Définition et utilisation des S-L graphes en démonstration automatique

Henri Saya

▶ **To cite this version:**

Henri Saya. Définition et utilisation des S-L graphes en démonstration automatique. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG; Université Joseph-Fourier - Grenoble I, 1975. Français. NNT : . tel-00286254

HAL Id: tel-00286254

https://theses.hal.science/tel-00286254v1

Submitted on 9 Jun 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

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

POUR OBTENIR LE GRADE DE

DOCTEUR INGENIEUR

Henri SAYA

DEFINITION ET UTILISATION DES S-L GRAPHES EN DEMONSTRATION AUTOMATIQUE

Soutenue le 19 mars 1975 devant la Commission d'Examen :

Président : Monsieur B. VAUQUOIS

Examinateurs : Messieurs A. COLMERAUER
R. KOWALSKI
C. BENZAKEN

Rapporteur : Monsieur G. VEILLON

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

M. Michel SOUTIF — Présidents — M. Louis NEEL

M. Gabriel CAU — Vice-Présidents — MM. Lucien BONNETAIN, Jean BENOIT

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

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Clinique de Pédiatrie et Puériculture
	BERNARD Alain	Mathématiques Pures
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BEZES Henri	Pathologie chirurgicale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BOUCHERLE André	Chimie et toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARLIER Georges	Biologie végétale
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Clinique Oto-Rhino-Laryngologique
	CHATEAU Robert	Thérapeutique (Neurologie)
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBERMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DELORMAS Pierre	Pneumo-Phtisiologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée

MM.	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DRUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de dermatologie et syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques pures
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Mathématiques appliquées
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	JANIN Bernard	Géographie
	KAHANE André	Physique Générale
	KLEIN Joseph	Mathématiques pures
	KOSZUL Jean-Louis	Mathématiques pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mlle	LUTZ Elisabeth	Mathématiques pures
	MALGRANGE Bernard	Mathématiques pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et pétrographie
	MICOUD Max	Clinique maladies infectieuses
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	MULLER Jean-Michel	Thérapeutique (néphrologie)
	NEEL Louis	Physique du solide
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	RINALDI Renaud	Physique
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et hygiène
	SENGEL Philippe	Zoologie
	SIBILLE Robert	Construction mécanique
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
MM.	VERAIN André	Physique
	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	CHEEKE John	Thermodynamique
	COPPENS Philip	Physique
	CORCOS Gilles	Mécanique
	CRABBE Pierre	CERMO
	GILLESPIE John	I.S.N.
	ROCKAFELLAR Ralph	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

Mlle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBROISE-THOMAS Pierre	Parasitologie
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BERTRANDIAS Jean-Paul	Mathématiques pures
	BIAREZ Jean-Pierre	Mécanique
	BILLET Jean	Géographie
Mme	BONNIER Jane	Chimie générale
MM.	BOUCHET Yves	Anatomie
	BRUGEL Lucien	Energétique
	CONTE René	Physique
	DEPASSEL Roger	Mécanique des fluides
	GAUTHIER Yves	Sciences biologiques
	GAUTRON René	Chimie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	GROULADE Joseph	Biochimie médicale
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Méd. Préventive
	IDELMAN Simon	Physiologie animale
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme	KAHANE Josette	Physique
MM.	KUHN Gérard	Physique
	LOISEAUX Jean	Physique nucléaire
	LUU-DUC-Cuong	Chimie organique
	MAYNARD Roger	Physique du solide
	PELMONT Jean	Biochimie
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Mlle	PIERY Yvette	Physiologie animale
MM.	RAYNAUD Hervé	Mathématiques appliquées
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	RICHARD Lucien	Biologie végétale
Mme	RINAUDO Marguerite	Chimie macromoléculaire
MM.	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
MM.	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	AMBLARD Pierre	Dermatologie
	ARMAND Gilbert	Géographie
	ARMAND Yves	Chimie
	BARGE Michel	Neurochirurgie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamique
M.	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BRODEAU François	Mathématiques (IUT B)
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHARDON Michel	Géographie
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CORDONNIER Daniel	Néphrologie
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DELOBEL Claude	M.I.A.G.
	DENIS Bernard	Cardiologie
	DOUCE Roland	Physiologie végétale
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FONTAINE Jean-Marc	Mathématiques pures
	GAUTIER Robert	Chirurgie générale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques appliquées
	GROS Yves	Physique (stag.)
	GUITTON Jacques	Chimie
	HICTER Pierre	Chimie
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	KOLODIE Lucien	Hématologie
	KRAKOWIAK Sacha	Mathématiques appliquées
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LEROY Philippe	Mathématiques
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (IUT A)
Mme	MINIER Colette	Physique
MM.	NEGRE Robert	Mécanique
	NEMOZ Alain	Thermodynamique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PERRET Jean	Neurologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAMBAUD Pierre	Pédiatrie
Mme	RENAUDET Jacqueline	Bactériologie
MM.	ROBERT Jean-Bernard	Chimie-Physique

MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	COLE Antony	Sciences nucléaires
	FORELL César	Mécanique
	MOORSANI Kishin	Physique

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

MM.	BOST Michel	Pédiatrie
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	FAURE Gilbert	Urologie
	MALLION Jean-Michel	Médecine du travail
	ROCHAT Jacques	Hygiène et hydrologie

Fait à Saint Martin d'Hères, OCTOBRE 1974.

