{ \{ \bar{L} \} \cup \beta \} \cup \Gamma$

où L (\bar{L}), α (β), Γ dénotent respectivement un littéral, un ensemble de littéraux et un ensemble de clauses.

Il suffit de considérer le cas où $X \cap M_G \neq \phi$ et $Y \cap M_D \neq \phi$, la démonstration des autres cas peuvent être extraites de celle-ci.

$$M_G = \{ \{ L \} \} \cup \{ \beta \} \cup \Gamma \quad \text{et}$$

$$M_D = \{ \alpha \} \cup \{ \{ \bar{L} \} \} \cup \Gamma$$

Il y a deux cas à considérer (le cas $\Gamma = \phi$ est assimilable au cas 2)

1) Γ n'a pas de modèle. M , M_G et M_D n'ont donc pas de modèle parce qu'il faudrait que ses modèles respectifs contiennent un modèle de Γ .

2) Γ a un modèle: m_Γ

En notant

$$L_\alpha \in \alpha$$

$$L_\beta \in \beta$$

tous les modèles de M peuvent être représentés comme:

$$m_1 = \{ L, L_\beta \} \cup m_\Gamma$$

$$m_2 = \{ L_\alpha, \bar{L} \} \cup m_\Gamma$$

$$m_3 = \{ L, L_\alpha, L_\beta \} \cup m_\Gamma$$

$$m_4 = \{ \bar{L}, L_\alpha, L_\beta \} \cup m_\Gamma$$

m_1 et m_3 représentent tous les modèles de M_G ; m_2 et m_4 représentent tous les modèles de M_D . ceci complète la preuve de l'équivalence énoncée.

REMARQUE: A cause de la présence de L dans M_G et de \bar{L} dans M_D , un modèle de M ne peut pas être un modèle de M_G et de M_D .

Le fait que tout modèle de M est, soit un modèle de M_G , soit un modèle de M_D , constitue la preuve du corollaire suivant:

COROLLAIRE: *La règle de RM est correcte.*

Cette règle, qui utilise la propriété qu'une paire de littéraux complémentaires peut rendre complémentaires plus d'un chemin, diminue le nombre de chemins à tester: en effet, si le nombre de littéraux de α et β est $(m-1)$ et $(n-1)$ respectivement, et le nombre de chemins dans Γ est n_Γ , alors le nombre maximum de chemins à tester pour prouver l'insatisfaisabilité d'une matrice est:

$$N_M = m \times n \times n_\Gamma$$

(évidemment, si la matrice est satisfaisable, on arrête le test quand on rencontre un chemin non contradictoire; le nombre de chemins testés peut être donc, dans ce cas, inférieur à N_M).

Le nombre maximum de chemins à tester si l'on applique à la matrice la règle de RM est

$$N_{M_G} + N_{M_D} = (m+n-2) \times n_\Gamma$$

Pour le cas d'application de la RM, où une seule matrice de l'ensemble résultat est non vide, on a $m = 1$, et dans le cas où la matrice est simplement contradictoire $m = 1$ et $n = 1$.

On voit que l'on réduit le nombre de chemins à tester, mais la MRM, comme nous les verrons au § 3.2.2.2, *ne teste pas la contradiction des chemins en les énumérant*: elle applique récursivement la RM jusqu'à ce que l'on obtienne une *matrice simplement insatisfaisable* ou une *matrice à laquelle on ne peut pas appliquer la RM*.

NOTATION: "{ [] }" dénotera tout ensemble de matrices simplement contradictoires, qui seront représentées d'après notre convention par " ϕ ".

Nous passons maintenant à la définition de la cm-déduction (déduction pour les ensembles de c-clauses en utilisant comme règle d'inférence la RM).

REMARQUE: Nous aurions pu définir un opérateur équivalent au R^n de la résolution (§2.2.7.1), mais ce n'est pas nécessaire ici.

Δ Soit M une c-matrice et $T = \{ M_1, M_2, \dots, M_r \}$ un ensemble fini de matrices. Une *cm-déduction de T à partir de M* , notée $M \xrightarrow{cm} T$ est une suite finie B_1, B_2, \dots, B_n d'ensembles finis de matrices tel que:

- 1) $B_1 = \{ M \}$ $B_n = T$
- 2) $\forall i \quad 2 \leq i \leq n$

$B_i = \{ M_1, M_2, \dots, M_K, M_{K+1}, \dots, M_p \}$ et

$\exists j \quad M_j \in B_{i-1} \quad M_C^{(M_j)} = \{ M_K, M_{K+1} \}$ et

$B_i = (B_{i-1} - \{ M_j \}) \cup \{ M_K, M_{K+1} \} \quad \forall$

Δ Une *cm-réfutation* de M est une *cm-déduction* de $\{[]\}$ à partir de M . \forall

3.2.2.1 CORRECTION ET COMPLETEUDE DE LA MRM POUR LES C-MATRICES

La RM pour les c-matrices est *correcte* et *complète*. En outre elle permet la recherche *simultanée* d'une réfutation et d'un modèle d'un ensemble de clauses. Ces deux propriétés sont prouvées dans le théorème suivant:

THEOREME: Soit M une c-matrice $M = \{ C_1, C_2, \dots, C_n \}$

1) M est insatisfaisable ssi $M \vdash \{[]\}$

2) Si $\exists i, j$ tel que $M_j \in B_i$ et qu'il n'est pas possible d'appliquer la RM à M_j , alors M est satisfaisable et l'ensemble

$\{ L_K / L_K \in C_K ; C_K \in M_j \quad 1 \leq K \leq n \}$ est un modèle de M .

PREUVE:

Nous prouvons d'abord le point 2 de l'énoncé du théorème.

Supposons que dans une *cm-déduction* $\exists i, j / M_j \in B_i$ et la RM n'est pas définie sur M_j . C'est-à-dire M_j ne contient que des clauses où tous les littéraux sont purs. M_j contient le même nombre d'éléments que M (l'application de la RM ne modifie pas le nombre de clauses des matrices résultat par rapport à M).

L'ensemble contenant un littéral de chaque clause de M_j est un modèle de M_j , et d'après le dernier lemme et la définition de la *cm-déduction*, est aussi un modèle de M .

Le point 1 de l'énoncé peut être prouvé de la façon suivante:

a) Supposons M insatisfaisable. D'après le lemme antérieur, par application de la RM on ne peut pas obtenir des matrices satisfaisables; en outre, du fait que la RM réduit le nombre de littéraux des clauses et d'après la démonstration ci-dessus, il existe un $n \in \mathbb{N}$ tel que, si B_1, B_2, \dots, B_n sont des ensembles de matrices dans une *cm-déduction*, alors $B_n = \{[]\}$.

b) Si l'on a déduit " $\{[]\}$ " de M , alors en raisonnant par l'absurde on peut supposer M satisfaisable. D'après le point 2 déjà prouvé, on ne peut pas déduire " $\{[]\}$ ". M est donc insatisfaisable.

L'algorithme suivant détecte l'insatisfaisabilité ou la satisfaisabilité (et dans ce cas il fournit un modèle) d'une c-matrice, sa correction et son arrêt sont assurés par le théorème ci-dessus. Il est intéressant de le comparer

avec l'algorithme insat du § 3.2.1.

fonction PRAW (M: matrice): booléen;

* M = { C₁, C₂, ..., C_n } ; ϕ ne représente pas la matrice vide, mais la matrice simplement contradictoire *

début

si M = [.] (M = ϕ) alors

praw \leftarrow vrai

sinon

si M_C n'est pas applicable à M alors

modèle \leftarrow { L_K / L_K \in C_K ; C_K \in M , 1 \leq K \leq n } ;

praw \leftarrow faux

(*) calculer M_C(M) = { X \cap M_G, Y \cap M_D } ;

praw \leftarrow praw (X \cap M_G) et praw (Y \cap M_D)

fsi

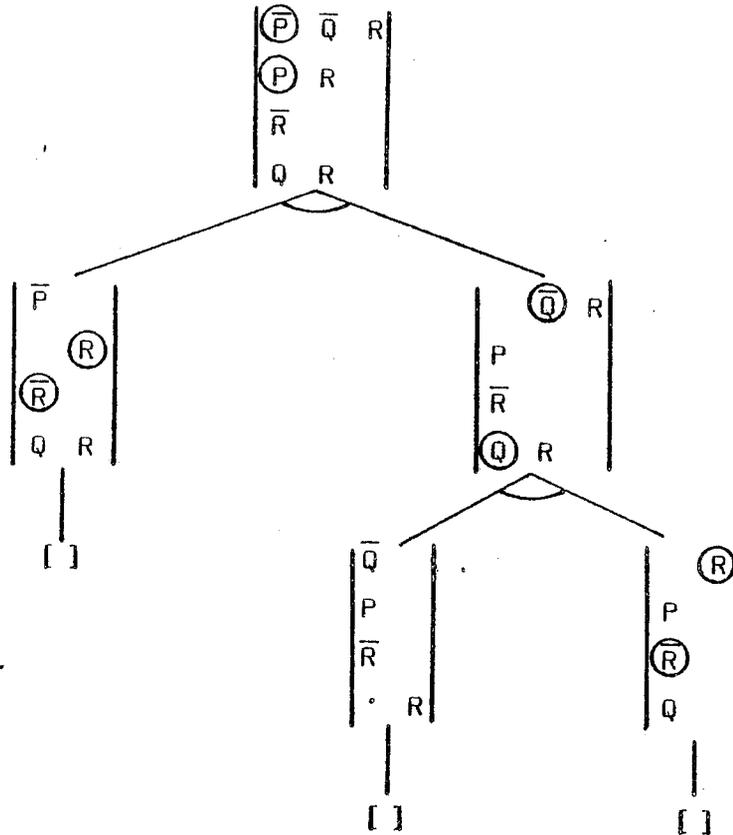
fsi

fin * PRAW * ;

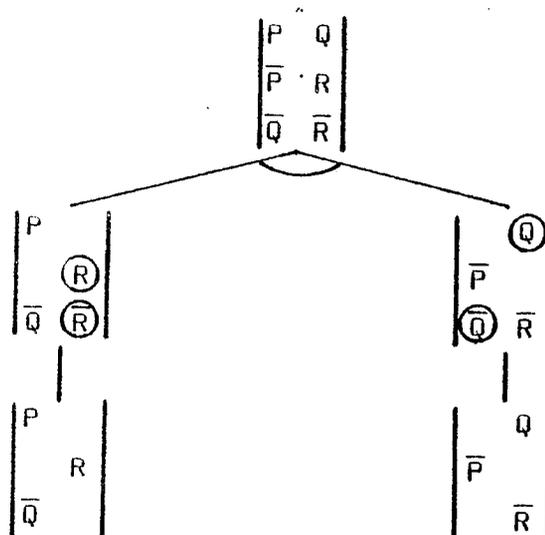
Cet algorithme admet l'incorporation des restrictions telles que le principe de pureté et la subsomption (§ 3.2.2.2); il suffit d'ajouter après la ligne (*) une action dont l'effet est de modifier (en appliquant le raffinement voulu) M_G et M_D.

Les exemples suivants montrent une réfutation d'un ensemble de c-clauses utilisant la RM, ainsi que la construction des modèles d'un ensemble de c-clauses satisfaisable.

EXEMPLE 1: Réfutation d'une c-matrice utilisant la MRM (les littéraux encadrés sont ceux choisis pour l'application de M_C)



EXEMPLE 2: Essai de réfutation d'une c-matrice satisfaisable



La réfutation n'est pas possible, les modèles de M obtenus à partir des feuilles de l'arbre sont $\{ P, \bar{Q}, R \}$, $\{ \bar{P}, Q, \bar{R} \}$.

3.2.2.2 RESTRICTIONS APPLICABLES A LA MRM POUR LES C-MATRICES : CONSEQUENCES

Nous considérons ici trois restrictions, qui s'appliquent couramment à la résolution et qui, comme le fait remarquer Loveland [89], sont simplement des règles d'élimination superposables à tout raffinement de la résolution. Ces 3 restrictions sont : l'application du principe de pureté, l'élimination des tautologies et l'élimination de clauses subsumées.

1) APPLICATION DU PRINCIPE DE PURETE

D'après le théorème du §2.2.6, le principe de pureté ne concerne pas une méthode en particulier, mais la satisfaisabilité d'un ensemble de clauses. Comme nous le verrons au §3.2.2.3, la division de littéraux opérée par la RM favorise l'application de ce principe, et comme nous l'avions avancé au §3.2.2.1 il aurait pu être incorporé à l'algorithme "praw".

2) ELIMINATION DES TAUTOLOGIES

Dans l'application de la méthode de résolution à un ensemble de c-clauses, on peut engendrer des nouvelles clauses qui sont des tautologies. Par exemple, les résolventes de AVB et de $\bar{A}\bar{B}$ sont $A\bar{A}$ et $B\bar{B}$. D'une façon générale, si on élimine les tautologies engendrées au cours d'une RC-déduction la complétude de la méthode n'est pas compromise. Evidemment, l'application de cette règle d'élimination diminue l'espace de recherche simplifiant les CR-déductions (ou CR-réfutations).

En ce qui concerne la RM : l'application de cette règle *divise* les clauses et n'ajoute pas de nouvelle clause. Pour la RM pour les c-matrices cette règle n'est donc d'aucune utilité. Notre conclusion sur ce point se réduit donc à la constatation suivante :

La RM n'engendre pas de tautologie (comparer avec la méthode de Davis et Putnam §2.2.3.2).

Cette vérification permet de mettre en valeur une des caractéristiques importantes de la RM. En effet l'élimination des tautologies dans la résolution ne va pas sans poser de problèmes, le problème étant de savoir *quand* les éliminer.

La complétude d'une stratégie utilisant l'élimination des tautologies dépend de *la façon* dont les tautologies sont éliminées (voir p. ex. [27], [89]). Dans [89] il est montré une restriction de la résolution pour laquelle on perd la

complétude, si les tautologies sont éliminées dès qu'elles sont engendrées.

3) ELIMINATION DES CLAUSES SUBSUMEES

Si l'on applique la définition de subsumption donnée au §2.1.1.1 à des c-clauses, elle revient simplement à dire qu'une c-clause A subsume une c-clause B ssi A est contenue dans B.

On avait déjà vu au §2.2.3.2 avec la méthode de Davis et Putnam que l'on pouvait éliminer les clauses qui étaient subsumées par d'autres clauses. Ce même principe peut évidemment être appliqué dans la RM. Etant donné que la règle de RM isole des littéraux, elle favorise l'application de cette restriction.

En particulier, l'utilisation de cette restriction permet de simuler la méthode de Davis et Putnam (l'observation que la méthode de Davis et Putnam peut être simulée par la RM est aussi faite dans [13]).

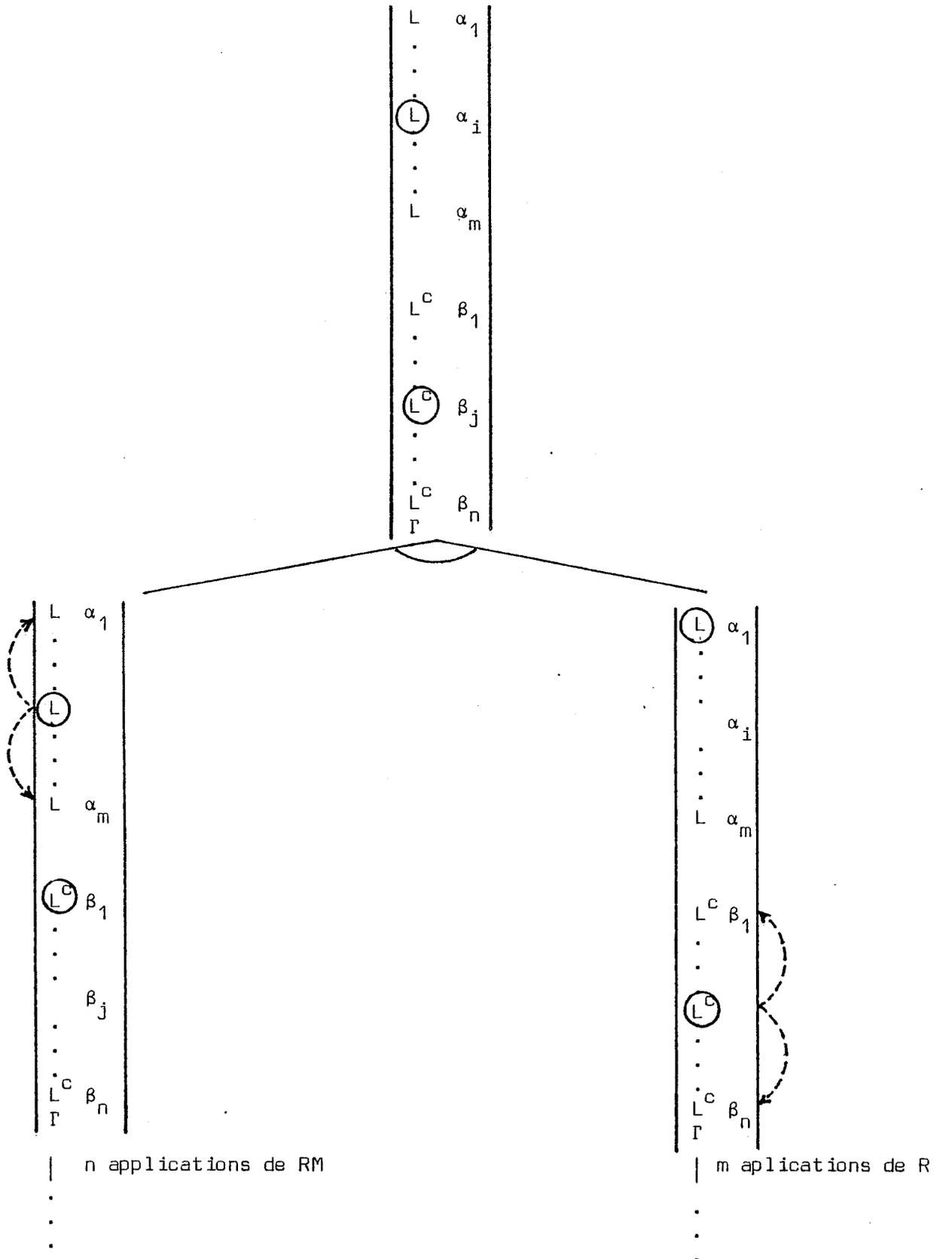
Nous indiquons graphiquement le principe à employer (une démonstration formelle en est triviale) pour simuler la règle 3 de Davis et Putnam. Dans le dessin ci-dessous :

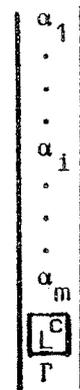
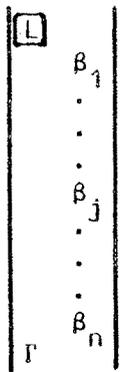
L et L^C dénotent des littéraux complémentaires.

α_i et β_j des ensembles de littéraux ($1 \leq i \leq m$; $1 \leq j \leq n$)

$L \not\subset \alpha_i$; $L^C \not\subset \alpha_i$; $L \not\subset \beta_j$; $L^C \not\subset \beta_j$ ($1 \leq i \leq m$; $1 \leq j \leq n$)

Γ : ensemble de clauses ($\forall C \in \Gamma$; $L \not\subset C$; $L^C \not\subset C$)





---> : subsume

$\boxed{}$: littéral pur (pouvant donc être éliminé)

La simulation des règles de Davis et Putnam par la méthode de RM peut être faite comme suit :

REGLE 1 : RM + subsomption + principe de pureté

REGLE 2 : Principe de pureté

REGLE 3 : RM + subsomption + principe de pureté (voir dessin ci-dessus)

3.2.2.3 EXTENSION DU PRINCIPE DE RM : L'HYPERREDUCTION MATRICIELLE AVEC SUBSOMPTION (SH-RM)

Nous pouvons étendre le principe de RM de façon à obtenir une règle similaire à l'hypermérisation.

Dans l'exemple suivant, il est clair que l'on peut tester l'insatisfaisabilité de la matrice sans appliquer effectivement 3 fois la RM.

EXEMPLE :



Le principe d'hyperreduction matricielle (HRM) est une généralisation de cette propriété.

Profitant du fait que la RM *isole* des littéraux et que ceci rend *beaucoup plus probable* l'élimination de clauses subsumées, nous combinerons la règle de HRM avec la règle d'élimination de clauses subsumées et nous obtiendrons une règle qui sera appelée SH-RM.

(On notera ici l'intérêt d'avoir choisi $P_\phi(M^*)$ comme codomaine de la fonction M_C (§3.2.2)).

Nous spécifions la SH-RM en forme algorithmique.

REGLE DE SH-RM:

Soit M une matrice

- 1) Fixer un ordre parmi tous les littéraux des clauses de M .
- 2) Choisir une clause $C = \beta \bigcup \{L_1, L_2, \dots, L_n\}$ tel que pour chaque L_i ($1 \leq i \leq n$) existe une clause dans M de la forme $C_i = \alpha_i \bigcup \{L_i^c\}$ (chaque L_i^c appartenant à une clause différente).

(Nous supposons $\alpha_i \neq \phi$ ($1 \leq i \leq n$) et $\beta \neq \phi$, mais la modification pour le cas $\alpha_i = \phi$ ou $\beta = \phi$ ne présente pas de difficulté).

L'application de la SH-RM donne comme résultat $n+1$ matrices où M_i ($1 \leq i \leq n+1$) est obtenue à partir de M de la façon suivante:

- a) *Éliminer toutes les clauses contenant $L_1^c, L_2^c, \dots, L_{i-1}^c, L_i$.*
- b) *Éliminer des clauses restantes $L_1, L_2, \dots, L_{i-1}, L_i^c$
(Si dans l'application de la règle b on élimine tous les littéraux d'une clause ou l'on élimine toutes les clauses d'une matrice par application de la règle a, alors $M_i = []$).*

En appliquant le fait que la RM est une règle correcte, le principe d'élimination de clauses subsumées et le principe de pureté il est facile de prouver que:

La HS-RM est une règle correcte.

Une HS-RM déduction peut être définie d'une façon similaire à une RM-déduction.

LEMME: *Une matrice M est insatisfaisable ssi $M \xrightarrow{HS-RM} []$.*

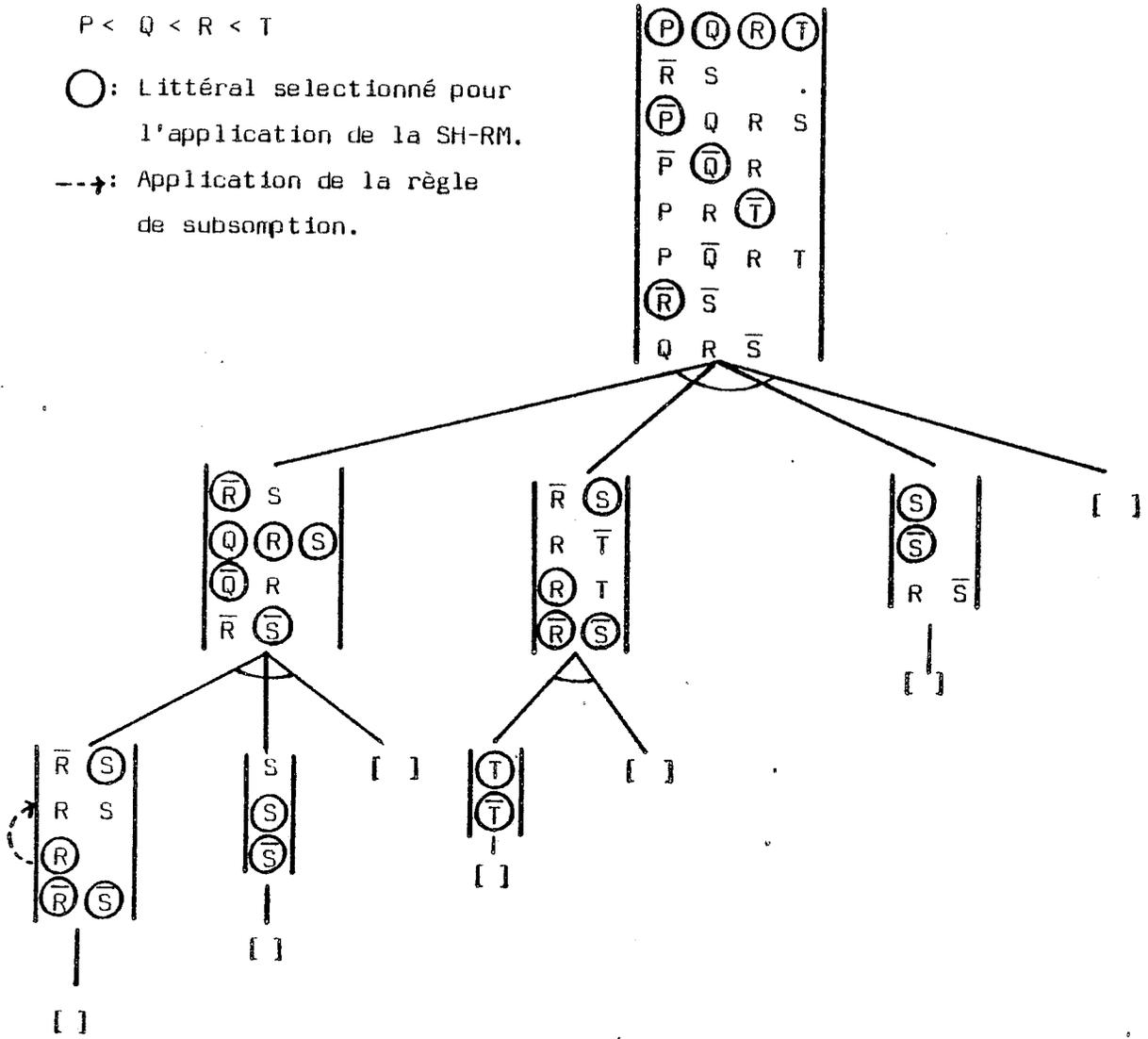
PREUVE: Immédiate par application répétée de la RM et des principes de pureté et de subsumption. ■

EXEMPLE:

$$P < Q < R < T$$

○: Littéral sélectionné pour l'application de la SH-RM.

-->: Application de la règle de subsumption.



3.2.3 LA RÉDUCTION MATRICIELLE DANS LE CAS GÉNÉRAL

3.2.3.1 CONDITIONS DE SUBSTITUTION ET SUBSTITUTION UNIFICATRICE

Ces deux concepts nous seront nécessaires pour la définition de la MRM dans le cas général. Ils sont présentés séparément du reste, parce qu'en les considérant ainsi nous suggérons la gp-abstraction et le choix de l'algorithme d'unification (voir Chapitre 4).

Δ L'ensemble de deux littéraux, dont le symbole de prédicat est le même et qui diffèrent dans le signe :

$$\{P(t_1, \dots, t_n), \bar{P}(t'_1, \dots, t'_n)\},$$

est appelé une *contradiction possible* ; l'ensemble d'équations dans les termes

$$\{t_1=t'_1, \dots, t_n=t'_n\}$$

est appelé la *condition de substitution correspondante* pour cette contradiction possible. ▽

On généralise cette définition à un ensemble de plus de 2 littéraux

Δ Un ensemble fini de littéraux dont le symbole de prédicat et le signe sont les mêmes

$$\{P(t_1^1, \dots, t_n^1), P(t_1^2, \dots, t_n^2), \dots, P(t_1^p, \dots, t_n^p)\}$$

est dit un *singleton possible* ; l'ensemble d'équations dans les termes

$$\{t_1^1=t_1^2, \dots, t_n^1=t_n^2, \dots, t_1^1=t_1^p, \dots, t_n^1=t_n^p\}$$

est appelé la *condition de substitution correspondante*.

(Il est évident, d'après la transitivité de l'égalité, que l'on aurait pu prendre t_1^j ($2 \leq i \leq n$; $1 \leq i \leq p$) comme terme gauche des équations). ▽

REMARQUE : Il est clair que si la condition de substitution a une solution, l'ensemble de littéraux est un ensemble unifiable (voir §2.2.7.1).

Dans tout ce qui suit "condition de substitution" sera synonyme d'"ensemble fini d'équations dans les termes".

Nous aurons besoin, sans la définition de déduction utilisant la RM, de la notion de *compatibilité*. Elle peut être définie pour les conditions de substitution ou pour les substitutions (les deux définitions correspondent au même concept et sont équivalentes).

La *compatibilité* de n conditions de substitution est une application :

$$\mathbb{C} : \text{ECS}^n \longrightarrow \{V, F\}$$

(où ECS : ensemble de toutes les conditions de substitution), définie comme suit :

$$\mathbb{C}(CS_1, \dots, CS_n) = \begin{cases} V & \text{si } CS_1 \cup CS_2 \cup \dots \cup CS_n \text{ a une solution.} \\ F & \text{sinon.} \end{cases}$$

Pour la définition de compatibilité de substitutions nous suivons [135]. Cette opération doit combiner l'action de deux substitutions sans donner priorité à l'une ou l'autre, et elle est *différente de la composition de substitutions*.

On généralise la définition d'unificateur le plus général de la façon suivante :

Δ Une substitution γ est la *substitution la plus générale satisfaisant une propriété*, ssi elle satisfait la propriété et pour toute substitution σ satisfaisant la propriété il existe une substitution θ telle que $\sigma = \gamma \circ \theta$. \forall

Δ La *composition unificatrice de deux substitutions* est une application :

$$\mathcal{U} : \text{ES} \times \text{ES} \longrightarrow \text{ES}$$

où ES : ensemble de toutes les substitutions
définie comme suit :

$$\begin{aligned} \mathcal{U}(\alpha, \beta) &= \gamma \\ \text{ssi } \exists \gamma \in \text{ES} / \\ \gamma \circ \alpha &= \alpha \circ \gamma = \gamma = \gamma \circ \beta = \beta \circ \gamma. \quad \forall \end{aligned}$$

EXEMPLE :

$$\alpha = \{ \langle x, a \rangle \} ; \beta = \{ \langle x, b \rangle \} ; \gamma = \{ \langle x, f(y) \rangle, \langle z, f(a) \rangle \} ; \delta = \{ \langle z, f(y) \rangle, \langle y, a \rangle \}$$

$\mathcal{U}(\alpha, \beta)$ non définie

$$\mathcal{U}(\gamma, \delta) = \{ \langle x, f(a) \rangle, \langle y, a \rangle, \langle z, f(a) \rangle \}$$

REMARQUE : Il est facile de vérifier qu'en général $(\alpha, \beta) \neq \alpha \circ \beta$

Un algorithme qui calcule le résultat de l'opération \mathcal{U} a été défini dans [154].

Δ La *compatibilité* de n ($n \geq 2$) substitutions est une fonction booléenne :

$$\mathbb{C} : (\text{ES})^n \longrightarrow \{V, F\}$$

ES : ensemble de substitutions

$$\mathbb{C}(\alpha_1, \dots, \alpha_n) = \begin{cases} V & \text{si } \mathcal{U}(\alpha_1, \mathcal{U}(\alpha_2, \dots, \mathcal{U}(\alpha_{n-1}, \alpha_n) \dots)) \text{ est définie} \\ F & \text{sinon. } \forall \end{cases}$$

Comme nous le verrons au Chapitre 4, nous avons choisi pour le test de compatibilité de substitutions l'approche induite par la première de ces définitions. L'algorithme utilisé pour résoudre le système d'équations est celui d'unification de Huet. Cet algorithme contient l'opération \llcorner et il n'y a pas de calcul effectif de substitutions, tant que l'on n'est pas assuré de la compatibilité de toutes les conditions de substitution engendrées au cours d'une démonstration.

Nous pouvons maintenant énoncer la règle de RM et la notion de déduction dans le cas général.

3.2.3.2 LA METHODE DE RM POUR LES MATRICES

Comme pour la RM pour les c-clauses nous appelons matrices les ensembles de clauses. Nous acceptons conventionnellement qu'une matrice représente un ensemble de chemins dont il faut prouver la contradiction pour prouver l'insatisfaisabilité de la matrice correspondante. Avec cette convention la matrice vide correspond à un ensemble de clauses insatisfaisable.

Δ Soit

L^* : ensemble de tous les littéraux

$C^* = P_f(L^*)$: ensemble de parties finies de L^* (les clauses)

$M^* = P_f(C^*)$: ensemble de parties finies de C^* (les matrices)

La règle de réduction matricielle (RM) est une application

$$M : M^* \times (C^*)^2 \times (L^*)^2 \longrightarrow P_f(M^*)$$

si les conditions (a), (b) et (c) ci-dessous sont satisfaites:

$$(a) \quad M \in M^*; C_i \in M; C_j \in M; L_i \in C_i; L_j \in C_j$$

$$(b) \quad EV(C_i) \cap EV(C_j) = \phi$$

(c) $\{L_i, L_j\}$ est une contradiction possible et la condition de substitution correspondante a une solution σ (autrement dit $\sigma\{L_i, L_j^c\}$ est un singleton) la fonction est définie de la façon suivante:

$$M(M, C_i, C_j, L_i, L_j) = \{X \cap M_G, Y \cap M_D\}$$

où

$$M_G = (M - \{C_i\} - \{C_j\}) \cup \{\sigma L_i\} \cup \sigma\{C_j - \{L_j\}\}$$

$$M_D = \eta(M - \{C_i\} - \{C_j\}) \cup \{\sigma L_j\} \cup \sigma\{C_i - \{L_i\}\}$$

η : substitution de renommage ou identité partout

$$X = \begin{cases} \phi & \text{si } C_j = \{ L_j \} \\ M_G & \text{sinon} \end{cases}$$

$$Y = \begin{cases} \phi & \text{si } C_i = \{ L_i \} \\ M_D & \text{sinon} \end{cases}$$

si l'une quelconque des conditions (a), (b) ou (c) n'est pas vérifiée alors la fonction n'est pas définie. \forall

La règle que nous venons de définir aurait pu être appelée, par analogie avec la résolution, "RM binaire" (c'est d'ailleurs celle qui a été définie par Prawitz). On peut également définir une règle de RM faisant intervenir des *ensembles de littéraux* de clauses. Cependant les deux règles sont équivalentes dans ce sens qu'à toute déduction utilisant une des règles on peut faire correspondre une autre déduction utilisant l'autre règle (voir théorème ci-après). La notion de factorisation n'est donc pas nécessaire dans la RM.

Δ Soit

L^* : ensemble de tous les littéraux

$C^* = P_f(L^*)$ ensemble de parties finies de L^* (les clauses)

$M^* = P_f(C^*)$ ensemble de parties finies de C^* (les matrices)

la règle de réduction matricielle non binaire (RMNB) est une application:

$$M_N : M^* \times (C^*)^4 \longrightarrow P_f(M^*)$$

si les conditions (a), (b) et (c) ci-dessous sont satisfaites:

(a) $M \in M^*$; $C_i \in M$; $C_j \in M$; $D_i \subset C_i$; $D_j \subset C_j$; $D_i \neq \phi$; $D_j \neq \phi$

(b) D_i et D_j sont de singletons possibles, CS_1 et CS_2 sont les conditions de substitution correspondantes

(c) $CS_1 \cup CS_2$ a une solution σ

la fonction est définie de la façon suivante:

$$M_N(M, C_i, C_j, D_i, D_j) = \{ X \cap M_G, Y \cap M_D \}$$

où

$$M_G = (M - \{ C_i \} - \{ C_j \}) \cup \{ \sigma D_i \} \cup \{ \sigma (C_j - D_j) \}$$

$$M_D = \eta (M - \{ C_i \} - \{ C_j \}) \cup \{ \sigma D_j \} \cup \{ \sigma (C_i - D_i) \}$$

η : substitution de renommage ou identité partout

$$X = \begin{cases} \phi & \text{si } C_j = D_j \\ M_G & \text{sinon} \end{cases}$$

$$\gamma = \begin{cases} \phi & \text{si } C_i = D_i \\ M_D & \text{sinon} \end{cases}$$

si l'une quelconque des conditions (a), (b) ou (c) n'est pas satisfaite alors la fonction n'est pas définie. \forall

On définit une déduction et une réfutation pour les matrices d'une façon similaire à celle du § 3.2.2 en ajoutant une condition de compatibilité des substitutions correspondants à l'application de l'opération **M** :

Δ Soit M une matrice et $T = \{ M_1, M_2, \dots, M_n \}$ un ensemble fini de matrices. Une m -déduction de T à partir de M , notée $M \xrightarrow{m} T$ est une suite finie B_1, B_2, \dots, B_n d'ensembles finis de matrices tel que:

- 1) $B_1 = \{ M \}$, $B_n = T$
- 2) $\forall i \quad 2 \leq i \leq n$

$$B_i = \{ M_1, M_2, \dots, M_K, M_{K+1}, \dots, M_P \},$$

$\exists j \quad M_j \in B_{i-1} \quad \mathbf{M}(M_j) = \{ M_K, M_{K+1} \} \quad \sigma_{i-1}$ est la substitution correspondante à l'application de **M**

$$\mathbf{C}(\sigma_{i-1}, \sigma_1, \sigma_2, \dots, \sigma_{i-2}) = V$$

(ou $\mathbf{C}(\sigma_{i-1}, \gamma) = V$, avec $\gamma = \mathbf{U}(\sigma_1, \mathbf{U}(\sigma_2, \dots, \mathbf{U}(\sigma_{i-3}, \sigma_{i-2}))) \dots \forall$

Pour prouver la correction et la complétude de la RM on utilise le lemme suivant

LEMME: Soit M une matrice $\gamma_1, \gamma_2, \gamma_3$ des substitutions sans variables (§ 2.1.1.1) telles que $\gamma_1 = \mathbf{U}(\gamma_2, \gamma_3)$ et

$$\gamma_1 = \delta_1 \circ \sigma \quad \gamma_2 = \delta_2 \circ \sigma \quad \gamma_3 = \delta_3 \circ \sigma$$

où σ est la substitution correspondante à une application de la RM à M :

$$\mathbf{M}(M, C_i, C_j, L_i, L_j) = \{ X \cap M_G, Y \cap M_D \}$$

Tous les chemins de γM sont contradictoires ssi tous les chemins de $\gamma_1 (X \cap M_G)$ et tous les chemins de $\gamma_2 (Y \cap M_D)$ sont contradictoires.

PREUVE: D'après les définitions de RM, de **M** et du Lemme correspondant pour les c-matrices. ■

La MRM pour les matrices est correcte et complète :

THEOREME: Une matrice M est insatisfaisable ssi il existe une m -réfutation pour un ensemble de variantes de clauses de M .

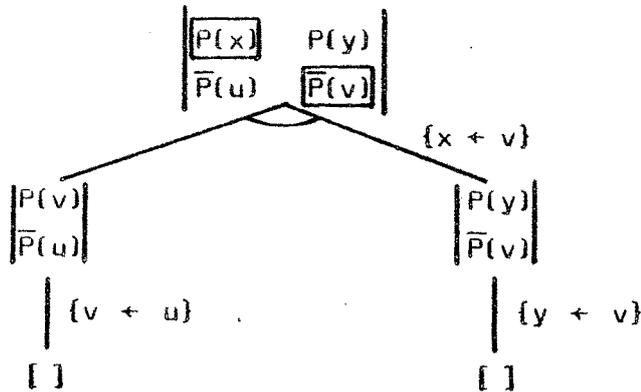
PREUVE: En utilisant le lemme ci-dessous, la définition de m-déduction et la théorie de Herbrand. ■

La MRM ("binaire") n'a pas besoin de factorisation:

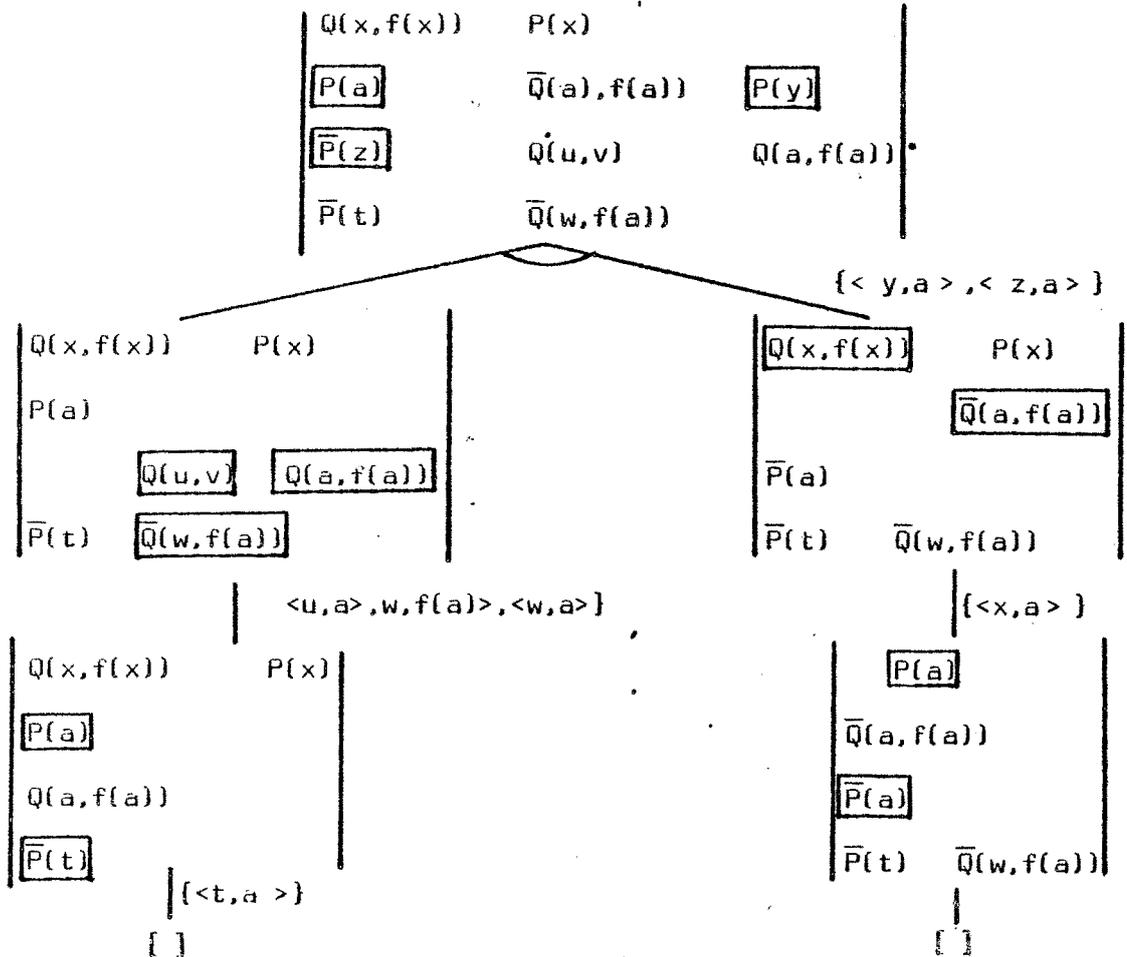
THEOREME: A toute m-déduction on peut faire correspondre une m-déduction ("non binaire") et réciproquement.