"MEMBRES DU CORPS ENSEIGNANT DE L'I.N.P.G."

PROFESSEURS TITULAIRES

MM. BENOIT Jean	Radioélectricité
BESSON Jean	ELectrochimie
BONNETAIN Lucien	Chimie Minérale
BONNIER Etienne	Electrochimie, Electrométallurgie
BRISSONNEAU Pierre	Physique du solide
BUYLE-BODIN Maurice	ELectronique
COUMES André	Radioélectricité
FELICI Noël	Electrostatique
PAUTHENET René	Physique du solide
PERRET René	Servomécanismes
SANTON Lucien	Mécanique
SILBER Robert	Mécanique des fluides

PROFESSEUR ASSOCIE

M. BOUDOURIS Georges	Radioélectricité

PROFESSEURS SANS CHAIRE

MM. BLIMAN Samuel	Electronique
BLOCH Daniel	Physique du solide et cristallographie
COHEN Joseph	Electrotechnique
DURAND Françis	Metallurgie
MOREAU René	Mécanique
POLOUJADOFF Michel	ELectrotechnique
VEILLON Gérard	Informatique fondamentale et appliquée
ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM. BOUVARD Maurice	Génie mécanique
CHARTIER Germain	Electronique
FOULARD Claude	Automatique
GUYOT Pierre	Chimie minérale
JOUBERT Jean Claude	Physique du solide
LACOUME Jean Louis	Géophisique
LANCIA Roland	Physique atomique
LESPINARD Georges	Mécanique
MORET Roger	Electrotechnique nucléaire
ROBERT François	Analyse numérique
SABONNADIERE Jean Claude	Informatique fondamentale et appliquée
Mme SAUCIER Gabrièle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M. LANDAU Ioan Doré	Automatique

CHARGE DE FONCTIONS DE MAITRES DE CONFERENCES

M. ANCEAU François	Mathématiques appliquées

Fait à Saint Martin d'Hères JANVIER 1974

Je tiens à remercier :

Monsieur B. VAUQUOIS, Professeur à l'Université de Grenoble I, qui m'a fait l'honneur de présider le jury de cette thèse. Ses cours sont à l'origine de mon intérêt pour l'étude de la logique mathématique et des systèmes formels,

Monsieur G. VEILLON, Professeur à l'Institut National Polytechnique de Grenoble, qui, tout en dirigeant mes recherches, a su me laisser suffisamment d'autonomie dans mon travail. Les nombreuses discussions que nous avons eues ensemble m'ont permis de clarifier mes idées et d'éviter certaines erreurs,

Monsieur A. COLMERAUER, Professeur à l'Université d'Aix-Marseille, qui a bien voulu s'intéresser à mon travail. Chacune de nos rencontres a été fructueuse et a été le point de départ d'améliorations de mon système,

Monsieur R. KOWALSKI, Professeur à l'Imperial College of Science and Technology de Londres, qui a bien voulu donner un avis de spécialiste sur mon travail. Les résultats de ses recherches sont à la base de mon travail et les critiques constructives qu'il a formulées, au cours de l'élaboration de mon démonstrateur, ont été fort appréciées,

Monsieur C. BENZAKEN, Professeur à l'Université de Grenoble I, qui a bien voulu juger mon travail avec beaucoup d'intérêt.

Je remercie également :

Michel LEVY et Henri AJENSTAT, avec lesquels j'ai eu beaucoup de contacts et qui, par leurs conseils et encouragements, m'ont fourni une aide non négligeable,

Augustin LUX, dont les travaux sur l'amélioration du système LISP fonctionnant au laboratoire m'ont été très utiles, et dont l'aide, notamment pour la programmation de mon système, a été précieuse,

Le Centre National de la Recherche Scientifique, sans l'aide financière duquel ce travail n'aurait pu voir le jour.

Je remercie enfin Mademoiselle NAUDIN qui s'est acquittée du travail de dactylographie avec gentillesse et application, Monsieur RASOLONJATOVO qui a dessiné les figures, ainsi que tout le personnel du service de reproduction, pour son amabilité et sa compétence.