PREUVE: D'après la définition de déduction et de celle de RM "binaire" et "non binaire". ■

EXEMPLE 1: Réfutation de la matrice de l'exemple de § 2.2.7.2

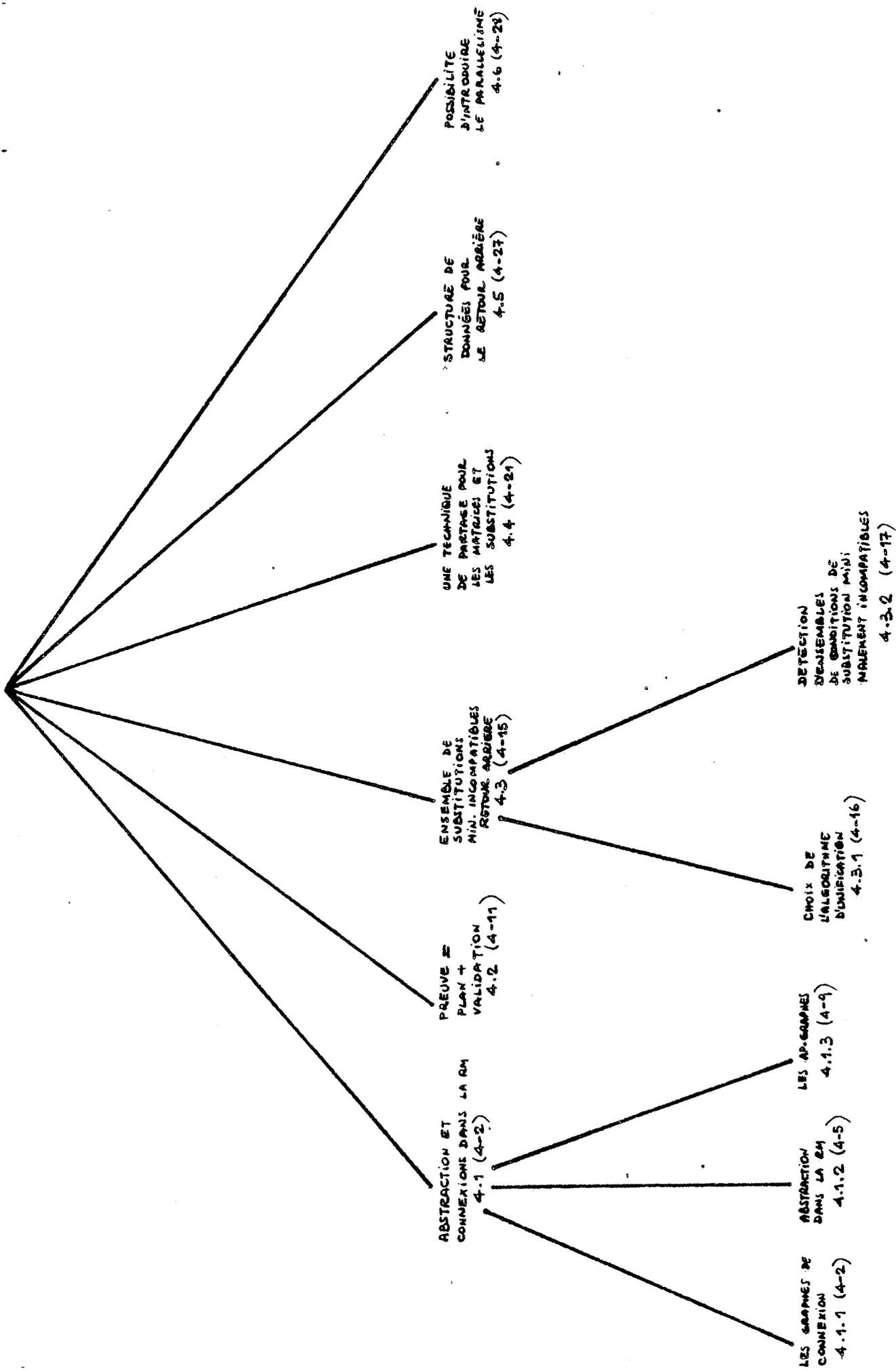


EXEMPLE 2: Réfutation avec RM non binaire



CHAPITRE 4

**ABSTRACTION, PLANS, VALIDATIONS, RETOUR ARRIERE ET PARTAGE DE
STRUCTURE DANS LA REDUCTION MATRICIELLE**



POSSIBILITE
D'INTRODUIRE
LE PARALLELISME
4.6 (4-28)

STRUCTURE DE
DONNEES POUR
LE RETOUR ARRISERE
4.5 (4-27)

UNE TECHNIQUE
DE PRETRAIEMENT POUR
LES MATRICES ET
LES SUBSTITUTIONS
4.4 (4-21)

ENSEMBLE DE
SUBSTITUTIONS
MIN. INCOMPATIBLES
RETOUR ARRISERE
4.3 (4-15)

PREUVE ET
PLAN +
VALIDATION
4.2 (4-11)

ABSTRACTION ET
CONNEXIONS DANS LA RM
4.1 (4-2)

DETECTION
D'ENSEMBLES
DE SUBSTITUTIONS DE
SUBSTITUTIONS MIN.
INCOMPATIBLES
4.3.2 (4-17)

CHOIX DE
L'ALGORITHME
D'UNIFICATION
4.3.1 (4-16)

LES AP-GRAMMES
4.1.3 (4-9)

ABSTRACTION
DANS LA RM
4.1.2 (4-5)

LES GRAMMES DE
CONNEXION
4.1.1 (4-2)

Nous pouvons résumer en deux points les critiques qui ont été adressées à la MRM, et qui ont certainement influencé l'attitude vis-à-vis d'elle dans un premier temps :

1) Le travail de Loveland [88] (travail qui date de 1972). Il y est montré que la RM correspond ("presque toujours") à des résolutions linéaires d'une certaine forme.

Il est intéressant de citer les points principaux de cet article pour pouvoir placer ces conclusions dans le bon contexte (et en même temps de les relativiser) :

- a) Par rapport aux méthodes les plus importantes basées sur le théorème de Herbrand et connues à l'époque, la question suivante y est posée : "Faut-il continuer les recherches pour des procédures autres que la résolution ?".
- b) Il reconnaît l'importance des implémentations, ainsi que la difficulté de les comparer.
- c) Le souhait de l'existence d'une théorie générale de l'efficacité y est émis.
- d) Une alternative à ce dernier point est *d'établir un rapport* entre procédures différentes de façon à estimer *leurs performances relatives*. Cette option a été celle de l'auteur et elle l'a conduit à la conclusion citée dans le point 1.
- e) La conclusion générale de l'article est "qu'en gros" on peut considérer les procédures autres que la résolution, étudiées dans l'article, comme des formes de la résolution linéaire.

Nous pensons qu'on a répondu aux points a et e depuis (voir par exemple [14], [17]) et qu'il y a un consensus général sur les points b et c. Reste à considérer le point d. Le fait de vouloir absolument voir la RM du "point de vue de la résolution" était une façon de faire qui ignorait les caractéristiques propres de la RM, en particulier ses possibilités de généralisation (voir travaux de Bibel [14]).

Au chapitre 3, nous avons montré certaines caractéristiques de la RM indépendamment des stratégies et des raffinements qu'on peut lui appliquer. Ce point qui peut paraître sans importance en a cependant une : il suffit pour s'en convaincre de considérer deux exemples. Le premier est la difficulté de savoir *quand* utiliser la factorisation dans la méthode de résolution (pour éviter de contribuer à une explosion combinatoire) [145]. Le deuxième est le problème d'élimination des clauses subsumées (voir [89]). Il faut y ajouter les problèmes

posés par la preuve de complétude de certaines stratégies.

2) Du fait que la méthode manipule des ensembles de clauses, et non des clauses, la deuxième critique était dirigée contre *la place* mémoire qu'elle exige (voir [134]).

Dans ce chapitre, nous combinerons deux concepts avec la RM ; l'utilisation de ces deux concepts aura comme conséquence naturelle une technique de partage de structure qui répond à la deuxième critique.

Les deux concepts sont celui de "graphes de connexion", et celui d'"abstraction". Ils sont fusionnés dans un seul : ce que nous appelons les "ap-graphes" ou les "gp-abstractions".

La technique employée sépare, dans la recherche d'une preuve, la partie purement déductive de l'algorithme d'unification. Une idée similaire a été utilisée par Huet [67].

Nous utiliserons cette caractéristique pour pouvoir faire un retour arrière "non aveugle", en utilisant les échecs de l'unification.

4.1 ABSTRACTION ET CONNEXIONS DANS LA RÉDUCTION MATRICIELLE : LES AP-GRAPHES

Nous rappelons brièvement la méthode des graphes de connexion due à Kowalski ([75], [76]).

4.1.1 LES GRAPHES DE CONNEXION

Dans cette méthode l'ensemble de clauses est transformé en un graphe dont les noeuds sont les littéraux des clauses. Les arcs relient les littéraux qui peuvent être rendus complémentaires en appliquant à chaque littéral la substitution qui étiquette l'arc. On ne travaille qu'avec des ensembles ne contenant pas de littéraux purs (avec la terminologie du §2.2.6 avec des "ensembles de paires liées"). La règle de résolution est appliquée pour transformer le graphe, le but étant de trouver la clause vide, c'est-à-dire un arc dont les noeuds qu'il relie sont les seuls composants de leurs clauses respectives.

REMARQUE : Les arcs du graphe sont appelés "liens" ("links"). Ils permettent de caractériser *toutes* les résolvantes possibles qui peuvent être obtenues à partir de l'ensemble initial de clauses (et des variantes de ces clauses). En particulier, pour *ne pas perdre la complétude*, il faut considérer les liens entre des littéraux qui peuvent être rendus complémentaires, dans la *même clause* (ce qui donnerait une autorésolvante). Cette exigence est contenue dans le théorème de Herbrand (voir aussi remarque du §2.2.5).

Nous ne donnons pas ici une définition mathématique des graphes de connexion, nous le ferons pour les ap-graphes. L'algorithme permettant de détecter l'insatisfaisabilité d'un ensemble de clauses, utilisant la méthode de connexion est le suivant [75].

- 1 - Éliminer les clauses contenant des littéraux purs, ainsi que leurs liens associés jusqu'à ce que toutes les clauses contenant des littéraux purs aient été éliminées.
- 2 - Sélectionner un lien, l'éliminer et ajouter au graphe la résolvente et les nouveaux liens correspondant à la résolvente.

REMARQUE : Des restrictions comme par exemple l'élimination des tautologies peuvent être incorporées. La factorisation est *nécessaire* à la complétude.

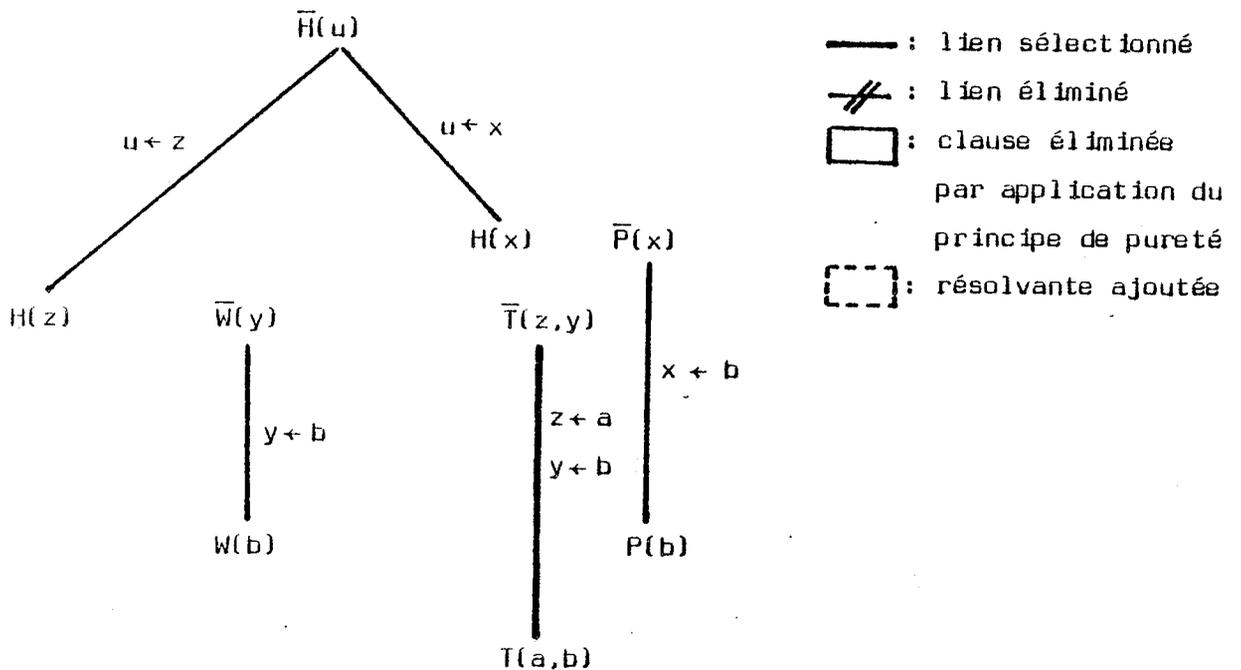
Voici un exemple de réfutation utilisant cette méthode

EXEMPLE [75] :

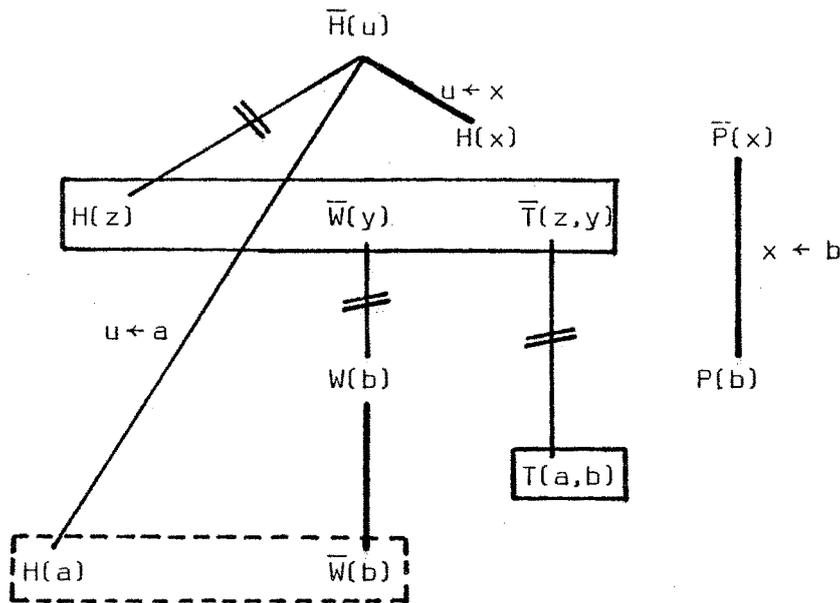
A l'ensemble de clauses :

- $\bar{H}(u)$
- $H(x) \bar{P}(x)$
- $H(z) \bar{W}(y) \bar{T}(z,y)$
- $P(b) W(b)$
- $T(a,b)$

On fera correspondre le graphe initial :



qui sera transformé en :



En appliquant la résolution sur les liens $\bar{H}(u), H(x)$; $W(b), \bar{W}(b)$; $\bar{P}(x), P(b)$ nous arrivons à ce que la résolution sur ce dernier lien produit la clause vide (nous ne donnons pas le détail, le dessin ci-dessus contient toutes les opérations possibles dans le graphe).

Nous passons maintenant au deuxième concept : celui d'abstraction (l'abstraction propositionnelle -voir §4.1.2- était implicite dans [130], [131] et a été traitée dans [24]).

Un concept similaire a été utilisé par Plaisted [108], [109] et il a appliqué cette idée à la démonstration de théorèmes par résolution.

L'idée qui nous a amené à l'abstraction (et qui est d'ailleurs semblable à celle exprimée dans [108], [109]) est basée sur le principe suivant : un problème P peut être *décomposé* en deux autres problèmes P_1 et P_2 ; P_1 étant un problème *plus simple* (ou *analogue* à un autre problème que l'on sait résoudre), et qui peut correspondre à une *infinité* de problèmes avec la *même structure* que P . P_2 étant un problème qui "particularise" (si possible) la solution proposée par P_1 pour la *classe* de problèmes à laquelle appartient P . Si P a une solution alors P_1 et P_2 en ont une. Si P_1 et P_2 ont une solution alors P en a une (dans ce sens P_1 et P_2 sont des *sousproblèmes* de P). Nous présentons d'abord la définition d'abstraction quelques *abstractions syntaxiques* de Plaisted. On parlera seulement de celles qui nous intéressent dans un but de comparaison avec l'abstraction que nous proposons dans les ap-graphes (§4.1.3).

L'idée de base est de construire une *esquisse* de preuve (que nous appellerons "plan") et de *préciser les détails* après. L'être humain confronté à la difficulté de démontrer un théorème agit souvent de cette façon.

L'abstraction peut être appliquée à *plusieurs niveaux* (dans notre cas ce sera simplement, par exemple, la suppression de tous les arguments, de tous sauf 1, ..., d'aucun argument). Une abstraction des arguments pour un littéral correspond à l'idée de "ion" dans [73]. ($P(-_1, -_2, \dots, -_n)$ est un ion du littéral $P(t_1, t_2, \dots, t_n)$).

Plaisted considère l'abstraction comme une classe particulière d'analogie. Pour nous ce sera *l'analogie par excellence* puisqu'elle correspond, via une représentation standard des problèmes, à *l'identité de structure*.

4.1.2 ABSTRACTIONS POUR LES CLAUSES

Les abstractions sont des applications qui (comme le fait remarquer Plaisted [109]) convertissent un ensemble de clauses A dans un ensemble de clauses *plus simple* B (pour nous ce sera *plus simple* ou *analogue*). Plaisted considère des abstractions syntaxiques et des abstractions sémantiques. Nous ne considérons que des abstractions syntaxiques. (Il introduit des "m-clauses" qui sont des multiensembles de littéraux. Pour nous les m-clauses sont une conséquence naturelle de l'abstraction proposée).

Δ Soit

L^* : ensemble de tous les littéraux

une application

$$A : L^* \longrightarrow L^*$$

est dite une *abstraction pour les littéraux* ssi

- 1) $A(\bar{L}) = \overline{A(L)}$
- 2) Le signe de $A(L)$ est le même que celui de L .
- 3) Si $L=P(\dots)$, $L'=Q(\dots)$ alors $A(L) \neq A(L')$ (noter que nous n'exigeons pas de A qu'elle soit injective).
- 4) Si $L=P(t_1, \dots, t_n)$ et $L'=P(t'_1, \dots, t'_n)$ alors le symbole de prédicat que A assigne à L et à L' doit être le même. \forall

REMARQUE : Ces conditions sont plus restrictives que celles de Plaisted.

EXEMPLE :

$P(x,y)$ est une abstraction de $P(a,b)$
 Q " " " " $Q(x,a,z,b,c)$
 $Q(a,b,c)$ " " " " $Q(b,c,a)$
 $\bar{Q}(a)$ " " " " $\bar{Q}(a,b,c)$
 $R(x,y,z)$ " " " " $R(x)$

REMARQUE 1 : Dans le dernier exemple on augmente le nombre d'arguments du littéral (Plaisted n'admet pas ce type d'abstraction), ce qui serait intuitivement contraire à l'idée d'abstraction, mais l'on peut considérer que le fait d'avoir résolu un "problème plus complexe" peut aider dans la solution d'un problème plus simple.

REMARQUE 2 : L'abstraction n'est pas orientée vers une règle d'inférence particulière.

On généralise la notion d'abstraction aux clauses et aux ensembles de clauses :

Δ Soit

L^* : ensemble de tous les littéraux

$C^* = P_f(L^*)$ ensemble des parties finies de L^* : les clauses.

$S = P_f(C^*)$ ensemble des parties finies de C^* : les ensembles de clauses.

A une abstraction pour les littéraux

Une abstraction pour les clauses est une application

$$A_C : C^* \longrightarrow C^*$$

définie comme suit :

si $C = \{L_1, L_2, \dots, L_n\}$ alors $A_C(C) = \{A(L_1), A(L_2), \dots, A(L_n)\}$

et une abstraction pour les ensembles de clauses est une application :

$$A_E : S^* \longrightarrow S^*$$

définie comme suit :

si $S = \{C_1, C_2, \dots, C_n\}$ alors

$$A_E(S) = \{A_C(C_1), A_C(C_2), \dots, A_C(C_n)\} \cdot \forall$$

Pour les ensembles de clauses, une abstraction intéressante (puisqu'elle ne "change rien" à la structure du problème) est une abstraction que nous appelons simplement "transformation équivalente"

Δ Soit S un ensemble de clauses, et soit A une abstraction pour les littéraux de S avec la propriété suivante :

$\forall L \in C \quad \forall C \in S$

$$A(L(t_1, t_2, \dots, t_n)) = L'(t'_1, t'_2, \dots, t'_n)$$

où t'_1, t'_2, \dots, t'_n est une permutation de t_1, t_2, \dots, t_n (la même pour tous les littéraux).

Cette abstraction est dite *transformation équivalente* de l'ensemble de clauses S . \forall

Nous n'avons pas défini formellement les abstractions pour les termes (fonctionnels) et nous en parlons ci-dessous (dans 3, 4, 5). La signification découle du contexte.

Nous pouvons comparer les abstractions qui viennent d'être définies avec certaines abstractions syntaxiques de Plaisted.

L'abstraction pour les clauses que nous avons définie correspond aux abstractions suivantes de Plaisted :

1) *L'abstraction constante ("ground abstraction")*

$$A_C(C) = C' \quad C' \text{ est une instance constante de } C$$

2) *L'abstraction propositionnelle*

$$A(P(t_1, \dots, t_n)) = P$$

3) *Renommage de symboles fonctionnels*

A la différence de Plaisted, nous exigeons que le renommage de symboles de fonction soit une fonction injective.

4) *Permutation d'arguments* (nous l'avons appelée "transformation équivalente")

La permutation d'arguments est aussi permise pour les fonctions.

5) *Changement du nombre d'arguments* (diminution ou augmentation)

aussi bien pour les littéraux que pour les fonctions.

Plaisted n'admet que la diminution du nombre d'arguments.

Evidemment le cas 2 est un cas particulier du cas 5.

REMARQUE : Le résultat de l'application d'une abstraction à une clause peut donner comme résultat un *multiensemble* de littéraux (c'est-à-dire un ensemble où un même élément peut figurer plusieurs fois). Ce multiensemble est appelé dans [108], [109] une "m-clause".

Pour être rigoureux, il aurait fallu définir une *cm-déduction* ou une *cR-déduction* pour les m-clauses, mais le sens intuitif est suffisamment clair et nous nous contentons d'un exemple.

EXEMPLE :

Soit l'ensemble de clauses

1) $\bar{P}(a) \quad \bar{P}(b) \quad Q(c)$

2) $P(a)$

3) $P(b) \quad R(d)$

l'abstraction propositionnelle donne l'ensemble de m-clauses :

1') $\bar{P} \quad \bar{P} \quad Q$

2') P

3') $P \quad R$

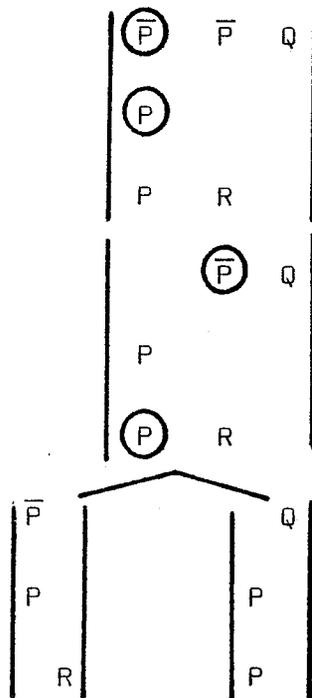
On peut faire, par exemple, une CR-déduction ou une CM-déduction pour cet ensemble.

Une CR-déduction serait :

4') $\bar{P} \quad Q$ (de 1' et 2')

5') $Q \quad R$ (de 3' et 4')

Une CM-déduction :



Le pas suivant a été pour nous d'incorporer à la MRM les *connexions et l'abstraction* en les combinant dans ce que nous avons appelé les *gp-abstractions* ou les *ap-graphes*. Cette notion nous sert à diviser les preuves en un "plan" et une "validation du plan".

4.1.3 LES GP-ABSTRACTIONS OU LES AP-GRAPHES

Δ Soit M une matrice, et soit A une abstraction propositionnelle pour les littéraux de M, tel que

$$A(L(t_1, \dots, t_n)) = L'_i \text{ où } i \text{ identifie la clause à laquelle appartient } L$$

une *gp-abstraction de M* ou un *ap-graphe de M* (noté indistinctement $\text{gpa}(M)$ ou $\text{apg}(M)$) est un graphe non orienté, étiqueté :

$$\text{apg}(M) = \langle N, B, E \rangle$$

(N : ensemble de noeuds ; B : ensemble des arcs ; E : fonction d'étiquetage)

N = (multi)ensemble de littéraux : image des littéraux des clauses de M par A

B : ensemble des arcs ou des liens

$$A = \{ \langle n_1, n_2 \rangle \mid n_1 \in N, n_2 \in N ; \text{ si } n_1 = L \text{ alors } n_2 = L^c, n_1 \in C_1, n_2 \in C_2, C_1 \neq C_2 \}$$

(Noter que nous ne considérons pas les pseudo-liens parce que nous en tiendrons compte d'une façon équivalente : nous élargirons la matrice en ajoutant des variantes des clauses).

L'étiquetage des arcs est une application

$$E : B \longrightarrow \mathcal{P}_f(\epsilon)$$

ϵ : ensemble de toutes les équations dans les termes

$\mathcal{P}_f(\epsilon)$ = ensemble de parties finies de ϵ : ensemble de toutes les conditions de substitution

E est définie de la façon suivante

$$a \in B \quad a = \langle A(L), A(L') \rangle \quad L = P(t_1, \dots, t_n) \quad L' = \bar{P}(t'_1, \dots, t'_n)$$

$$e_i : \{t_i = t'_i\} \quad (1 \leq i \leq n)$$

$$E(a) = \bigcup_{i=1}^n e_i \cdot \forall$$

REMARQUE 1 : En pratique, et pour des raisons d'implémentation, la fonction E sera "dédoublée" en deux autres fonctions E_1 et E_2

$$\text{apg}(M) = \langle N, B, E_1, E_2 \rangle$$

$$E_1 : B \longrightarrow N \quad E_2 : N \longrightarrow \mathcal{P}_f(\epsilon)$$

E_1 et E_2 sont telles que :

$$E = E_1 \circ E_2$$

En pratique donc, les arcs seront étiquetés par des entiers correspondant à des *noms* de conditions de substitution et non par des conditions de substitution. Le lien entre les noms des conditions et les conditions étant fait par ailleurs.

REMARQUE 2 : Il est clair que dans la construction du graphe on aurait pu considérer d'autres abstractions que les abstractions propositionnelles. Par exemple celles qui suppriment m parmi les n arguments d'un littéral ($m \leq n$). Il est clair aussi que l'on peut considérer des abstractions d'une abstraction, l'abstraction "globale" correspondant à la composition d'abstractions.

4.2 PREUVE = PLAN + VALIDATION

Etant donné une matrice M et une gp-abstraction de M , on peut introduire les notions de "plan" et de "validation d'un plan".

Δ Soit M une matrice et $gpa(M)$ une gp-abstraction de M . Un *plan de déduction* d'un ensemble de matrices T à partir de M est un ensemble fini :

$$\{n_1^{j_1}, n_2^{j_2}, \dots, n_p^{j_p}\}$$

où les n_i ($1 \leq i \leq p$) sont des étiquettes du graphe dans la gp-abstraction (correspondant à des noms de conditions de substitution), et où les j_i ($1 \leq i \leq p$) permettent d'identifier la matrice dans laquelle on a appliqué la RM utilisant les littéraux qui sont les noeuds de n_i . ∇

Δ Un *plan de réfutation* est un plan de déduction de $\{\{\}\}$. ∇

Il est clair que, sauf dans le cas propositionnel, un plan de réfutation est une condition nécessaire, mais non suffisante pour trouver une réfutation.

Etant donné une matrice M , une gp-abstraction de M et un plan de déduction (réfutation) $P: \{n_1^{j_1}, \dots, n_p^{j_p}\}$, une *validation du plan de déduction (réfutation)* est une solution du système d'équations :

$$S = \bigcup_{i=1}^p n_i^{j_i} E(n_i^{j_i}) \quad E : \text{fonction d'étiquetage dans } apg(M)$$

$n_i^{j_i}$: substitution de renommage des variables de $E(n_i^{j_i})$ ou identité partout.
 n_i correspond au renommage des variables incluses dans la règle de RM (§3.2.3.2).
 Si P correspond à un plan de réfutation, alors la solution de S est une *substitution de réfutation* (§2.2.5) de M .

REMARQUE : Nous n'avons pas parlé de connexions, d'abstraction, de plan et de validation pour les c-matrices, parce qu'il est évident que c'est un cas particulier de la gp-abstraction où la validation est toujours possible (l'ensemble d'équations est vide). Si, dans une déduction, on décide de représenter les matrices par ses ap-graphes correspondants (ce qui est l'idée principale du partage de structure pour les matrices), l'algorithme qui détecte l'insatisfaisabilité d'une matrice est le suivant. Avec des modifications concernant entre autres le retour arrière (voir §4.3) il peut être considéré comme spécifiant (d'une façon très générale) l'action principale du démonstrateur. Il correspond à implantation séquentielle ; comme nous le verrons au §4.6 la division en plan + validation se prête à une implantation en parallèle.

procédure MRM (M : matrice) ;

* cherche à trouver une réfutation comme plan + validation *

début

arrêt ← faux;

sat ← faux;

insat ← faux;

choisir une façon d'énumérer les chemins dans
l'arbre de recherche ;

* équivaut à énumérer les plans éventuels *

tantque non arrêt et non insat

faire

plan ← plan-suivant ; * la fonction plan-suivant élargit la matrice
abstraite *

si plan ≠ \emptyset alors

si validation (plan) alors

insat ← vrai

sinon

si critère arrêt vérifié alors

arrêt ← vrai

fsi

sinon

arrêt ← vrai

sat ← vrai * correspond à un cas décidable *

fsi

faire;

si insat alors

imprimer ('M insatisfaisable', 'subst de réfutation;', solution
système d'équations fournie par validation)

sinon

si sat alors

imprimer ('M satisfaisable')

sinon

imprimer ('?')

fsi

fsi

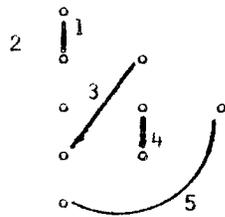
fin * MRM *

Il est intéressant de comparer sur quelques points la MRM utilisant la gp-abstraction, avec la méthode des graphes de connexion de Kowalski.

L'exemple déjà traité par la méthode des graphes de connexion sera traité par cette méthode de la façon suivante :

EXEMPLE :

La gp-abstraction de M est

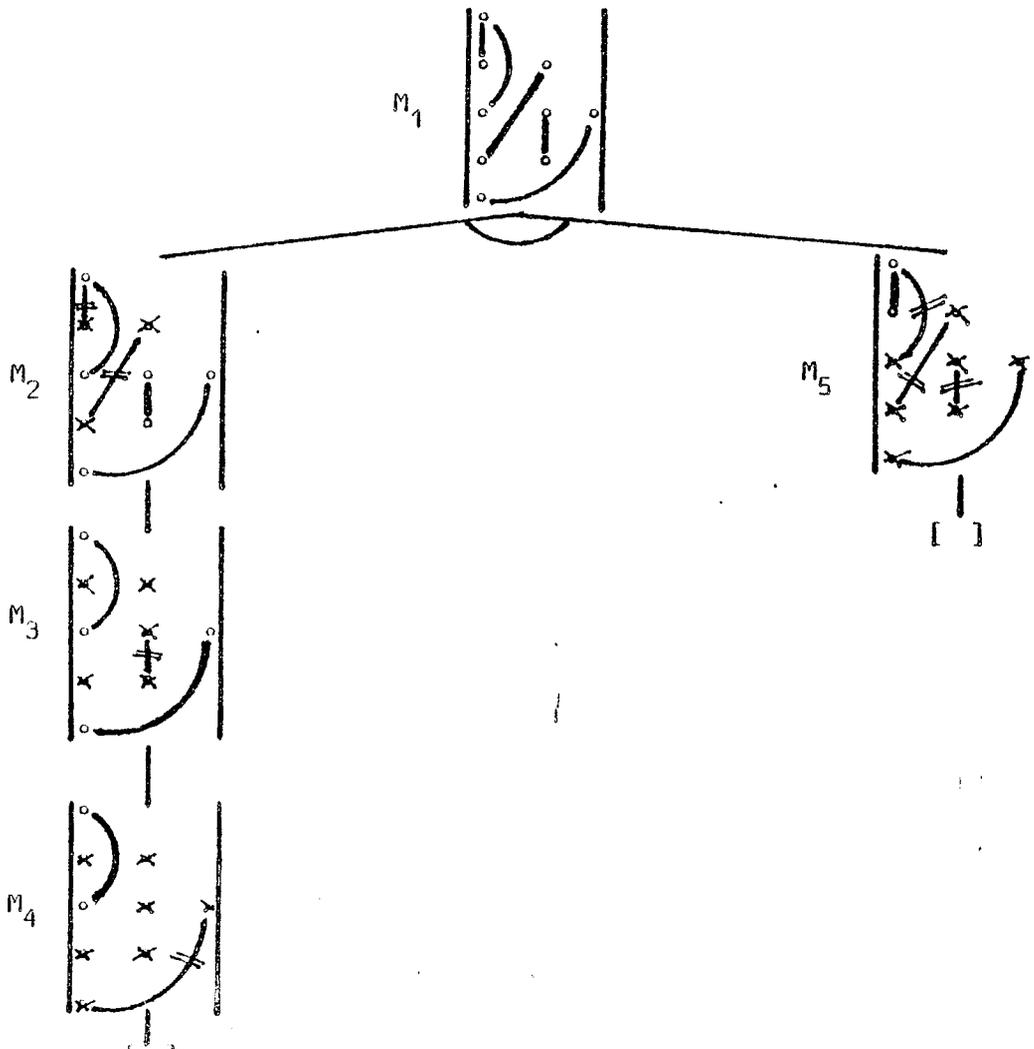


- $E(1) = \{u=x\}$
- $E(2) = \{u=z\}$
- $E(3) = \{x=b\}$
- $E(4) = \{y=b\}$
- $E(5) = \{z=a, y=b\}$

Dans la réfutation suivante, nous notons

— : lien choisi

: lien éliminé (par élimination d'un noeud par application RM ou par application principe de pureté)



Un plan de réfutation est donc

$$\{3(M_1), 4(M_2), 5(M_3), 2(M_4), 1(M_5)\}$$

Validation (renommage de u dans M_5)

L'ensemble d'équations

$$\{x=b, y=b, u_1=x, z=a, y=b, u=z\}$$

a une solution.

Quelques différences (générales) peuvent être remarquées sur cet exemple :

- 1) Les liens du graphe de la gp-abstraction ne sont pas étiquetés par un p.g.u., mais par une condition de substitution.
- 2) Aucun p.g.u. n'est calculé avant la phase de validation (comparer avec [28]).
- 3) Dans la méthode de graphes de connexion on *élargit* successivement un graphe ; dans la MRM on *remplace* un graphe par un (ou deux) autre(s) de ses sous-graphes.

4.3 ENSEMBLE DE CONDITIONS DE SUBSTITUTION MINIMALEMENT INCOMPATIBLES: UTILISATION POUR LE RETOUR ARRIERE

L'essai de validation d'un plan consiste à tenter de résoudre un système d'équations dans les termes.

Une stratégie possible, qui est d'ailleurs celle qui a été choisie, pour chercher une solution est de considérer la partition fournie par les conditions de substitution associées au plan, de fixer un ordre, et d'ajouter (dans l'ordre choisi) une à une ces conditions en résolvant des ensembles croissants d'équations.

D'après la définition de solution d'un système d'équations, si le système a une solution, alors, pour *n'importe quelle partition* de l'ensemble, tous les ensembles de la partition ont une solution.

Il est aussi clair que *l'ordre* dans lequel on décide de résoudre les équations, ne peut pas influencer la propriété de l'ensemble initial d'avoir ou de ne pas avoir une solution. Par contre dans le cas où l'ensemble à valider *n'a pas de solution*, l'ordre dans lequel les ensembles de la partition sont considérés, a une importance pour détecter le plus tôt possible l'inexistence d'une solution.

On peut considérer comme exemple l'ensemble d'équations $S = S' \cup \{x = a\}$

$\cup \{y = b\} \cup \{x = y\}$ où S' est un ensemble d'équations ayant une solution.

Si l'on commence par tester l'existence d'une solution pour $\{x = a\} \cup \{y = b\}$

$\cup \{x = y\}$ on peut arrêter immédiatement avec une réponse négative à la question de l'existence d'une solution. Le cas n'est pas le même si l'on décide de commencer la recherche de la solution par S' (S' peut être un ensemble de très grande cardinalité).

Nous ne nous intéressons pas au problème de savoir dans quel ordre résoudre un système d'équations. Dans le cas où un système d'équations n'a pas de solution, nous voulons détecter des sous-ensembles particuliers.

Δ Un ensemble E d'équations dans les termes ($\text{card}(E) \geq 2$) est dit *minimalement incompatible* ssi E n'a pas de solution et pour tout E' tel que $E' \subset E$ E' a une solution. \forall

La restriction $\text{card}(E) \geq 2$ est dans la définition pour écarter le cas trivial d'une équation sans solution.

En général, un ensemble d'équation qui n'a pas de solution contiendra plus d'un sous-ensemble minimalement incompatible.

Nous nous intéresserons à des ensembles de conditions de substitution, la définition d'incompatibilité minimale correspondante est obtenue en remplaçant "ensemble d'équations" par "ensemble de conditions de substitution".

Dans un essai de réfutation, une fois un ensemble minimalement incompatible détecté, on ne choisira pas de plan le contenant. L'intérêt de pouvoir identifier de tels ensembles est donc évident.

Une façon que nous avons trouvée pour ce faire est de mémoriser certaines informations pendant la phase de validation.

Comme nous l'avons déjà annoncé au Chapitre 2, nous avons choisi d'utiliser l'algorithme d'unification de Huet (§ 2.1.2.2). Nous exposons maintenant les raisons de ce choix.

4.3.1 CHOIX DE L'ALGORITHME D'UNIFICATION

Une première version de cet algorithme a été donnée par Robinson en 1965 [122], [123]. D'autres algorithmes plus performants ont été publiés depuis: ceux de Baxter [9], Huet [68], Martelli et Montanari [96], Paterson et Wegman [105] et Robinson [120].

Nous n'avons pas choisi en fonction exclusivement de leur complexité relative (voir par exemple [68], [96]), mais en fonction de caractéristiques autres que la simple efficacité.

Les caractéristiques qui nous intéressent dans l'algorithme de Huet sont:

1) Il *sépare* le processus d'unification en deux phases (l'algorithme de Baxter sépare aussi les deux phases).