H. SAYA

TABLE DES MATIERES

3ème Partie : MISE EN OEUVRE D'UN DEMONSTRATEUR SUR ORDINATEUR

VIII - Description des systèmes réalisés
- VIII.1 - Méthodologie
- VIII.2 - Choix du langage
- VIII.3 - Mise en oeuvre de la stratégie diagonale montante
- VIII.4 - Représentation interne d'un S-L graphe
- VIII.5 - Principes de simplification d'un S-L graphe
  - VIII.5.1 - Suppression des tautologies
  - VIII.5.2 - Suppression des liens incompatibles
- VIII.6 - Différences entre les deux versions du démonstrateur
  - VIII.6.1 - Structure de données
  - VIII.6.2 - Algorithmes

IX - Expérimentation du système
- IX.1 - Liste des exemples traités
- IX.2 - Paramètres de contrôle
- IX.3 - Variabilité de l'espace liste
- IX.4 - Influence de la formalisation
- IX.5 - Ordre des clauses et des littéraux
- IX.6 - Utilisation de la subsomption
- IX.7 - Choix de différents supports

X - Extensions et modifications du système
- X.1 - Prise en compte des A-littéraux
- X.2 - Test de subsomption
- X.3 - Génération de lemmes
- X.4 - Raffinements de la fonction heuristique
- X.5 - Traitement de formules du second ordre
- X.6 - Littéraux et fonctions évaluables
- X.7 - Suppression de certaines redondances
- X.8 - Nouveaux paramètres de contrôle

"Ce que je demande, ce sont des critères de la vérité qui soient palpables et qui ne laissent pas plus de place au doute que des calculs sur les nombres .....
Ce qu'il faut, ce sont des marques palpables de ce qui est clair et distinct, puisque souvent les hommes sont en désaccord là-dessus".

G.W.Leibniz
(Nouveaux essais sur l'entendement humain)

I - INTRODUCTION

Depuis le début des années 60, on assiste au développement d'une discipline nouvelle que l'on a convenu d'appeler "Intelligence artificielle". Des techniques sophistiquées ont été mises au point et ont permis le développement rapide de ce domaine de recherche. L'objet de ces techniques est la résolution, à l'aide d'un ordinateur, de problèmes qui présupposent, pour le sens commun, l'intervention d'une intelligence humaine.

A l'intérieur de ce domaine, une place prépondérante est accordée à l'étude de la démonstration automatique car celle-ci possède de nombreuses applications ; en effet, outre la démonstration de théorèmes mathématiques (par exemple GUARD et al. 1969), on peut citer :

- Les systèmes de questions-réponses (par exemple COLMERAUER et al. 1972)
- les systèmes de documentation automatique (par exemple DARLINGTON 1969)
- la vérification de programmes (par exemple MANNA 1969)
- la synthèse de programmes (par exemple MANNA et WALDINGER 1971)
- les systèmes de guidage de robots (par exemple GREEN 1969)
- et, d'une manière générale, tout système utilisant un processus déductif (par exemple, systèmes de résolution de problèmes : NILSSON 1971).

La recherche de techniques de démonstration automatique est très ancienne : LEIBNIZ au XVIIème siècle, PEANO à la fin du siècle dernier, l'école de HILBERT vers 1920, ont cherché tour à tour une procédure de décision générale qui déterminerait, le cas échéant, le caractère valide ou contradictoire d'une formule logique.

CHURCH et TURING ont démontré (1936) l'impossibilité de trouver une telle procédure, dès que l'on se place dans la logique des prédicats

du 1er ordre. On sait, en outre, depuis les travaux de HERBRAND (1930), qu'il est néanmoins possible de construire une procédure de preuve qui vérifie qu'une formule est valide quand celle-ci l'est effectivement, mais qui peut ne pas se terminer dans le cas contraire.

Les premières tentatives d'utilisation sur ordinateur des résultats de HERBRAND eurent lieu aux alentours des années 60 (GILMORE, PRAWITZ, DAVIS et PUTNAM, WANG). Un pas décisif fut franchi en 1965, lorsque ROBINSON proposa une règle d'inférence unique, la résolution, pour démontrer qu'un ensemble de clauses est contradictoire.

Depuis lors, de nombreux raffinements ont été apportés à la résolution. Parmi ces raffinements, citons, par exemple

- la résolution avec ensemble de support (WOS et al. 1965)
- la résolution linéaire (LOVELAND 1970a, LOCKHAM 1970)
- la S-L résolution et la procédure de Model-elimination (KOWALSKI et KUEHNER 1970, LOVELAND 1970b)

qui ont amélioré sensiblement les performances des procédures de preuve sans, pour autant, en diminuer le domaine d'application.