Cette caractéristique pourrait être utilisée dans un langage comme PROLOG [75], [125] pour un test de cycle ("occur check") facultatif.

Cette séparation n'est pas faite dans les algorithmes de Martelli et Montanari, et de Paterson et Wegman.

2) Les paires de termes à unifier peuvent être choisis *dans n'importe quel ordre*. L'algorithme original de Robinson n'avait pas cette caractéristique, mais celui publié dans [120] l'a incorporée.

L'intérêt que cela a dans une implantation est évident (voir aussi [68]).

Dans [96] une comparaison est faite entre les algorithmes de Huet, de Martelli et Montanari, et de Paterson et Wegman, dont nous retiendrons les conclusions suivantes:

- . L'algorithme de Paterson et Wegman est théoriquement le meilleur. Il a une complexité linéaire.
- . L'algorithme de Huet et celui de Martelli et Montanari n'ont pas une complexité linéaire. L'algorithme de Huet a une complexité presque linéaire.
- . Les 3 algorithmes comparés peuvent s'arrêter, soit avec succès, soit avec échec. Dans le deuxième cas, l'arrêt peut être dû à un essai d'unification de deux termes de nom différent ("clash"), ou à un cycle.

En appelant

P_m : probabilité d'arrêt avec succès
 P_t : " " " échec ("clash")
 P_c : " " " arrêt ("cycle")

Trois possibilités sont considérées:

1) $P_m \gg P_c$ $P_m \gg P_t$

L'algorithme de Paterson et Wegman est asymptotiquement le meilleur (mais cette caractéristique n'est pas très utilisée dans les démonstrateurs de théorèmes parce qu'en général on n'unifie pas de termes "très longs").

2) $P_c \gg P_t \gg P_m$

L'algorithme de Paterson et Wegman est le meilleur.

3) $P_t \gg P_c \gg P_m$

L'algorithme de Huet est le meilleur.

Nous allons maintenant voir comment on peut utiliser l'information manipulée par l'algorithme de Huet, dans le cas où il y a eu échec pour déterminer un ensemble de conditions de substitution minimalement incompatible.

4.3.2 DETECTION D'ENSEMBLES D'EQUATIONS MINIMALEMENT INCOMPATIBLES

Notre présentation est orientée vers le problème qui nous intéresse, c'est-à-dire la division d'un essai de réfutation en "plan" et "validation", mais il est clair que le même algorithme peut être appliqué dans la résolution de n'importe quel système d'équations dans les termes.

Soit un plan de réfutation d'une matrice M

$$\{ n_1^{j_1}, n_2^{j_2}, \dots, n_p^{j_p} \}$$

Pour valider ce plan, il faut trouver une solution de l'ensemble d'équations

$$\bigcup_{i=1}^p E(n_i^{j_i}) \quad E: \text{fonction d'étiquetage du graphe de la gpa (M).}$$

Comment détecter les ensembles minimalement incompatibles quand un échec se produit dans la recherche d'une solution de l'ensemble d'équations à l'aide de l'algorithme d'unification ?

Dans l'algorithme d'unification de Huet, deux types d'échecs sont possibles:

- 1) Essai d'unification de deux termes dont le nom de fonction est différent (voir procédure eqclasses § 2.1.2.2).
- 2) Détection d'un cycle dans le graphe de classes d'équivalence (voir fonction noncycle § 2.1.2.2).

Un ordre étant choisi pour la solution de l'ensemble de conditions de substitution, une fonction f est utilisée pour représenter (mémoriser) la fusion des classes.

$$f: V \rightarrow (N \times V)$$

V : ensemble de variables du problème (dans le sens du § 2.1.2.2, c'est à dire soit une variable dans les termes à unifier, soit une variable de travail).

N : ensemble de noms de conditions de substitution du problème.

$f(v_1) = \langle n, v_2 \rangle$ signifie que la classe de la variable v_1 a été fusionnée avec la classe de la variable v_2 quand on a incorporé la condition de substitution n à l'essai de validation.

En particulier $v_1 = v_2 = v$ signifie que la classe de v n'a pas été fusionnée (elle contient seulement v , voir fonction UNIFIABLE § 2.1.2.2) et que v a été introduite quand on a incorporé n à l'essai de validation.

Δ L'*histoire d'une variable* v pour arriver à une classe dont le représentant est w , est la plus courte suite de noms de conditions de substitution n_1, n_2, \dots, n_p , telle que

$$n_i = pr_1 (f(v_i)) \quad 1 \leq i \leq p$$

$$w = pr_2 (f(v_p))$$

$$v_j = v$$

j : le plus grand entier tel que $v \in EV(n_j)$ (notation du Chapitre 2)

$pr_1 (f(x))$ et $pr_2 (f(x))$ sont respectivement la première et la deuxième projection de $f(x)$

$$v_{i+1} = pr_1 (f(v_i)) \quad 1 \leq i \leq p-1 \quad \Delta.V$$

Supposons que l'ensemble de conditions de substitution à tester pour compatibilité soit un ensemble de p conditions de substitution que nous ordonnons pour résoudre et que l'on renomme $1, 2, \dots, p$. On commence par 1 et on ajoute successivement $2, 3, \dots$ jusqu'à ce que il y ait échec ou que l'on puisse prouver la compatibilité de $1, 2, \dots, p$. Le but est d'extraire de l'information des échecs de l'algorithme d'unification (utilisé pour résoudre le système d'équations), ceci est fait à l'aide des fonctions `mininonunif_clash` et `mininonunif_cycle` ci-dessous (on utilise le § 2.1.2)

Cas 1: Echec de l'algorithme d'unification par échec de EQCLASSES.

Supposons que EQCLASSES a pu construire les classes pour $1, 2, \dots, n$ et l'échec se produit en essayant d'incorporer $n+1$ et ceci en essayant d'unifier les variables (au sens large) v_1 et v_2 .

```

fonction MININONUNIF_CLASH (v1, v2): ensemble de noms de conditions de substitution;
  * v1 ∈ [ w1 ] , v2 ∈ [ w2 ] *
début
  pour i: = 1, 2
    faire
      E ← { 1, 2, ..., n+1 }
      pour j: = 1 à n * pour trouver la suite la plus courte *
        faire
          si EV(j) ∩ { vi } ≠ ∅ alors
            E ← E - { 1, 2, ..., j - 1 }
          fini
        faire
          t ← pr2 (f(vi));
          hist ← pr1 (f(vi))
          tantque t ≠ wi
            faire * construction de l'histoire de vi *
              t ← pr2 (f (pr2 (f(t))));
              hist ← hist ∪ { pr1 (f (pr2 (f(t)))) }
            faire
          fini
        faire
          mininonunif_clash ← hist ∪ { n+1 }
          * le "clash" s'est produit avec l'ajout de n+1 *
fin * MININONUNIF_CLASH * ;

```

Il est intéressant de remarquer que cet algorithme est une approche "duale" de celle proposée par Cox [36], qui utilise l'algorithme de Baxter (similaire à celui de Huet). Il est plus simple que celui-ci. Il présente des similitudes avec celui de Pietrzikowski et Matwin [106].

Cas 2 : Echec de l'algorithme d'unification parce que NOCYCLE retourne "faux"

fonction MININONUNIF_CYCLE: ensemble de noms de conditions de substitution;

debut

mininonunif_cycle $\leftarrow \emptyset$;

pour chaque flèche $C_1 \rightarrow C_2$ dans le cycle

faire

* le représentant de la classe de C_1 est w_1 ($\rho(w_1) = f(\dots v \dots)$) *

* le représentant de la classe de C_2 est w_2 *

mininonunif_cycle \leftarrow mininonunif_cycle \cup hist (v, w);

* la fonction hist peut être obtenue à partir de mininonunif_clash *

faire

fin * MININONUNIF_CYCLE * ;

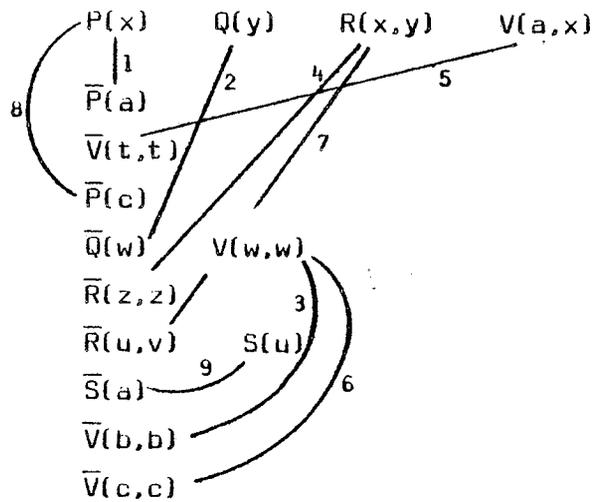
LEMME: Les algorithmes *mininonunif_clash* et *mininonunif_cycle* détectent un ensemble minimalement incompatible contenu dans un ensemble incompatible de conditions de substitution.

PREUVE: D'abord il est clair que les algorithmes proposés en sont effectivement (ils s'arrêtent toujours).

L'idée de la démonstration est la suivante: les ensembles détectés sont incompatibles (puisque'ils contiennent un "clash" ou un cycle). Ils sont minimalement incompatibles par construction. ■

EXEMPLE [106]:

Nous traitons avec notre méthode l'exemple suivant (comparer avec la façon dont il est traité en [106]).



Un plan possible (dans le sens de [106] (et aussi dans le notre) est {1,2,3,4,5}. La méthode proposée fonctionne de la façon suivante:

$$\begin{aligned}
 C^* &= \{ \{ a, x \} \} && \{1\} \\
 C^* &= \{ \{ a, x \}, \{ w, y \} \} && \{1,2\} \\
 C^* &= \{ \{ a, x \}, \{ b, w, y \} \} && \{1,2,3\} \\
 C^* &= \text{"clash" en voulant résoudre} && \{1,2,3\} \cup \{x = z, y = z\}
 \end{aligned}$$

L'histoire qui a conduit x dans la classe de a est {1}

L'histoire qui a conduit y dans la classe de b est {2,3}

Donc {1,2,3,4} est minimalement incompatible et {1,2,3} est compatible, ce qui peut être utilisé par le démonstrateur dans la recherche d'un autre plan (voir annexe).

4.4 UNE TECHNIQUE DE PARTAGE DE STRUCTURE POUR LES MATRICES ET POUR LES SUBSTITUTIONS

Un travail maintenant classique sur le partage de structure dans les démonstrateurs basés sur la résolution a été publié en 1972 par Boyer et Moore [22]. Il y a dans ce travail deux idées fondamentales :

- 1) La première est une idée qui avait été déjà utilisée en LISP [98] et qui *évite de créer physiquement la valeur d'un terme ou d'un littéral ou d'une clause* (appelés tous des *termes* dans [22]) dans le contexte d'une substitution.

EXEMPLE :

$L : P(x, f(y, g(z, x)))$

dans le contexte de la substitution (environnement)

$\sigma = \{ \langle x, a \rangle, \langle y, f(v, w) \rangle, \langle z, b \rangle \}$

représente $P(a, f(v, w), g(b, a))$

(L'application de σ à L n'étant pas effectivement réalisée)

- 2) Les résolvantes peuvent être obtenues (représentées), *sans appliquer la résolution et sans construire effectivement la clause résolvante à partir des clauses parents.*

Pour ce faire, Boyer et Moore introduisent conventionnellement un ordre parmi les littéraux des clauses, et il montrent (ce qui est intuitivement facile à accepter) que la résolvante de deux clauses C et D peut être représentée par un t -uplet.

$T : \langle C, i, D, j, NL, MI, \sigma \rangle$

où C et D repèrent les clauses parents.

i et j : les littéraux sur lesquels on a résolu.

NL : le nombre de littéraux dans la clause résolvante.

MI : indice utilisé pour le renommage des variables.

σ : modification de l'environnement due au processus d'unification dans l'obtention de la résolvante (l'environnement est global au processus et représente l'ensemble des liens des variables).

Un algorithme d'unification et un algorithme de résolution adaptés à la représentation proposée sont donnés. Il faut signaler que Robinson avait déjà décrit

[123] comment unifier des termes dans le contexte d'une substitution, *sans* appliquer la substitution.

Nous avons utilisé dans [130] et dans [24] des idées semblables, en les généralisant à des matrices, c'est-à-dire des ensembles de clauses.

Des algorithmes adaptés à cette représentation ont été donnés dans [130], [131].

Nous allons présenter la technique de partage de structure proposée, tout en la comparant à celle de Boyer et Moore.

Les deux points principaux sont également :

- 1) La représentation des termes *sans* application des substitutions.
- 2) La représentation *économique* des matrices.

1) LA REPRESENTATION DES TERMES SANS APPLICATION DES SUBSTITUTIONS

Elle est naturellement faite par l'algorithme d'unification choisi. Les classes d'équivalence et l'environnement donnent une représentation économique de substitutions.

La fonction Eqclasses (§2.1.2.2) manipule des classes d'équivalence ; elle détecte si un élément appartient à une classe et elle fait l'union des classes d'équivalence. Cette spécification correspond à ce qui est connu dans la littérature comme "union find algorithm" [2], [53], [83].

Par rapport à la représentation de Boyer et Moore, l'utilisation de l'algorithme de Huet a un avantage additionnel : celui de ne pas représenter une substitution comme un ensemble de couples, mais comme des classes d'équivalence, ce qui équivaut à ne représenter qu'une fois un même terme composant de plus d'un composant de substitution.

Il faut aussi remarquer que l'on *ne calcule pas le p.g.u.*, mais simplement qu'on s'assure de son existence.

Evidemment, il n'est pas nécessaire d'adapter l'algorithme d'unification à cette représentation économique, comme ont été obligés de le faire Boyer et Moore.

Dans [22] l'environnement de substitution est modifié avec chaque calcul de résolvante. Les classes d'équivalence le sont aussi quand on résout une équation dans les termes correspondant à une condition de substitution.

EXEMPLE [22] :

Il s'agit de détecter si un ensemble de deux littéraux est unifiable.

L'ensemble de deux littéraux est représenté comme un ensemble

$$E = \{P(x_2, y_2), P(g(x_7), z_7)\}$$

et un environnement

$$\theta = \langle \underline{x_2, x_3}, \langle y_2, f(x_4, y_4), \underline{y_4, x_3}, \langle z_7, f(x_8, y_8) \rangle, \langle x_8, x_7 \rangle, \langle y_8, g(y_5) \rangle \rangle$$

(Les doublets soulignés seront fusionnés dans une seule classe d'équivalence par l'algorithme d'unification de Huet).

Ceci correspond, comme il est intuitivement clair, à essayer d'unifier l'ensemble de deux littéraux suivant :

$$\{P(x_3, f(x_4, x_3)), P(g(x_7), f(x_7, g(y_5)))\}$$

L'équivalent de θ est l'ensemble de classes d'équivalence suivant :

$$C^* = \{\{f(x_4, y_4), y_2\}, \{x_3, x_2, y_4\}, \{f(x_8, y_8), z_7\}, \{x_7, x_8\}, \{g(y_5), y_8\}\}$$

Après unification l'environnement devient

$$\theta \cup \{\langle \underline{x_4, y_5}, \underline{x_7, x_4}, \langle x_3, g(x_7) \rangle\}$$

Le C^* correspondant est

$$C^* = \{\{f(x_4, y_4), y_2\}, \{y_5, x_4, x_7, x_8\}, \{g(x_7), x_2, x_3, y_4\}, \{f(x_8, y_8), z_7\}\}$$

Nous pensons que cet exemple illustre clairement l'avantage représenté par l'utilisation d'un algorithme comme celui de Huet.

2) REPRESENTATION ECONOMIQUE DES MATRICES

Une première version a été donnée dans [130] [131]. Nous y avons proposé une adaptation de la méthode de Boyer et Moore à la MRM. Il y avait une *forme étendue* et une *forme réduite* pour les matrices.

La matrice sous forme étendue était représentée une seule fois en mémoire et elle correspondait à l'ensemble de clauses initial.

Une matrice sous forme réduite était une liste de paires correspondant aux RM appliquées pour arriver à la matrice en question :

$$[\langle (L_n, P_n), CS_n \rangle, \langle (L_{n-1}, P_{n-1}), CS_{n-1} \rangle, \dots, \langle (L_1, P_1), CS_1 \rangle]$$

CS_i ($1 \leq i \leq n$) : conditions de substitution

On parlait aussi de "compatibilité verticale" et "compatibilité horizontale" des substitutions appliquées (correspondant respectivement à la compatibilité avec les substitutions appliquées aux ancêtres et aux frères dans l'arbre de recherche).

Nous avons modifié (en la simplifiant) cette représentation depuis [24]. En ce qui concerne le traitement des classes d'équivalence nous avons utilisé le mécanisme décrit au §4.3 pour pouvoir utiliser les échecs de l'unification pour un retour arrière non aveugle, ce qui a été possible par l'introduction du concept d'abstraction. Les algorithmes associés de recherche d'un plan et d'une réfutation sont considérablement plus simples.

Dans la représentation économique des matrices, la différence avec [22] est plus grande à cause de l'introduction des connexions.

La matrice initiale M est représentée une seule fois en mémoire. A partir de cette matrice, on obtient le gp-graphe de M, et l'on représente M comme un ensemble de liens :

$$M = \{1, 2, \dots, n\}$$

et une fonction faisant correspondre à chaque composant de M la condition de substitution qui lui est associée.

Les matrices obtenues par RM à partir de M seront des sous-ensembles de M ; c'est pourquoi on peut utiliser une structure de données statique comme les ensembles.

Nous avons choisi Pascal [70] comme langage d'implantation, pour des raisons similaires à celles évoquées dans [92].

Dans ce langage, la structure d'ensemble est l'une des structures fondamentales.

Un ensemble est représenté d'une façon convenable par sa *fonction caractéristique*, sa représentation interne étant une *chaîne de bits* (on peut donc simuler cette structure, avec les mêmes avantages, dans des langages comme PL1 ou ADA, par exemple). Dans cette représentation les opérations d'union, intersection et différence d'ensembles peuvent être implantées comme des opérations logiques élémentaires qui opèrent simultanément pour tous les bits d'un mot (voir p. ex. [159]).

La fonction d'étiquetage est représentée par un tableau ECS dont nous donnons la déclaration en Pascal.

```
type condsbst = record
    licolit,licolitcomp : integer ; * ligne et colonne des
                                noeuds d'un lien *
    descpairetermes : ptstrcond ; * pointeur vers une structure
                                décrivant les paires de termes composants
                                de la condition de substitution *
    end;
var ECS : array [1..N] of condsbst ; * ECS : ensemble de conditions de
                                substitution *
```

Nous donnons aussi la description Pascal de noeuds de l'arbre de recherche. Il y a deux types de noeuds : les noeuds "et" et les noeuds "ou".

```
type
    liens : 1..N;
    ptnoeudet : ↑noeudet;
    ptnoeudou : ↑noeudou;
    ensdeliens: set of liens;
    noeudet = record * matrice *
        liencourant : ptnoeudou ; * lien choisi pour l'application RM *
        liens : ensdeliens ; * ensemble de liens de la matrice *
        listeliens : ptnoeudou ;
    end;
    noeudou = record * liens: sur lesquels on applique la RM *
        nomlien : liens ; * un lien *
        matgauche,matdroite : ptnoeudet ;
        liensuisant : ptnoeudou ; * ordre défini par stratégie ou
                                heuristique *
        actif : boolean ;
    end;
```

Parmi les algorithmes nécessaires à l'implantation du démonstrateur, celui qui correspond le plus directement à la structure de données proposée est celui de l'application de la règle de RM à une matrice ; cette matrice et la (les) matrice(s) résultat étant représentées comme des ensembles de liens.

procédure MATRED (M: ensdereds; var M₁, M₂:ensdeliens);

* elle n'est pas appliquée si M = [], lien choisi: l = < l₁, l₂ > *

debut

EL ← ensemble de tous les littéraux de toutes les lignes de M;

* on peut obtenir cet ensemble utilisant M et ECS *

M₁ ← M;

M₂ ← M;

EL1 ← EL - { l₁ } - (ligne l₂ - { l₂ });

EL2 ← EL - { l₂ } - (ligne l₁ - { l₁ });

pour i: = 1 à N * N = nb maximum de liens dans une matrice *

faire

si i ∈ M alors

obtenir de ECS les deux nœuds du lien: i = < n₁, n₂ > ;

si n₁ ∈ EL1 ou n₂ ∈ EL1 alors

M₁ ← M₁ - { i }

fsi

si n₁ ∈ EL2 ou n₂ ∈ EL2 alors

M₂ ← M₂ - { i }

fsi

fsi

faire

Appliquer le principe de pureté à M₁ et à M₂;

si M₁ = φ ou M₂ = φ alors

marquer le lien l comme non actif

fsi

fin * MATRED * ;

4.5 STRUCTURE DES DONNEES UTILISEE POUR LE RETOUR ARRIERE

Après un échec de validation, un retour arrière dans l'arbre de recherche est nécessaire. Nous sommes intéressés à des retours arrière "non aveugles (non exhaustifs)". La question se pose de savoir quelle est la façon la plus convenable (la plus rapide) pour avoir accès à n'importe quel nœud de l'arbre, sans augmenter la place mémoire occupée pour représenter l'arbre. Ce problème est similaire à celui traité dans [45].