Il existe une autre approche plus intuitive (qui n'est pas celle que nous avons choisie dans cette étude), de la démonstration automatique, fondée sur la décomposition d'un problème en sous-problèmes successifs utilisant les arbres ET/OU en conjonction avec un processus analogue à l'unification (GELERNTER 1959, NEWELL et al. 1057, HEWITT 1972). Plus récemment, LOVELAND et STICKEL (1973) ont utilisé la résolution pour pallier certains inconvénients de cette dernière approche, s'efforçant ainsi de combiner les deux démarches.

En France, les programmes de démonstration fondés sur les travaux de HERBRAND sont assez rares : NGOA (1967) a comparé les algorithmes de ROBINSON et de DAVIS et PUTNAM, ainsi que diverses stratégies, et a étudié quelques sous-classes décidables du calcul des prédicats du 1er ordre ; COLMERAUER et ROUSSEL (1972) ont réalisé, dans le cadre d'un système de questions-réponses, un démonstrateur basé sur la S-L, résolution qui traite l'égalité formelle, HUET (1973) a étudié les problèmes posés par

la mécanisation de la logique d'ordre supérieur. Dans une optique différente, PITRAT (1966) a utilisé la définition axiomatique des systèmes formels dans un programme général qui fait des déductions en utilisant des métathéorèmes et qui démontre des théorèmes "intéressants" (selon un critère d'intérêt prédéfini), DUPRAZ (1966) a essayé d'utiliser les techniques d'algèbre de Boole pour la démonstration automatique ; signalons enfin d'autres travaux, utilisant l'approche intuitive de la démonstration (OUABDESSELAM, équipe de PITRAT), dans lesquels le domaine d'application est restreint à un certain type de formules arithmétiques, le principe étant de se ramener par une succession d'opérations de réduction, de combinaison et de normalisation, à des formules connues placées dans une base de données.

Il est curieux de remarquer que l'opposition et la controverse entre cette dernière approche et les approches précédentes existaient déjà chez les logiciens du XVIIème siècle : le courant intuitif était représenté par DESCARTES et le courant formaliste par LEIBNIZ. Notons, au passage, que les 4 règles énoncées en 1637 par DESCARTES dans "Le discours de la méthode" (règles de l'évidence, de l'analyse, de la synthèse, du dénombrement) sont reprises sous une forme assez semblables dans l'approche intuitive de la démonstration automatique.

L'objet du travail que nous exposons est l'étude et la mise en oeuvre de procédures de preuve pour le problème suivant :

Problème P1 - Vérifier qu'une formule bien formée A est une conséquence logique d'un ensemble de formules bien formées Γ :

$$\Gamma \models A ,$$

A et les formules de Γ étant des formules du calcul des prédicats du 1er ordre.

Nous montrerons comment on peut passer du problème P1 aux problèmes suivants successivement équivalents :

Problème P2 - Vérifier qu'une formule bien formée A est contradictoire.

Problème P3 - Vérifier qu'un ensemble de clauses donné est contradictoire.

Problème P4 - Trouver la clause vide dans un certain espace de recherche de clauses.

L'originalité de mon travail consiste en une utilisation conjointe de plusieurs méthodes et techniques (qui, considérées séparément, ne sont pas originales) pour la mise en oeuvre sur ordinateur d'un système de démonstration automatique. La méthode de déduction utilisée a été obtenue en combinant et en modifiant la méthode de S-L résolution et la méthode des graphes de classification (KOWALSKI, 1973). J'ai appelé cette nouvelle méthode "méthode des S-L graphes". Les résultats d'une première expérimentation ainsi que des remarques faites par R. KOWALSKI, m'ont incité à n'utiliser qu'une forme plus simple et plus restreinte de cette dernière méthode : c'est la "méthode des S-L graphes simplifiée".

Pour explorer l'espace de recherche défini par cette méthode, j'ai choisi la stratégie diagonale montante (HART-NILSSON-RAPHAEL 1968, POHL 1969, KOWALSKI 1970) qui permet, dans certaines conditions, de trouver une solution de coût optimal. En partant du principe qu'il vaut mieux tenter de trouver une solution peu coûteuse (en temps et en mémoire utilisés) par une méthode incomplète que de chercher une solution "certaine" (i.e. par une méthode complète) mais plus coûteuse, j'ai modifié la stratégie diagonale en y incluant divers tests qui interdisent de dépasser une complexité donnée.

La stratégie peut, de toute façon, être rendue complète par le choix d'une norme de complexité infinie. De plus, les paramètres relatifs à ces tests, ainsi que d'autres relatifs au contrôle du déroulement de la démonstration, peuvent être connus et modifiés dynamiquement pendant la démonstration (ce qui s'avère très utile dans certains cas).