La solution la plus naturelle est d'utiliser un tableau, comme nous l'indiquons graphiquement ci-après.

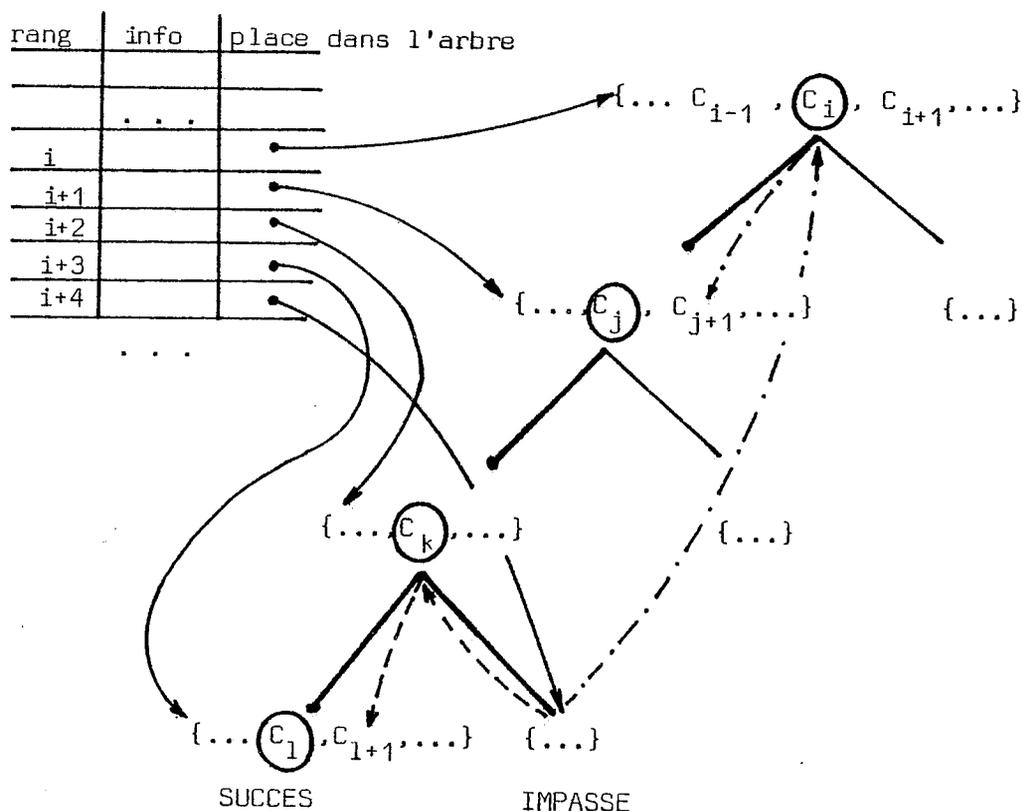
Par rapport aux deux types classiques de retour arrière (voir par exemple [144]) notre approche correspond à un "retour arrière d'état" plutôt qu'un "retour arrière séquentiel".

Cette représentation nous permet de repérer simplement la partie de l'arbre qui reste inchangée dans le retour arrière, et d'effectuer un "ramassage de miettes" ad hoc des branches de l'arbre inutiles.

La place occupée par les pointeurs dans le tableau est la même qui occuperaient les éventuels "pointeurs vers l'ancêtre".

Les indices du tableau correspondent à l'ordre de parcours (construction) de l'arbre de recherche.

Un même lien ne peut pas apparaître plus d'une fois dans un plan, la taille maximale du tableau est donc égale au nombre de liens de la matrice initiale.



- : chemin parcouru à la recherche d'une réfutation
- : retour arrière aveugle
- : possible retour arrière guidé.

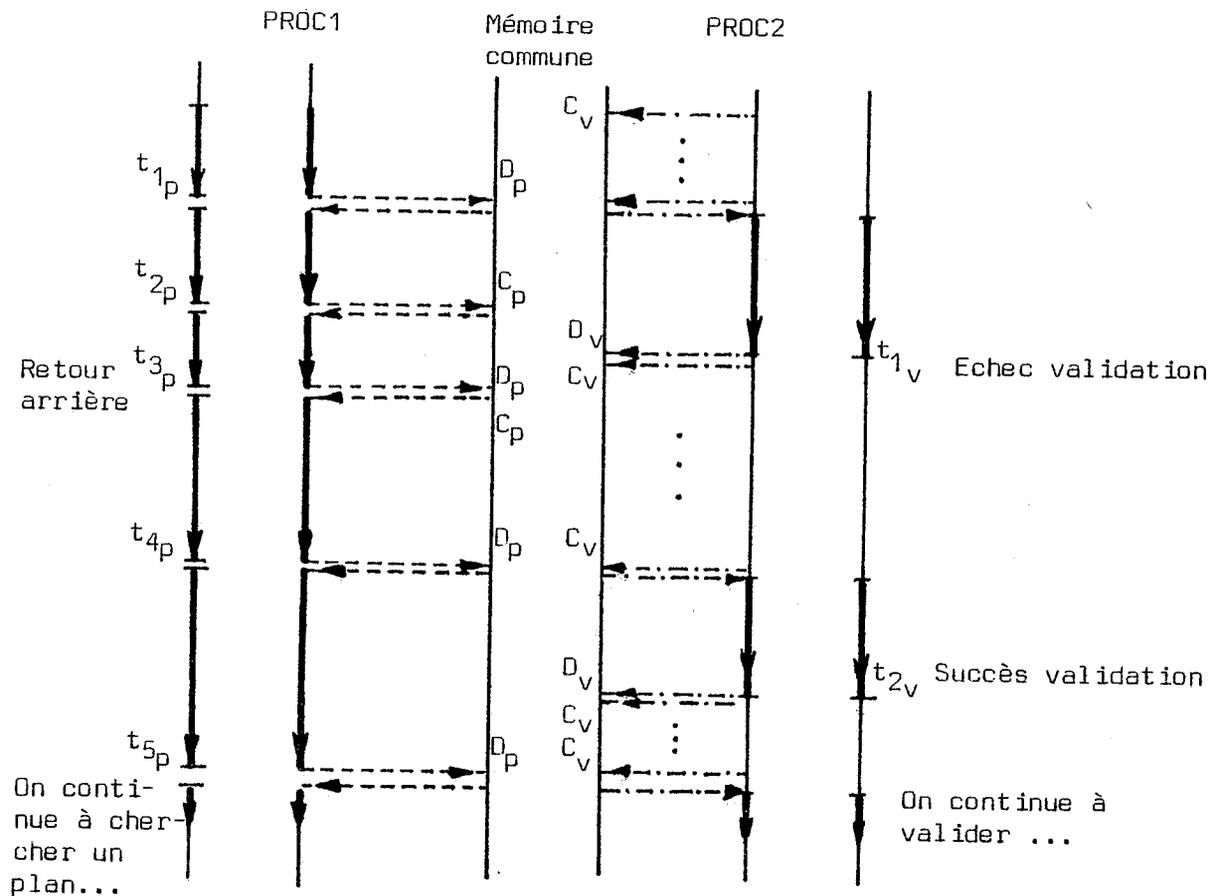
4.6 POSSIBILITE D'INTRODUIRE LE PARALLELISME

Une fois que la division en plan et validation est faite, la question suivante se pose naturellement: "Pourquoi attendre d'avoir fini la recherche d'un plan pour commencer le processus de validation?". Autrement dit: si l'on dispose de, disons, deux processeurs (PROC1 et PROC2) et d'une façon de communiquer des processus entre eux, on peut lancer un processeur à la recherche d'un plan et l'autre à la validation de sous-plans communiqués par le premier.

PROC1 communiquera à PROC2 le sous-plan qu'il est en train d'obtenir. PROC2 communiquera à PROC1 l'échec éventuel de la validation d'un sous-plan ou bien le sous-plan qu'il a pu valider avec éventuellement, des informations pour guider le retour arrière. Nous n'avons fait aucune hypothèse sur la durée relative des pas élémentaires dans l'obtention d'un plan et d'une validation. Nous supposons que la communication peut se faire de façon asynchrone. On peut imaginer par exemple une communication par mémoire commune. Evidemment, PROC1 se servira du mécanisme de retour arrière décrit auparavant.

Comme l'ordre de validation est indifférent, on peut se servir d'informations extérieures pour changer l'ordre d'évaluation par rapport à l'ordre des choix communiqués par PROC1.

Avec un graphisme ad hoc, nous exprimons l'évolution dans le temps des processus se déroulant sur PROC1 et PROC2 (le temps croît vers le bas)



D: dépôt d'information

C: consultation d'information

t_{ip} ($i \geq 1$): durée d'un choix élémentaire

t_{iv} ($i \geq 1$): durée d'une validation élémentaire

Le dessin représente un cas de figure, mais nous n'avons fait aucune hypothèse sur les valeurs relatives de t_{ip} et t_{iv} ainsi que sur la communication de PROC1 à PROC2 (un à un ou groupés).

Trois cas peuvent se présenter (soit toujours, soit du point de vue statistique)

- 1) $t_{ip} < t_{iv}$
- 2) $t_{ip} = t_{iv}$
- 3) $t_{ip} > t_{iv}$

qui contiennent les trois "cas pratiques" suivants:

- 1') $t_{ip} \ll t_{iv}$
- 2') $t_{ip} \approx t_{iv}$
- 3') $t_{ip} \gg t_{iv}$

Dans le cas 1 et 2 (1' et 2'), la mise en parallèle du processus est intéressante.

sante, tandis que dans le cas 3 (3') elle ne l'est pas.

Dans le cas 1', il se peut que le processus de recherche d'un plan ait le temps de trouver plusieurs plans possibles avant que le processus de validation ait pu valider ou infirmer le premier des plans. Il faudra donc modifier convenablement la structure de retour arrière par accès direct exposée ci-dessus.

Comme conclusion de ce travail nous énumérons ce que nous considérons être les points importants de cette thèse et des suites possibles de ce travail.

Les points importants peuvent être divisés en deux:

Une première partie est un travail de synthèse, moins ambitieux que celui des 3 ou 4 excellents ouvrages (en anglais) existant dans le domaine, mais plus orienté vers une utilisation par un informaticien.

Nous sommes fermement convaincus que la place accordée à cette partie est pleinement justifiée (nous pensons qu'il est non seulement intéressant mais aussi *utile* de "s'attarder" un peu sur des résultats qui ont aidé à la fondation d'un domaine d'études).

La deuxième partie développe les idées que nous avons introduites et donne quelque détails sur l'implantation réalisée:

. Les ap-graphes: représentation d'un ensemble de clauses comme un graphe étiqueté par des noms d'ensembles d'équations sur les termes. Avec une représentation naturelle des ap-graphes, on peut représenter des ensembles de clauses par des *ensembles de naturels*, chaque naturel représentant une possibilité d'application de la règle d'inférence (l'information restante étant représentée une seule fois statiquement).

. Partage de structure pour les substitutions: utilisation des caractéristiques de l'algorithme d'unification choisi (celui de Huet).

. Retour arrière non aveugle: modification de l'information mémorisée par l'algorithme d'unification dans ces deux phases pour permettre ce retour arrière.
 . Introduction de la règle de SH-RM, utilisant la caractéristique de la méthode, qui sépare les littéraux des clauses.

. Possibilité d'utiliser la méthode comme une "réduction matricielle non binaire".

Ces points ont permis de répondre aux objections faites contre la méthode, ainsi que de montrer l'utilité des méthodes que l'on pourrait qualifier de "globales" (par rapport à la résolution qui pourrait être qualifiée de "locale").

L'idée d'abstraction (qui a été développée d'une façon similaire par Plaisted) a été parfois critiquée en ceci que l'on peut éventuellement produire beaucoup de plans sans intérêt puisque destinés à ne pas pouvoir être validés. La conception du programme pour une mise en parallèle de 2 processus fondamentaux (plus éventuellement d'autres) répond aussi à cette objection puisque l'on ne "perd rien" si pendant que l'on attend le résultat d'une unification, on continue à chercher la suite d'un plan. Cette observation est aussi valable si l'on étiquette le graphe avec les p.g.u correspondants. Il est aussi parfaitement possible d'utiliser la subsomption sur l'abstraction des clauses. Ceci équivaudrait à supposer que la subsomption est possible, la validation ultérieure étant faite

de la même façon que pour les plans sans tenir compte de la subsomption.

D'ailleurs les ap-graphes constituent à notre avis, même en admettant les critiques valables pour une implantation séquentielle, une "bonne abstraction": elle abstrait la *forme* du problème. Elle est parfaitement adéquate à une mise en parallèle du démonstrateur, qui constitue une des suites possibles de ce travail (la programmation en Pascal permet une traduction "facile" en ADA et une utilisation des possibilités offertes par ce langage pour le traitement du parallélisme). D'autres suites possibles seraient l'incorporation de l'égalité [131] et de l'induction mathématique.

ANNEXE

EXEMPLE 1 : (Détection d'ensembles minimalement insatisfaisables)

```

RUN DEMO1

VOULEZ VOUS LIRE LES CLAUSES A PARTIR DE LA CONSOLE, D'UN FICHIER
OU DES DEUX? (C/F/D)
F
ENTREZ LE NOM DU FICHIER(SANS SON TYPE):KOW
*****
ENSEMBLE DE CLAUSES A REFUTER:
C 1  +Q(X1)                                -P(X1)
C 2  +Q(Z2)                                -S(Y2)
                                           -T(Z2,Y2)
C 3  +P(B)                                  +S(B)
C 4  +T(A,B)
C 5  -Q(U5)
*****
*****
REDUCTIONS:
1:  31- 12 = +P(B)-P(X1)
2:  21- 51 = +Q(Z2)-Q(U5)
3:  11- 51 = +Q(X1)-Q(U5)
4:  32- 22 = +S(B)-S(Y2)
5:  41- 23 = +T(A,B)-T(Z2,Y2)
*****
SCHEMA:
1  2  3  4  5
ARBRE DE REFUTATION:
*****
MATRICE      FILS GAUCHE      FILS DROIT
M1
M2           M3           M4
M3
M4           M5
M5
*****
REDUCTIONS COMPATIBLES:
1 :  5 ( 41- 23): +T(A,B)-T(Z2,Y2) SUR MATRICE M1
2 :  4 ( 32- 22): +S(B)-S(Y2) SUR MATRICE M2
3 :  2 ( 21- 51): +Q(Z2)-Q(U5) SUR MATRICE M3
4 :  3 ( 11- 51): +Q(X1)-Q(U5) SUR MATRICE M4

INCOMPATIBLES AVEC:
5 :  1 ( 31- 12): +P(B)-P(X1) SUR MATRICE M5

ENSEMBLE RESPONSABLE:
1  2  3  5
ENSEMBLES RESPONSABLES(INCOMPAT.) JUSQU'ICI:
1  2  3  5
ENTREZ NO.(S) LIGNE(S) A AJOUTER AINSI:NO. LIGNE...NB. DE FOIS
...

```

L'ajout des variantes de clauses est ici nécessaire puisqu'il n'a pas eu de renommage des variables (voir § 4.2). Cet ajout peut être fait interactivement (possibilité de perte de la complétude) ou automatiquement.

EXEMPLE 2 :

(Traité avec RM "non binaire" au § 3.2.3.2)

RUN DEMO1

VOULEZ VOUS LIRE LES CLAUSES A PARTIR DE LA CONSOLE, D'UN FICHIER OU D

ES DEUX? (C/F/D)

F

ENTREZ LE NOM DU FICHIER(SANS SON TYPE);PRA

ENSEMBLE DE CLAUSES A REFUTER:

- C 1 +Q(X1,F(X1)) +P(X1)
- C 2 +P(A) -Q(A,F(A))
- +P(Y2)
- C 3 -P(Z3) +Q(U3,V3)
- +Q(A,F(A))
- C 4 -P(X4) -Q(W4,F(A))

REDUCTIONS:

- 1: 23- 41 = +P(Y2)-P(X4)
- 2: 23- 31 = +P(Y2)-P(Z3)
- 3: 21- 41 = +P(A)-P(X4)
- 4: 21- 31 = +P(A)-P(Z3)
- 5: 12- 41 = +P(X1)-P(X4)
- 6: 12- 31 = +P(X1)-P(Z3)
- 7: 33- 42 = +Q(A,F(A))-Q(W4,F(A))
- 8: 33- 22 = +Q(A,F(A))-Q(A,F(A))
- 9: 32- 42 = +Q(U3,V3)-Q(W4,F(A))
- 10: 32- 22 = +Q(U3,V3)-Q(A,F(A))
- 11: 11- 42 = +Q(X1,F(X1))-Q(W4,F(A))
- 12: 11- 22 = +Q(X1,F(X1))-Q(A,F(A))

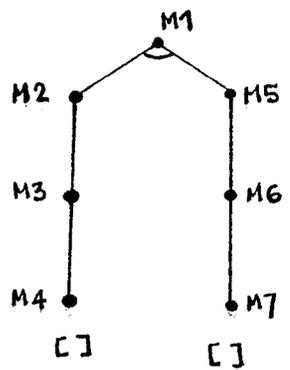
REDUCTIONS = Toutes les possibilités d'application de la RM. (Evidemment il n'est pas nécessaire de l'imprimer)

SCHEMA:

1 3 6 8 10 11 12

ARBRE DE REFUTATION:

MATRICE	FILS GAUCHE	FILS DROIT
M1	M2	M5
M2	M3	
M3	M4	
M4		
M5	M6	
M6	M7	
M7		



```

*****
ENSEMBLE DE CLAUSES A REFUTER:
C 1  +Q(X1,F(X1))                +P(X1)
C 2  +P(A)                        -Q(A,F(A))
                +P(Y2)
C 3  -P(Z3)                        +Q(U3,V3)
                +Q(A,F(A))
C 4  -P(X4)                        -Q(W4,F(A))
*****

```

```

*****
UN SCHEMA DE REFUTATION A ETE TROUVE; REDUCTIONS APPLIQUEES:
 1 : 12 ( 11- 22); +Q(X1,F(X1))-Q(A,F(A)) SUR MATRICE M1
 2 : 11 ( 11- 42); +Q(X1,F(X1))-Q(W4,F(A)) SUR MATRICE M2
 3 :  3 ( 21- 41); +P(A)-P(X4) SUR MATRICE M3
 4 :  1 ( 23- 41); +P(Y2)-P(X4) SUR MATRICE M4
 5 : 10 ( 32- 22); +Q(U3,V3)-Q(A,F(A)) SUR MATRICE M5
 6 :  8 ( 33- 22); +Q(A,F(A))-Q(A,F(A)) SUR MATRICE M6
 7 :  6 ( 12- 31); +P(X1)-P(Z3) SUR MATRICE M7
*****

```

```

*****
SUBSTITUTION DE REFUTATION:
X 1 <- A
Y 2 <- A
Z 3 <- A
U 3 <- A
V 3 <- F(A)
X 4 <- A
W 4 <- A
*****

```

ARBRE DE REFUTATION:

```

*****
MATRICE      FILS GAUCHE  FILS DROIT
M1           M2           M5
M2           M3
M3           M4
M4
M5           M6
M6           M7
M7

```

```

M1:
*****
ENSEMBLE DE CLAUSES A REFUTER:
C 1  +Q(X1,F(X1))                +P(X1)
C 2  +P(A)                        -Q(A,F(A))
                +P(Y2)
C 3  -P(Z3)                        +Q(U3,V3)
                +Q(A,F(A))
C 4  -P(X4)                        -Q(W4,F(A))
*****

```

M2:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +Q(X1,F(X1))

C 2 +P(A)

+P(Y2)

C 3 -P(Z3)

+Q(U3,V3)

+Q(A,F(A))

C 4 -P(X4)

-Q(W4,F(A))

M3:

ENSEMBLE DE CLAUSES A REFUTER:

C 1

C 2 +P(A)

+P(Y2)

C 3

C 4 -P(X4)

Les littéraux Q disparaissent
par application du principe
de pureté (§ 2.2.3.2)

M4:

ENSEMBLE DE CLAUSES A REFUTER:

C 1

C 2

+P(Y2)

C 3

C 4 -P(X4)

M5:

ENSEMBLE DE CLAUSES A REFUTER:

C 1

C 2

+P(X1)

-Q(A,F(A))

C 3 -P(Z3)

+Q(U3,V3)

+Q(A,F(A))

C 4 -P(X4)

-Q(W4,F(A))

ENSEMBLE DE CLAUSES A REFUTER:

C 1

+P(X1)

C 2

-Q(A,F(A))

C 3 -P(Z3)

+Q(A,F(A))

C 4 -P(X4)

-Q(W4,F(A))

M7:

ENSEMBLE DE CLAUSES A REFUTER:

C 1

+P(X1)

C 2

C 3 -P(Z3)

C 4

EXEMPLE 3 : [27]

RUN DEMO1

VOULEZ VOUS LIRE LES CLAUSES A PARTIR DE LA CONSOLE, D'UN FICHIER
OU D
ES DEUX? (C/F/D)

F

ENTREZ LE NOM DU FICHIER(SANS SON TYPE):G10

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(G(X1,Y1),X1,Y1)

C 2 -P(X2,H(X2,Y2),Y2)

C 3 -P(X3,Y3,U3)

+P(Y3,Z3,V3)

-P(X3,V3,W3)

+P(U3,Z3,W3)

C 4 -P(K(X4),X4,K(X4))

REDUCTIONS:

1: 34- 41 = +P(U3,Z3,W3)-P(K(X4),X4,K(X4))

2: 34- 21 = +P(U3,Z3,W3)-P(X2,H(X2,Y2),Y2)

3: 32- 41 = +P(Y3,Z3,V3)-P(K(X4),X4,K(X4))

4: 32- 21 = +P(Y3,Z3,V3)-P(X2,H(X2,Y2),Y2)

6: 11- 33 = +P(G(X1,Y1),X1,Y1)-P(X3,V3,W3)

7: 11- 31 = +P(G(X1,Y1),X1,Y1)-P(X3,Y3,U3)

8: 11- 21 = +P(G(X1,Y1),X1,Y1)-P(X2,H(X2,Y2),Y2)

On peut noter qu'il manque la réduction 5 (elle correspond à 11-41).
Ceci est dû au fait qu'un pretraitement l'a éliminée.
L'élimination a été faite en utilisant seulement la première phase
de l'algorithme d'unification; c'est pourquoi la réduction 8
reste (elle contient un cycle), et sera éliminée par la suite.