La démonstration, bien que pouvant fonctionner de façon autonome, pourra être utilisée en tant que module d'un système plus important contenant d'autres modules, en cours de réalisation ou en projet, tels que, par exemple :

un analyseur de langues naturelles (COURTIN, 1973), un synthétiseur-analyseur de formules logiques, le S-L graphe étant un des supports de communication entre les modules. Cette communication servirait, par exemple, à communiquer au démonstrateur un texte et des questions en

français, le démonstrateur pouvant à son tour poser des questions ou faire connaître à l'utilisateur, en français, l'état de ses déductions. On peut également envisager d'utiliser les capacités déductives du démonstrateur pour le fonctionnement interne des autres modules.

Nous n'aborderons que très peu, dans cette étude, la question du choix, pour un problème donné, parmi les diverses formalisations possibles de ce problème (notamment choix des prédicats et des fonctions) ; nous supposerons cette question réglée lors de la soumission du problème au démonstrateur.

Le démonstrateur a été décrit dans une version améliorée de LISP 1.5 (MAC CARTHY 1961, LUX 1973) utilisée dans l'environnement du système conversationnel et à temps partagé CP-CMS de l'ordinateur IBM/360-67 de l'Université scientifique et médicale de Grenoble I.

Nous présenterons dans les premiers chapitres les fondements théoriques de la démonstration automatique en nous efforçant d'unifier les concepts ; les chapitres suivants auront trait aux procédures de preuve (composées d'une méthode de déduction et d'une stratégie de recherche), et à la présentation des méthodes "S-L graphes" et "S-L graphes simplifiés"; nous décrirons ensuite le système réalisé ; après un chapitre consacré à l'expérimentation du système sur un grand nombre d'exemples, nous décrirons quelques extensions et modifications qu'il serait envisageable de faire au système actuel.

La démonstration automatique étant encore une discipline récente, le vocabulaire qu'elle utilise n'est pas encore normalisé. C'est pourquoi nous définirons tous les termes que nous emploieront (lorsqu'ils sont spécifiques au domaine) ; pour certains, nous donnerons quelques synonymes rencontrés dans la littérature ; en général, un terme a été choisi en fonction de sa fréquence d'apparition dans la littérature et/ou en fonction de son contenu sémantique pour le sens commun.

Signalons enfin quelques ouvrages qui nous ont paru utiles et intéressants pour une initiation à la démonstration automatique :

- le livre de NILSSON (1971), et plus particulièrement ses chapitres 6 et 8, constituent une introduction claire, concise et

attrayante, à ce domaine

- le rapport de PIROTTE (1972) contient, dans un volume restreint, un recensement quasi-exhaustif des raffinements de la résolution
- le livre de CHANG et LEE (1973) fourmille d'exemples et d'exercices.

PREMIERE PARTIE

FONDEMENTS THEORIQUES DE LA DEMONSTRATION AUTOMATIQUE

- Systèmes logiques
- Le calcul des prédicats du 1er ordre
- Théorème et procédures de HERBRAND

II - SYSTEMES LOGIQUES

Les définitions et propriétés énoncées dans ce chapitre ont été réduites au minimum.

Pour une étude plus approfondie, on pourra se reporter par exemple à MENDELSON (1964) ou KLEENE (1971).

II.1 - Définitions

Il existe deux approches possibles pour la définition d'un système logique :

- l'approche axiomatique formelle qui, se référant à des concepts syntaxiques, définit des relations entre symboles et expressions d'un langage

- l'approche sémantique qui fait appel à des notions d'interprétation, de vérité, de modèles, en se référant à la théorie des ensembles.

Les deux approches ont en commun la définition d'un langage au moyen d'un vocabulaire et de règles permettant de construire des formules bien formées (f.b.f.).

Dans la première approche (appelée parfois approche syntaxique ou encore, déductive), on définit en outre :

- un ensemble d'axiomes qui sont les théorèmes du système qui ne nécessitent aucune démonstration

- un ensemble de règles d'inférence qui permettent de générer, à partir de l'ensemble des axiomes, les théorèmes du système logique.

Dans cette approche, on s'intéresse aux théorèmes du système, et le problème fondamental est le suivant (problème analogue au problème du mot en théorie des langages) :

Etant donnée une f.b.f. quelconque A, du système, peut-on affirmer, ou nier, la qualité de théorème de A ?

Ce qui se note : $\vdash A$ ou $\nvdash A$?

Dans l'approche sémantique, on s'attache à donner une signification aux f.b.f. Pour ce faire, on définit l'interprétation d'une f.b.f., à l'aide de règles permettant de lui affecter une valeur de vérité (vrai ou faux), ces règles dépendant de l'interprétation considérée.