SCHEMA:

B

ARBRE DE REFUTATION:

MATRICE FILS GAUCHE FILS DROIT

M1

IL Y A UN CYCLE:

SCHEMA:

2 4 6 7

ARBRE DE REFUTATION:

MATRICE FILS GAUCHE FILS DROIT

M1

M2

M2

M3

M3

M4

M4

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(G(X1,Y1),X1,Y1)

C 2 -P(X2,H(X2,Y2),Y2)

C 3 -P(X3,Y3,U3)

+P(Y3,Z3,W3)

-P(X3,V3,W3)

+P(U3,Z3,W3)

C 4 -P(K(X4),X4,K(X4))

UN SCHEMA DE REFUTATION A ETE TROUVE; REDUCTIONS APPLIQUEES:

1 : 7 (11- 31); +P(G(X1,Y1),X1,Y1)-P(X3,Y3,U3) SUR MATRICE M1

2 : 6 (11- 33); +P(G(X1,Y1),X1,Y1)-P(X3,V3,W3) SUR MATRICE M2

3 : 4 (32- 21); +P(Y3,Z3,W3)-P(X2,H(X2,Y2),Y2) SUR MATRICE M3

4 : 2 (34- 21); +P(U3,Z3,W3)-P(X2,H(X2,Y2),Y2) SUR MATRICE M4

SUBSTITUTION DE REFUTATION:

X 1 <- Y1

Y 1 <- Y1

X 2 <- Y1

Y 2 <- Y1

X 3 <- G(Y1,Y1)

Y 3 <- Y1

U 3 <- Y1

Z 3 <- H(Y1,Y1)

V 3 <- Y1

W 3 <- Y1

X 4 <- X4

ARBRE DE RECHERCHE:

ARBRE DE REFUTATION:

MATRICE FILS GAUCHE FILS DROIT

M1 M2

M2 M3

M3 M4

M4

M1:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(G(X1,Y1),X1,Y1)

C 2 -P(X2,H(X2,Y2),Y2)

C 3 -P(X3,Y3,U3)

+P(Y3,Z3,W3)

-P(X3,V3,W3)

+P(U3,Z3,W3)

C 4 -P(K(X4),X4,K(X4))

M2:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(G(X1,Y1),X1,Y1)

C 2 -P(X2,H(X2,Y2),Y2)

C 3

-P(X3,U3,W3)

+P(Y3,Z3,U3)

C 4 -P(K(X4),X4,K(X4))

+P(U3,Z3,W3)

M3:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(G(X1,Y1),X1,Y1)

C 2 -P(X2,H(X2,Y2),Y2)

C 3

+P(Y3,Z3,U3)

C 4 -P(K(X4),X4,K(X4))

+P(U3,Z3,W3)

M4:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(G(X1,Y1),X1,Y1)

C 2 -P(X2,H(X2,Y2),Y2)

C 3

C 4 -P(K(X4),X4,K(X4))

+P(U3,Z3,W3)

Le même exemple peut être traité avec une version totalement interactive du démonstrateur, ce qui nous permet de trouver une autre solution.

∴ (Même ensemble de clauses)

UN SCHEMA DE REFUTATION A ETE TROUVE; REDUCTIONS APPLIQUEES:
1 : 8 (11- 21): +P(G(X1,Y1),X1,Y1)-P(X2,H(X2,Y2),Y2) SUR MATRICE

ARBRE DE REFUTATION:

MATRICE FILS GAUCHE FILS DROIT

M1

ACCEPTEZ-VOUS CE SCHEMA?(O/N)

O

IL Y A UN CYCLE;

VOULEZ-VOUS FINIR?(O/N)

N

VOULEZ-VOUS MODIFIER CE SCHEMA?(O/N)

O

A PARTIR DE QUELLE REDUCTION (1,2,...) VOULEZ-VOUS RECOMMENCER?

1

UN SCHEMA DE REFUTATION A ETE TROUVE;REDUCTIONS APPLIQUEES;

1 : 7 (11- 31); +P(G(X1,Y1),X1,Y1)-P(X3,Y3,U3) SUR MATRICE M1

2 : 6 (11- 33); +P(G(X1,Y1),X1,Y1)-P(X3,U3,W3) SUR MATRICE M2

3 : 4 (32- 21); +P(Y3,Z3,U3)-P(X2,H(X2,Y2),Y2) SUR MATRICE M3

4 : 2 (34- 21); +P(U3,Z3,W3)-P(X2,H(X2,Y2),Y2) SUR MATRICE M4

ARBRE DE REFUTATION;

MATRICE FILS GAUCHE FILS DROIT

M1 M2

M2 M3

M3 M4

M4

ACCEPTEZ-VOUS CE SCHEMA?(O/N)

N

VOULEZ-VOUS FINIR?(O/N)

N

VOULEZ-VOUS MODIFIER CE SCHEMA?(O/N)

O

A PARTIR DE QUELLE REDUCTION (1,2,...) VOULEZ-VOUS RECOMMENCER?

4

UN SCHEMA DE REFUTATION A ETE TROUVE;REDUCTIONS APPLIQUEES;

1 : 7 (11- 31); +P(G(X1,Y1),X1,Y1)-P(X3,Y3,U3) SUR MATRICE M1

2 : 6 (11- 33); +P(G(X1,Y1),X1,Y1)-P(X3,U3,W3) SUR MATRICE M2

3 : 4 (32- 21); +P(Y3,Z3,U3)-P(X2,H(X2,Y2),Y2) SUR MATRICE M3

4 : 1 (34- 41); +P(U3,Z3,W3)-P(K(X4),X4,K(X4)) SUR MATRICE M4

ARBRE DE REFUTATION;

MATRICE FILS GAUCHE FILS DROIT

M1 M2

M2 M3

M3 M4

M4

EXEMPLE 4 : [106]

RUN DEMO (Version totalement interactive du démonstrateur)

VOULEZ VOUS UNE TRACE?(O/N)

N

VOULEZ VOUS LIRE LES CLAUSES A PARTIR DE LA CONSOLE, D'UN FICHER
OU DES DEUX? (C/F/D)

F

ENTREZ LE NOM DU FICHER(SANS SON TYPE):PM

ENSEMBLE DE CLAUSES A REFUTER:

```
C 1  +P(X1)                                +Q(Y1)
      +R(X1,Y1)
      +S(A,X1)
C 2  -P(A)
C 3  -S(U3,U3)
C 4  -P(C)
C 5  -Q(W5)                                +S(W5,W5)
C 6  -R(Z6,Z6)
C 7  -R(U7,U7)                            +T(U7)
C 8  -T(A)
C 9  -S(B,B)
C10  -S(C,C)
*****
```

REDUCTIONS!

```
1: 11- 41 = +P(X1)-P(C)
2: 11- 21 = +P(X1)-P(A)
3: 12- 51 = +Q(Y1)-Q(W5)
4: 13- 71 = +R(X1,Y1)-R(U7,U7)
5: 13- 61 = +R(X1,Y1)-R(Z6,Z6)
6: 52-101 = +S(W5,W5)-S(C,C)
7: 52- 91 = +S(W5,W5)-S(B,B)
8: 52- 31 = +S(W5,W5)-S(U3,U3)
11: 14- 31 = +S(A,X1)-S(U3,U3)
12: 72- 81 = +T(U7)-T(A)
```

UN SCHEMA DE REFUTATION A ETE TROUVE;REDUCTIONS APPLIQUEES:

```
1 : 12 ( 72- 81): +T(U7)-T(A) SUR MATRICE M1
2 : 11 ( 14- 31): +S(A,X1)-S(U3,U3) SUR MATRICE M2
3 : 8 ( 52- 31): +S(W5,W5)-S(U3,U3) SUR MATRICE M3
4 : 5 ( 13- 61): +R(X1,Y1)-R(Z6,Z6) SUR MATRICE M4
5 : 3 ( 12- 51): +Q(Y1)-Q(W5) SUR MATRICE M5
6 : 2 ( 11- 21): +P(X1)-P(A) SUR MATRICE M6
```

ARBRE DE REFUTATION:

MATRICE FILS GAUCHE FILS DROIT

```
M1 M2
M2 M3
M3 M4
M4 M5
M5 M6
M6
```

ACCEPTEZ-VOUS CE SCHEMA?(O/N)

N

VOULEZ-VOUS FINIR?(O/N)

N

VOULEZ-VOUS MODIFIER CE SCHEMA?(O/N)

O

A PARTIR DE QUELLE REDUCTION (1,2,...) VOULEZ-VOUS RECOMMENCER?

4

UN SCHEMA DE REFUTATION A ETE TROUVE;REDUCTIONS APPLIQUEES:

```
1 : 12 ( 72- 81): +T(U7)-T(A) SUR MATRICE M1
2 : 11 ( 14- 31): +S(A,X1)-S(U3,U3) SUR MATRICE M2
3 : 8 ( 52- 31): +S(W5,W5)-S(U3,U3) SUR MATRICE M3
4 : 4 ( 13- 71): +R(X1,Y1)-R(U7,U7) SUR MATRICE M4
5 : 3 ( 12- 51): +Q(Y1)-Q(W5) SUR MATRICE M5
6 : 2 ( 11- 21): +P(X1)-P(A) SUR MATRICE M6
```

ARBRE DE REFUTATION:

```

*****
MATRICE      FILS GAUCHE  FILS DROIT
M1           M2
M2           M3
M3           M4
M4           M5
M5           M6
M6

```

```

*****
ACCEPTEZ-VOUS CE SCHEMA?(O/N)
0

```

SUBSTITUTION DE REFUTATION:

```

X 1 <- A
Y 1 <- A
U 3 <- A
W 5 <- A

```

```

U 7 <- A
V 7 <- A

```

ARBRE DE RECHERCHE:

ARBRE DE REFUTATION:

```

*****
MATRICE      FILS GAUCHE  FILS DROIT
M1           M2
M2           M3
M3           M4
M4           M5
M5           M6
M6

```

```

*****
VOULEZ VOUS IMPRIMER LES LITTERAUX COMPOSANTS DE CHAQUE
MATRICE DE LA REFUTATION?(O/N)
0

```

M1:

ENSEMBLE DE CLAUSES A REFUTER:

```

C 1  +P(X1)                +Q(Y1)
                +R(X1,Y1)

C 2  +S(A,X1)
C 3  -P(A)
C 4  -S(U3,U3)
C 5  -P(C)
C 6  -Q(W5)                +S(W5,W5)
C 7  -R(Z6,Z6)
C 8  -R(U7,V7)            +T(U7)
C 9  -T(A)
C10  -S(B,B)

```

M2:

ENSEMBLE DE CLAUSES A REFUTER:

```

C 1  +P(X1)                +Q(Y1)
                +R(X1,Y1)

C 2  +S(A,X1)
C 3  -P(A)
C 4  -S(U3,U3)
C 5  -P(C)
C 6  -Q(W5)                +S(W5,W5)
C 7  -R(Z6,Z6)
C 8  -R(U7,V7)
C 9  -S(B,B)
C10  -S(C,C)

```

-T(A) disparaît par application du
principe de pureté (§ 2.2.3.2)

M3:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(X1) +R(X1,Y1) +Q(Y1)

C 2 -P(A)
C 3 -S(U3,U3)
C 4 -P(C)
C 5 -Q(W5) +S(W5,W5)
C 6 -R(Z6,Z6)
C 7 -R(U7,V7)
C 8
C 9 -S(B,B)
C10 -S(C,C)

M4:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(X1) +R(X1,Y1) +Q(Y1)

C 2 -P(A)
C 3
C 4 -P(C)
C 5 -Q(W5)
C 6 -R(Z6,Z6)
C 7 -R(U7,V7)
C 8
C 9
C10

M5:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(X1) +Q(Y1)

C 2 -P(A)
C 3
C 4 -P(C)
C 5 -Q(W5)
C 6
C 7
C 8
C 9
C10

M6:

ENSEMBLE DE CLAUSES A REFUTER:

C 1 +P(X1)

C 2 -P(A)
C 3
C 4 -P(C)
C 5
C 6
C 7
C 8
C 9
C10

EXEMPLE 5 : (Utilisation du démonstrateur pour résoudre un système d'équations dans les termes)

Soit l'ensemble d'équations

$$E = \left\{ \underline{x} = \underline{f(y, a)} ; \underline{f(u, x)} = \underline{f(z, f(b, z))} \right\}$$

on introduit un symbole fonctionnel d'arité = à la cardinalité de E
(dans ce cas on peut utiliser f)

RUN DEMO1

VOULEZ VOUS LIRE LES CLAUSES A PARTIR DE LA CONSOLE, D'UN FICHIER
OU DES DEUX? (C/F/D)

F

ENTREZ LE NOM DU FICHIER (SANS SON TYPE); HUE

ENSEMBLE DE CLAUSES A REFUTER:

C 1 $\neg P(F(X1, F(U1, X1)))$

$\neg P(F(F(Y1, A), F(Z1, F(B, Z1))))$

C 2 $\neg P(X2)$

SUBSTITUTION DE REFUTATION:

X 1 $\leftarrow F(B, A)$

U 1 $\leftarrow A$

Y 1 $\leftarrow B$

Z 1 $\leftarrow A$

X 2 $\leftarrow F(F(B, A), F(A, F(B, A)))$

BIBLIOGRAPHIE

- [1] ACKERMANN W.
"Solvable cases for the decision problem"
North-Holland 1954.
- [2] AHO A.V., HOPCROFT J.E., ULLMAN J.D.
"The Design and Analysis of Computer Algorithms"
Addison-Wesley Pu. Co. 1976
- [3] ANDREWS P.B.
"Theorem proving via General matings"
JACM, Vol. 28, No 2, April 1981. 193-214
- [4] ANDREWS P.B.
"Refutations by matings"
IEEE Transactions on Computers Vol. C-25, No 8, August 1976. 801-807
- [5] BARWISE J. (Ed.)
"Handbook of mathematical logic"
North-Holland 1977
- [6] BARWISE J.
"An introduction to First-Order Logic"
dans [5]
- [7] BAUER M., BRAND D., FISCHER M.J., MEYER A.R., PATERSON M.S.
"A note on disjunctive form tautologies"
SIGACT News, April 1973. 17-20
- [8] BAXTER L.D.
"An efficient Unification Algorithm"
Technical Report CS-73-23.
Dept. of Applied Analysis and Computer Science,
University of Waterloo, 1973.
- [9] BAXTER L.D.
"A practically linear unification algorithm"
Res. Rep. CS-76-13, Dep. of Applied Analysis and Computer Science,
Univ. of Waterloo, Waterloo, Ontario, Canada 1976
- [10] BERGE C.
"Graphes et hypergraphes"
Dunod-Paris. 1973.
- [11] BIBEL W.
"Tautology testing with a generalized matrix reduction method"
Theoretical Computer Science 8, 1979. 31-44
- [12] BIBEL W.
"On matrices with connections"
JACM Vol.28, No.4, October 1981. 633-645
- [13] BIBEL W.
"A Comparative Study of Several Proof Procedures"
Artificial Intelligence 18, 1982. 269-293

- [14] **BIBEL W.**
"Automated theorem proving"
Vieweg, 1982
- [15] **BLANCHE R.**
"La logique et son histoire.d'Aristote à Russell"
Armand Colin, 1970
- [16] **BLEDSOE W.**
"Splitting and reduction heuristics in automatic theorem-proving"
Artificial Intelligence 2, 1971, 55-77
- [17] **BLEDSOE W.**
"Non-resolution theorem proving"
Artificial Intelligence 9, 1977, 1-36
- [18] **BLEDSOE W., BOYER R., HENNEMAN W.**
"Computer proofs of limit theorems"
Artificial Intelligence 3, 1972, 27-60
- [19] **BLEDSOE W., BRUELL P.**
"A man-machine theorem proving system"
Proc. of the I.J.C.A.I 1973, 56-65
- [20] **BOOLOS G.S., JEFFREY R.C.**
"Computability and logic"
Cambridge University Press 1974
- [21] **BOYER R.S., MOORE J.S.**
"A computational logic"
ACM Monograph Series, Academic Press, Inc.1979
- [22] **BOYER R.S., MOORE J.S.**
"The sharing of structure in theorem proving programs"
Machine Intelligence 7, Meltzer and Michie (Eds.)
Edinburgh University Press 1972, 101-116
- [23] **BUNDY A.**
"There is no best proof procedure"
SIGART Newsletter, December 1971, 6-7
- [24] **CAFERRA R.**
"Proof by matrix reduction as plan + validation"
6th. Conference on Automated Deduction.
Lectures Notes in Computer Science 138
Springer Verlag 1982, 309-325
- [25] **CHANG C.L**
"Theorem proving with variable-Constrained Resolution"
Information Sciences 4, 1972, 217-231
- [26] **CHANG C.L.**
"Resolution plans in theorem proving"
IBM Research Laboratory RJ 2469, 1979

- [27] **CHANG C.L., LEE R.**
"Symbolic logic and Mechanical theorem proving"
Academic Press, New York 1973
- [28] **CHANG C.L., SLAGLE J.R.**
"Using rewriting rules for connection graphs to prove theorems"
Artificial Intelligence 12, 1979, 159-180
- [29] **CHURCH A.**
"Introduction to mathematical logic"
Vol I. Princenton University Press 1956
- [30] **CHURCH A.**
"A note on the Entscheidungsproblem"
The Journal of Symbolic logic, Vol.1, 1936, 40-41, 101-102
- [31] **CHURCH A., QUINE W.V.O.**
"Some theorems on definability and decidability"
The Journal of Symbolic Logic Vol.17, No.3, 1952, 179-187
- [32] **COLMERAUER A.**
"Sur les bases théoriques de PROLOG"
Bulletin Gropian No.9, 1979, 128-152
- [33] **COOK S.A.**
"On the complexity of Theorem-Proving Procedures"
Proc. 3rd. Annual ACM Symposium on Theory of Computing 1971, 151-158
- [34] **COOK S., RECKHOW R.**
"On the lengths of proofs in the propositional calculus"
Preliminary version
Proc. of Sixth Annual ACM Symposium on Theory of Computing,
Seattle, Washington 1974, 135-148
(Voir "Corrections" à cet article, par les auteurs, dans
SIGACT News, July 1974, 15-22)
- [35] **COOK S., RECKHOW R.**
"The relative efficiency of propositional proof systems"
The Journal of Symbolic Logic Vol.44, No.1, March 1979, 36-50
- [36] **COX P.T.**
"Locating the source of unification failure"
Proc. of the Second National Conference of Canadian Society for
Computational Studies of the Intelligence
Toronto, July 1978, 20-29
- [37] **COX P.T.**
"Representational economy in a mechanical theorem proving"
4th. Workshop on Automated Deduction, Austin, Texas
February 1979, 122-128
- [38] **COX P.T., PIETRZYKOWSKI T.**
"Deduction plans: a basis for intelligent backtracking"
Research Report CS-79-41
Dept. of Computer Science, University of Waterloo Canada, December 1979