Une f.b.f. A est dite satisfaisable (ou réalisable ou consistante), si elle est rendue vraie par une interprétation particulière J. On dit alors que A est satisfaite par J.

Une f.b.f. est dite insatisfaisable ou contradictoire, si elle n'est vraie pour aucune interprétation.

Une f.b.f. est dite logiquement valide si elle est vraie pour toutes les interprétations possibles.

En résumé, on peut classer les f.b.f. selon le schéma suivant.

valides	neutres	insatisfaisables
vrai pour toute interprétation	vrai pour 1 interprétation fausse pour 1 interprétation au moins	fausse pour toute interprétation

non valides

satisfaisables

Les définitions de satisfaisabilité et d'insatisfaisabilité s'étendent de façon naturelle à un ensemble de f.b.f. :

Un ensemble de f.b.f. Γ est satisfaisable (insatisfaisable) si, pour une (aucune) interprétation J, toutes les formules de Γ sont vraies simultanément.

Une f.b.f. est logiquement valide si et seulement si sa négation est insatisfaisable.

Dans l'approche sémantique, on s'intéresse à la validité logique des formules du système. Le problème fondamental est le suivant.

Etant donnée une f.b.f. quelconque A, peut-on affirmer ou nier la validité logique de A ?

Ce qui s'écrit : $\models A$ ou $\not\models A$?

Une f.b.f. découle logiquement d'une f.b.f. (ou A implique logiquement B) noté $A \models B$, si et seulement si toute interprétation satisfaisant A satisfait aussi B.

A et B sont logiquement équivalentes si l'on a $A \models B$ et $B \models A$.

Plus généralement, étant donné un ensemble de f.b.f. Γ, B est une conséquence logique de Γ, notée $\Gamma \models B$, si et seulement si toute interprétation satisfaisant toutes les formules de Γ, satisfait aussi B.

THEOREME I.1

Etant donnés un ensemble Γ de f.b.f. et une conséquence logique de B de Γ, si $\Gamma \vee \{B\}$ est insatisfaisable, alors Γ est insatisfaisable.

En effet, toute interprétation qui satisferait Γ satisferait B donc $\Gamma \vee \{B\}$.

II.2 - Cohérence d'un système logique

D'une manière générale, on dit qu'un système déductif a la propriété de cohérence si, au plus certaines formules du système sont des théorèmes (par exemple, seulement celles qui vérifient une certaine propriété $\mathcal{P}$). Dans le cas du calcul des prédicats, $\mathcal{P}$ sera : "être logiquement valide".

Un système est simplement cohérent si il n'existe pas de f.b.f. A telle que l'on ait à la fois $\vdash A$ et $\vdash \neg A$.

Un système est ω-cohérent (GÖDEL 1931) si on n'a pas à la fois

$\vdash A(a_0)$, $\vdash A(a_1)$, $\vdash A(a_2)$,....., et $\vdash \neg (\forall x) A(x)$

a_0, a_1, a_2,... étant une énumération de l'ensemble des valeurs possibles de x. Tout système ω-cohérent est simplement cohérent.

Remarques -

. On emploie aussi le terme consistance (peut-être à cause de l'anglais consistent) à la place de cohérence.

. On dira, par extension, qu'une règle d'inférence est cohérente (par rapport à la validité) si elle infère des conséquences logiquement valides à partir de prémisses logiquement valides.

II.3 - Complétude d'un système logique

Dans un système déductif, cette propriété signifie, d'une manière générale, qu'au moins certaines formules sont des théorèmes. (Par exemple, toutes celles qui vérifient une certaine propriété Q). Dans le cas du calcul des prédicats, nous nous intéresserons à la propriété de validité.

Exemples -

Le calcul des propositions est un système logique complet et cohérent (par rapport à la validité).

L'arithmétique et le calcul des prédicats d'ordre 2 sont incomplets (par rapport à la validité), (GÖDEL, 1931).

Le calcul des prédicats du 1er ordre est complet et cohérent (par rapport à la validité), (GÖDEL, 1930). Il est également simplement cohérent (HILBERT et ACKERMANN, 1928).

Remarques -

. Il existe une définition différente de la notion d'interprétation (HENKIN, 1950), et donc de la validité, par rapport à laquelle le calcul des prédicats d'ordre supérieur à 1 est complet.

. On rencontre aussi (MENDELSON, 1964) cette définition de la complétude:

Un système logique est complet si et seulement si pour toute f.b.f. fermée A, on a

soit $\vdash A$ soit $\vdash \neg A$

Selon cette définition, le calcul des prédicats du ler ordre n'est pas complet.

II.4 - Décidabilité

Un système est décidable si il existe une procédure permettant de déterminer, au bout d'un nombre fini de pas, si une f.b.f. donnée quelconque est, ou n'est pas, un théorème du système.

Dans ce cas l'ensemble des théorèmes du système est récursif; la procédure s'appelle une procédure de décision.

Exemples - Le calcul des propositions est décidable.

Le calcul des prédicats du ler ordre et l'arithmétique sont indécidables (CHURCH, TURING, 1936).

Un système est semi-décidable si il existe une procédure permettant de déterminer si une f.b.f. donnée A est un théorème, cette procédure se terminant au bout d'un nombre fini de pas dans le cas où A est effectivement un théorème, et pouvant ne pas se terminer si A n'est pas un théorème. Une telle procédure s'appelle procédure de preuve ou procédure de semi-décision.

Exemple - Le calcul des prédicats du ler ordre est semi-décidable (HERBRAND (1930) a fourni une procédure de preuve).

Par la suite, nous limiterons notre étude au calcul des prédicats du ler ordre, que nous désignerons par CP1.

III - LE CALCUL DES PREDICATS DU 1er ORDRE

Il constitue un langage, défini par une syntaxe. Les mots de ce langage sont les formules bien formées (f.b.f.).

III.1 - Syntaxe du CP1

On dispose d'un vocabulaire, constitué de plusieurs ensembles de symboles :

- des symboles de ponctuation : ( ) ,
- des symboles connecteurs logiques : $\neg$ $\supset$ .....
- des symboles de fonctions n.aires : f_i^n ($i \geq 1$, $n \geq 0$)
  (les f_i^0 sont appelées constantes. On les notera a, b, c,.....
  les f_i^n seront notées plus simplement f, g, h,.....)
- des symboles de prédicats (pi^n ($i \geq 1$, $n \geq 0$)
  (les p_i^0 sont appelées propositions, les p_i^n seront notées plus simplement P, Q, R, ...)
- des symboles de quantificateurs : $\forall$, $\exists$
- des symboles de variables : u, v , w , x , y, z,.....

A l'aide de ce vocabulaire, on construit récursivement des expressions de la façon suivante :

(i) termes

- les constantes et les variables sont des termes.
- si t_1, t_2.. .., t_n ($n \geq 1$) sont des termes, f_i^n (t_1, t_2,...., t_n) est un terme.

(ii) formules atomiques

- les propositions sont des formules atomiques.

- si $t_1, t_2, \ldots, t_n$ $(n \geq 1)$ sont des termes, alors l'expression $p_i^n (t_1, t_2, \ldots, t_n)$ est une formule atomique.

(iii) formule bien formée (f.b.f.)

- une formule atomique est une f.b.f.
- si A et B sont des f.b.f., $\neg A$ et $A \supset B$ sont des f.b.f.
- On introduit en outre les connectives $\wedge$, $\vee$, $\sim$ qui sont, par définition, des abréviations :

  - $A \wedge B$ est mis pour $\neg (A \supset \neg B)$,
  - $A \vee B$ est mis pour $(\neg A) \supset B$,
  - $A \sim B$ est mis pour $(A \supset B) \wedge (B \supset A)$.

- Si x est une variable et A une f.b.f., $(\forall x)$ A et $(\exists x)$ A sont des f.b.f.

  x est alors une variable liée dans A ; la portée du quantificateur $\forall$ ou $\exists$ est la f.b.f. A. Une variable qui n'est pas liée dans une formule A est libre dans A. Si une formule ne contient pas de variable libre, elle est fermée. Dans la suite, nous utiliserons l'abréviation f.b.f.f. pour dénoter une formule bien formée fermée. Une formule non fermée est dite ouverte.

Un terme t est libre pour x dans A, si et seulement si lorsqu'on remplace une occurrence libre de x dans A par t, aucune variable de t ne devient liée.

La fermeture d'une f.b.f. A, notée $\tilde{A}$, est une opération permettant d'obtenir une f.b.f. B, en préfixant A par des quantificateurs universels, portant sur toutes les variables libres de A. Si A est une formule fermée, $\tilde{A} = A$; donc $\tilde{\tilde{A}} = A$.

Nous qualifierons de constant (ou encore terminal, de base, fermé), toute expression (terme, formule) ou ensemble d'expressions ne contenant aucune variable.

Exemple - f(a) , P (f(a), b) , Q(a) $\vee$ R(b)

Soit A une f.b.f. Toute sous-expression de A qui est une f.b.f. s'appelle une sous-formule de A.

Le niveau ν_A d'imbrication fonctionnel d'un terme ou d'une formule A est défini récursivement de la façon suivante :

- Si A est une constante ou une variable, $\nu_A = 1$
- Sinon, soit n le maximum des niveaux d'imbrication fonctionnel des termes strictement contenus dans A ; si A est un terme, $\nu_A = n+1$, sinon $\nu_A = n$.