- [39] **COX P.T., PIETRZYKOWSKI T.**
 "Deduction Plans: A basis for Intelligent Backtracking"
 IEEE Trans. on Pattern Analysis and Machine Intelligence
 Vol. PAMI-3, No.1, January 1981, 52-65
- [40] **CURRY H.B**
 "Foundatlons of Mathematical Logic"
 Mc. Graw Hill, Dover Edition 1977
- [41] **DAVIS M.**
 "Eliminating the Irrelevant from mechanical proofs"
 Symposia In Applied Mathematics Vol.15, 1963, 15-30
- [42] **DAVIS M., PUTNAM H.**
 "A computing procedure for quantification theory"
 JACM Vol.7, No.3, 1960, 201-215
- [43] **DAVIS M., GURKEWITZ R., YARMUSH D.L.**
 "LOGIK:A special purpose language for writing theorem-provers"
 Courant Institute of Mathematical Sciences.
 New York University Report IMM 413, October 1976
- [44] **DE MILLO R.A., LIPTON R.J., PERLIS A.J.**
 "Social processes and proofs of theorems and programs"
 CACM Vol.22, No.5, May 1979, 271-280
- [45] **DOBKIN D.P., MUNRO J.I.**
 "Efficient uses of the past"
 21st Annual Symposium on Foundations of Computer Science, 1980, 200-206
- [46] **DREBEN B., GOLDFARB W.D.**
 "The decision problem Solvable Classes of quantificational formulas"
 Addison-Wesley Pu Co 1979
- [47] **ERNST G.W.**
 "The utility of Independent Subgoals in theorem proving"
 Information and Control 18, 1971, 237-252
- [48] **FISCHER M.J., RABIN M.O.**
 "Super exponential complexity of Presburger's arithmetic"
 SIAM-AMS Proceedings 7, 1974, 27-41
- [49] **FISHMAN D.H.**
 "A problem-oriented search procedure for theorem proving"
 IEEE Transactions on Computers, Vol.C-25, No.8, August 1976, 807-815
- [50] **FLEISIG S., LOVELAND D., SMILEY III A.K., YARMUSH D.L.**
 "An implementation of the model elimination proof procedure"
 JACM, Vol.21, No.1, January 1974, 124-139
- [51] **FREGE G.**
 "Begriffsschrift,a formula language, modeled upon that of arithmetic,
 for pure thought"
 1879 dans [154], 1-82

- [52] GALIL Z.
"On the complexity of regular resolution and the Davis-Putnam procedure"
Theoretical Computer Science 4, 1977, 23-46
- [53] GALLER B.A. FISHER M.J.
"An Improved equivalence Algorithm"
CACM, Vol.7, No.5, May 1964, 301-303
- [54] GENTZEN G.
"Recherches sur la déduction logique"
Presses Universitaires de France, 1955
- [55] GILMORE P.C.
"A proof method for quantification theory:its justification and realization"
IBM Journal of Res. and Dev., Jan 1960, 28-35
- [56] GODEL K
"Some metamathematical results on completeness and consistency. On formally undecidable propositions of Principia Mathematica and related systems I. and On completeness and consistency" 1930-1931. Dans [154], 592-617
- [57] GODEL K.
"The completeness of the axioms of the functional calculus of logic" 1930. Dans [154], 582-591
- [58] GODEL K.
"On the lengths of proofs"
dans The Undecidable. Davis M. (Ed.) Raven Press, 1965
- [59] GOLDBERG A.
"Average case complexity of the satisfiability problem"
Proc. of the Fourth Workshop on Automated Deduction, 1979, 1-6
- [60] GRATZER G.
"Universal Algebra"
Springer Verlag 2nd. Ed. 1979
- [61] HENSCHEN L.J.
"Semantic resolution for Horn Sets"
IEEE Transactions on Computers, Vol.C-25, No.8, August 1976
- [62] HENSCHEN L.J.
"Theorem Proving by covering expressions"
JACM, Vol.26, No.3, July 1979, 385-400
- [63] HENSCHEN L.J., WOS L.
"Unit refutations and Horn sets"
JACM, Vol.21, 1974, 590-605
- [64] HERBRAND J.
"Ecrits logiques"
Presses Universitaires de France, 1968

- [65] **HILBERT D., ACKERMANN W.**
"Principles of mathematical logic"
Chelsea Pub. Co., New York 1950
- [66] **HORN A.**
"On sentences which are true of direct unions of algebras"
Journal of Symbolic Logic Vol.16, 1951, 14-21
- [67] **HUET G.**
"Constrained resolution: A complete method for higher order logic"
Jennings Comp. Cen. Case Western Reserve Univ. Rep. 1117, 1972
- [68] **HUET G.**
"Résolution d'équations dans les langages d'ordre $1, 2, \dots, \omega$ "
Thèse d'état, Université Paris VII, 1976
- [69] **HUET G., OPPEN D.C.**
"Equations and Rewrite Rules: A Survey"
dans "Formal Languages Theory, Perspectives and Open Problems"
Ed. Ronald V. Book, Academic Press 1980, 349-405
- [70] **JENSEN K., WIRTH N.**
"Pascal: user manual and report"
2nd. Ed. Springer Verlag, 1978
- [71] **KANGER S.**
"A simplified proof method for elementary logic"
dans "Computer Programming and Formal Systems"
Eds. Braffort P. and Hirschberg, North-Holland 1963, 87-94
- [72] **KLEENE S.**
"Introduction to metamathematics"
Van Nostrand 1952
- [73] **KLEENE S.**
"Logique Mathématique"
Armand Colin 1971
- [74] **KLING R.E.**
"A paradigm for reasoning by analogy"
Artificial Intelligence 2, 1971, 147-148
- [75] **KOWALSKI R.**
"Logic for problem solving"
North-Holland 1979
- [76] **KOWALSKI R.**
"A proof procedure using Connection graphs"
JACM Vol.22, No.4, October 1975, 572-595
- [77] **KOWALSKI R.**
"Search Strategies for theorem proving"
Machine Intelligence 5, Meltzer and Michie (Eds.)
Edinburgh University Press 1969, 181-201

- [78] KOWALSKI R.
 "And-Or graphs, theorem-proving graphs and bi-directional Search"
 Machine Intelligence 7, Meltzer and Michie (Eds.)
 Edinburgh University Press 1972, 167-194
- [79] KOWALSKI R., HAYES P.J.
 "Semantic trees in automatic theorem-proving"
 dans Machine Intelligence 4, Meltzer et Michie (Eds.)
 Edinburgh University Press 1969, 87-101
- [80] KOWALSKI R., KUEHNER D.
 "Linear resolution with selection function"
 Artificial Intelligence 2, 1971, 227-260
- [81] KREISEL G.
 "On the kind of data needed for a theory of proofs"
 dans Gandy R., Hyland M.,(Eds.):Logic Colloquium 76
 North-Holland Pu. Co. 1977
- [82] KUEHNER D.G.
 "A note on the relation between resolution and Maslov's inverse method"
 Machine Intelligence 6, Meltzer and Michie (Eds.)
 Edinburgh University Press 1971, 73-76
- [83] LAO M.J.
 "A new data structure for the union-find problem"
 Information Processing Letters Vol.9, No.1, July 1979, 39-45
- [84] LEIBNIZ G.W.
 "Nouveaux essais sur l'entendement humain"
 (1703) Garnier-Flamarion, Paris 1966
- [85] LEWIS H.R., PAPADIMITRIOU C.H.
 "L'efficacité des algorithms"
 dans "Pour la Science", No.5, Mars 1978, 62-75
- [86] LOVELAND D.W.
 "Mechanical theorem proving by model elimination"
 JACM Vol.15, No.2, April 1968, 236-251
- [87] LOVELAND D.W.
 "A simplified format for the model elimination theorem-proving
 procedure"
 JACM Vol.16, July 1969, 349-363
- [88] LOVELAND D.W.
 "A unifying view of some linear Herbrand procedures"
 JACM Vol.19, No.2, April 1972, 366-384
- [89] LOVELAND D.
 "Automated Theorem Proving:A Logical Basis"
 North-Holland Pu. Co., Amsterdam, 1978

- [90] **LOWENHEIM L.**
 "On possibilities in the calculus of relatives"
 1915. dans [154] 228-251
- [91] **LUSK E.L., Mc CUNE W.W., OVERBEEK R.A.**
 "Logic machine architecture: kernel functions"
 6th. Conference on Automated Deduction
 Lectures Notes in Computer Science 138. Springer Verlag 1982. 70-84
- [92] **LUSK E.L., Mc CUNE W.W., OVERBEEK R.A.**
 "Logic machine architecture: Inference mechanisms"
 6th. Conference on Automated Deduction
 Lectures Notes in Computer Science 138. Springer Verlag 1982. 85-108
- [93] **MANNA Z.**
 "Mathematical Theory of Computation"
 Mc. Graw Hill 1974.
- [94] **MARTELLI A., MONTANARI U.**
 "Unification in linear time and space: a structured presentation"
 Universita di Pisa. Nota Interna B76-16. Juillet 1976
- [95] **MARTELLI A., MONTANARI U.**
 "Theorem proving with structure sharing and efficient unification"
 Universita di Pisa. Note Scientifiche S-77-7. Février 1977
- [96] **MARTELLI A., MONTANARI U.**
 "An efficient unification algorithm"
 ACM Transactions on Programming languages and systems.
 Vol.4. No.2. April 1982. 258-282
- [97] **MASLOV S.J.**
 "Proof search strategies for methods of the resolution types"
 Machine Intelligence 6. Meltzer and Michie (Eds.)
 Edinburgh University Press 1971. 77-90
- [98] **Mc CARTHY J., ABRAHAMS P.W., EDWARDS D.J., HART T.P., LEVIN M.I.**
 "LISP 1.5 Programmer's Manual"
 The M.I.T Press 1962.
- [99] **MELTZER B.**
 "The impossibility of Perfect Proof Procedures"
 AISB European Newsletter. Issue 15. Nov; 1973. 28-29
- [100] **MENDELSON E.**
 "Introduction to mathematical logic"
 D.Van Nostrand Co. 1964
- [101] **NEVINS A.J.**
 "A human oriented logic for Automatic Theorem-proving"
 JACM. Vol.21. No.4. October 1974. 606-621
- [102] **NILSSON N.J.**
 "Principles of Artificial Intelligence"
 Tioga Pu. Co. Palo Alto. 1980

- [103] NILSSON N.J.
"The interplay between experimental and theoretical methods in Artificial Intelligence"
Technical Note 229. SRI International. September 1980
- [104] OVERBEECK R., Mc CHAREN J., WOS L.
"Complexity and related enhancements for automated theorem-proving programs"
Comp.&Maths. with Apps., Vol.2. 1976. 1-16
- [105] PATERSON M.S., WEGMAN M.N.
"Linear unification"
Journal of Computer and System Sciences 16. 1978. 158-167
- [106] PIETRZYKOWSKI T., MATWIN S.
"Exponential Improvement of efficient backtracking. A strategy for plan-based deduction"
Proc. 6th. Conference on Automated Deduction.
Lectures Notes in Computer Science 138. Springer Verlag 1982. 223-239
- [107] PITRAT J.
"Un programme de démonstration de théorèmes"
DUNOD. 1970
- [108] PLAISTED D.A.
"Abstractions mappings in mechanical theorem proving"
Proc. of 5th. Conference on Automated Deduction.
Lectures Notes in Computer Science 87. Springer Verlag 1980. 264-280
- [109] PLAISTED D.A.
"Theorem proving with abstraction"
Artificial Intelligence 16. 1981. 47-108
- [110] PRAWITZ D.
"An improved proof procedure"
Theoria 26. 1960. 102-139
- [111] PRAWITZ D.
"Advances and problems in mechanical proof procedures"
Machine Intelligence 4. Meltzer and Michie (Eds.).
Edinburgh Univ. Press 1969. 59-71
- [112] PRAWITZ D.
"A proof procedure with matrix reduction"
Lectures Notes in mathematics 125. Symposium on Automatic Demonstration.
Springer Verlag 1970. 207-214
- [113] PRAWITZ D., PRAWITZ H., VOGHERA N.
"A mechanical proof procedure and its realization in an Electronic Computer"
JACM. Vol.7. No.1-2. 1960. 102-108
- [114] QUINE W.V.O.
"A way to simplify truth functions"
American Mathematical Monthly 62. 1956. 627-631

- [115] QUINE W.V.O.
"Méthodes de logique"
Armand Colin 1972
- [116] QUINE W.V.O.
"A proof procedure for quantification theory"
The Journal of Symbolic Logic Vol.20, No.2, June 1955, 141-149
- [117] RABIN M.O.
"Decidable Theories"
dans [5], 595-629
- [118] RABIN M.O.
"Theoretical Impediments to artificial Intelligence"
Information Processing 74, North-Holland Pu. Co. 615-619
- [119] RAULEFS P., SIEKMANN J., SZABO P., UNVERICHT E.
"A short survey on the state of the art in matching and unification Problems"
Universität Karlsruhe, Institut für Informatik I, SEKI 3-78, 1978
- [120] ROBINSON J.A.
"Logic : form and function, the mechanization of deductive reasoning"
University Press Edinburgh 1979
- [121] ROBINSON J.A.
"A machine oriented logic based on the resolution principle"
JACM, Vol.12, No.1, January 1965, 23-41
- [122] ROBINSON J.A.
"The Generalized Resolution Principle"
Machine Intelligence 3, Dale and Michie (Eds.),
Edinburgh University Press 1968, 77-93
- [123] ROBINSON J.A.
"Computational logic : The unification Computation"
Machine Intelligence 6, Meltzer and Michie (Eds.),
Edinburgh University Press 1971, 63-72
- [124] ROBINSON J.A.
"Theorem proving on the Computer"
JACM, Vol.10, No.2, 1963, 163-174
- [125] ROUSSEL P.
"PROLOG : Manuel de Référence et d'Utilisation"
Groupe d'Intelligence Artificielle, Université d'Aix-Marseille,
Luminy, Septembre 1975
- [126] SAYA H.
"Complétude de la stratégie du support dans la méthode de réduction matricielle"
R.R. IMAG No.93, Novembre 1977
- [127] SAYA H.
"A chain format for the linear matrix reduction proof procedure with selection function"
R.R. IMAG No.99, December 1977

- [128] SAYA H.
"Définition et utilisation des SL-graphes en démonstration automatique"
Thèse de Docteur Ingénieur. Université de Grenoble 1975
- [129] SAYA H., CAFERRA R.
"Une technique de représentation partagée des données pour la méthode de Prawitz en démonstration automatique"
R.R. IMAG No.63, Janvier 1977
- [130] SAYA H., CAFERRA R.
"A structure sharing technique for matrices and substitutions in Prawitz's theorem proving method"
R.R. IMAG No.101, December 1977
- [131] SAYA H., CAFERRA R.
"Représentation compacte des matrices et traitement de l'égalité formelle dans la méthode de Prawitz de démonstration automatique"
Congrès AFCET 1978, Tome 2, 371-381
- [132] SHOENFIELD J.R.
"Mathematical Logic"
Addison-Wesley Pu. Co. 1967
- [133] SHOSTAK R.E.
"On separating the first-order from the propositional in resolution theorem proving"
Report # 20-72 Center for research in computing technology
Harvard University 1972
- [134] SHOSTAK R.E.
"A graph-theoretic view of resolution theorem proving"
TR 20-74 Center of research in Computing Technology 1974
- [135] SHOSTAK R.E.
"On the role of unification in Mechanical Theorem Proving"
Acta Informatica 7, 1977, 319-323
- [136] SHOSTAK R.E.
"Refutations graphs"
Artificial Intelligence 7, 1976, 51-64
- [137] SICKEL S.
"A search Technique for Clause Interconnectivity Graphs"
IEEE Transactions on Computers, Vol.C-25, No.8, August 1976, 823-835
- [138] SICKEL S.
"Variable range restrictions in resolution theorem proving"
Machine Intelligence 8, Elcock and Michie (Eds.)
Ellis Horwood Ltd 1977, 73-85
- [139] SIEKMANN J.
"Unification and Matching Problems"
Essex University, Great Britain, March 1978
- [140] SIEKMANN J.
"String-Unification"
Part I, Essex University, March 1975

- [141] **SIEKMANN J.**
 "Unification of Commutative Terms"
 Institut für Informatik I Universität Karlsruhe Int. Ber. 1976
- [142] **SIEKMANN J., WRIGHTSON G.**
 "Paramodulated Connection Graphs"
 Acta Informatica 13. 1980. 67-86
- [143] **SIMON H.A., KADANE J.B.**
 "Problems of Computational Complexity in Artificial Intelligence"
 dans Traub J.E (Ed.) : "Algorithms and Complexity. New directions and recent results". Academic Press 1976
- [144] **SMITH D.C., ENEA H.J.**
 "Backtracking in MLISP2. An efficient backtracking method for LISP"
 Proc. of the 3rd. IJCAI 1973. 677-685
- [145] **STICKEL M.**
 "The programmable strategy theorem prover: An implementation of the linear MESON procedure"
 Comp. Sci. Dept., Carnegie Mellon Univ., Pittsburgh, PA. June 1974
- [146] **STILLMAN R.B.**
 "The concept of weak substitution in Theorem-Proving"
 JACM. Vol.20. No.4. October 1973. 648-667
- [147] **STOCKMEYER L.J., CHANDRA A.K.**
 "Intrinsically Difficult Problems"
 Scientific American. May 1979. 124-133
- [148] **TARJAN R.E.**
 "Efficiency of a Good but no linear set union algorithm"
 JACM. Vol.22. No.2. April 1975. 215-225
- [149] **TARJAN R.E.**
 "Complexity of combinatorial algorithms"
 SIAM Review Vol.20. No.3. July 1978. 457-491
- [150] **TARSKI A.**
 "Fundamental Concepts of the methodology of the deductive sciences"
 dans [152]. 60-109
- [151] **TARSKI A.**
 "The concept of truth in formalized languages"
 dans [152]. 152-278
- [152] **TARSKI A.**
 "Logic, Semantics, Metamathematics"
 Papers from 1923 to 1938
 Oxford-at the Clarendon Press 1956
- [153] **TSEITIN G.S.**
 "On the complexity of derivations in the propositional calculus"
 Structures in Constructive Mathematics and Mathematical Logic.
 Part II. Silenko A.O. (Ed.). 1968. 115-125

- [154] VAN HEIJENOORT (Ed.)
"From Frege to Gödel : A source book in mathematical logic, 1879-1931"
Harvard University Press 1971
- [155] VAN VAALEN J.
"An extension of unification with an application to automatic theorem proving"
Stichting Mathematisch Centrum, Amsterdam, The Netherlands, 1974
- [156] VAN WESTRHENEN S.C.
"Statistical studies of theoremhood in Classical propositional and first order predicate calculus"
JACM, Vol.19, No.2, April 1972, 347-365
- [157] WANG H.
"Towards mechanical mathematics"
IBM J. Research Dev. 4, 1960, 2-22
- [158] WILSON G.A., MINKER J.
"Resolution, Refinements and Search Strategies : A Comparative Study"
IEEE Transactions on Computers, August 1976, 782-801
- [159] WIRTH N.
"Algorithms + Data Structure = Programs"
Prentice Hall 1976
- [160] WITTGENSTEIN L.
"Tractatus logico-philosophicus"
(1921) Editions Gallimard 1961
- [161] WOS L., ROBINSON G.A., CARSON D.F.
"Efficiency of the set of support strategy in theorem proving"
JACM, Vol.12, No.4, October 1965, 536-541
- [162] YARMUSH D.L.
"The linked conjunct and other algorithms for mechanical theorem-proving"
Courant Institute of Mathematical Sciences New York University
Report IMM 412, July 1976
- [163] YATES R.A., RAPHAEL B., HART T.P.
"Resolution graphs"
Artificial Intelligence 1, 1970, 257-289
- [164] YELOWITZ L., KANDEL A.
"New Results and Techniques in Resolution Theory"
IEEE Transactions on Computers, Vol.C-25, No.7, July 1976, 673-677

Dernière page d'une thèse

VU

Grenoble, le 10.11.82.

Le Président de la thèse



Vu, et permis d'imprimer,

Grenoble, le

Le Président de l'Université Scientifique et Médicale



Le 10.11.82
M. TANCHE