Exemple - Soit A = P (f(x, g(b))) Q(g(g(y))) ; $\nu_A = 3$

Le CP1 possède, comme tout système logique, une approche axiomatique et une approche syntaxique. La description de ces approches est l'objet des deux sections suivantes.

III.2 - Définition axiomatique du CP1

Soient A, B, C des f.b.f. On peut définir le CP1 par

(i) Schémas d'axiomes

(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) $(\forall x) A(x) \supset A(t)$, A(x) étant une f.b.f., 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.

(ii) Règles d'inférences

(1) Modus Ponens : B est déduit de A et de $A \supset B$ (MP)

(2) Généralisation : $(\forall x)$ A est déduit de A (GEN)

Si une f.b.f. A est déduite d'un ensemble Γ (d'une ou deux f.b.f.) par une règle d'inférence, A est dite conséquence directe de Γ .

Une preuve dans le CP1 est une suite $A_1, A_2, \ldots, A_n$ de f.b.f. telle que pour tout i, A_i soit un axiome du CP1, ou une conséquence directe d'une ou plusieurs f.b.f. précédant A_i.

Un théorème du CP1 est une f.b.f. A, telle qu'il existe une preuve dont la dernière f.b.f. soit A.

Dans la 2ème approche, le langage défini ci-dessus est utilisé pour exprimer des assertions sur un certain domaine d'intérêt. La relation entre le langage et le domaine est spécifiée par la sémantique du langage.

III.3 - Définition sémantique du CP1 - Le problème P1

Pour donner un sens à une f.b.f., on lui associe un domaine non vide D, de la façon suivante :

- on fait correspondre à chaque constante un élément de D (appelé objet ou individu)
- à chaque fonction f_i^n on associe une fonction n-aire f_i^{*n}
  $f_i^{*n} : D^n \to D$
- à chaque prédicat p_i^n , on associe une relation n-aire p_i^{*n} dans D, c'est-à-dire une partie de D^n.

La spécification d'un domaine et de ces associations constitue une interprétation de la f.b.f. .

Soit L une formule atomique quelconque. Les f.b.f. $L_1 = L$ et $L_2 = \neg L$ sont appelées des littéraux ; nous appellerons valeur absolue de L_1 et de L_2 notée $|L_1|$, $|L_2|$, la formule atomique L ; nous dirons que le signe de L_1 (ou simplement L1) est positif et que celui de L_2 est négatif. Deux littéraux L_1 et L_2 de signe différent et ayant une même valeur absolue sont dits complémentaires ; L_1 est le complément de L_2, et réciproquement.

Une interpretation d'une f.b.f. A peut être représentée par un ensemble M de littéraux tel que :

- Pour tout litteral constant L de A, L ou son complément est dans M,
- M ne contient pas deux littéraux complémentaires.

M s'appelle un modèle de A. Tous les éléments de M ont, par définition la valeur vraie pour l'interprétation considérée de A. M satisfait A si il correspond à une interprétation qui satisfait A .

Exemple - Considérons l'interprétation définie par :

- D = {1, 2}
- a = 1 , b = 2 , f(1) = 2 , f(2) = 1
- P(1,1) = P(1,2) = Vrai , P(2,1) = P(2,2) = Faux

la formule $(\forall x)\ (\exists y)\ P(y,x)$ est vraie pour cette interprétation,

la formule $P(a,f(a)) \wedge P(b,f(b))$ est fausse pour cette interprétation.

Un modèle correspondant à cette interprétation est :

$$M = \{P(a,x)\ ,\ \neg P(b,x)\ ,\ \neg P(f(a),x)\ ,\ P(f(b)\ ,\ x)\ ,$$
$$P(f(f(a))\ ,\ x)\ ,\ \neg P(f(f(b))\ ,\ x),\ldots\}$$

P(a,x) est mis pour {P(a,a) , P(a,b) , P(a,f(a)),...}

Dans la suite nous utiliserons l'approche sémantique du CP1.

Avant de formuler le problème P1, rappelons brièvement quelques propriétés importantes du CP1.

III.4 - Cohérence, complétude et décidabilité dans le CP1

Si l'on définit les propriétés de cohérence et de complétude par rapport à la notion de validité, on peut énoncer pour le CP1 les propriétés suivantes :

THEOREME III.1 (Cohérence)

Tout théorème est une formule valide, c'est-à-dire :

$$\forall A \text{ f.b.f.} \qquad \vdash A \Longrightarrow \models A$$

THEOREME III.2 (Complétude)

Toute formule valide est théorème, c'est-à-dire :

$$\forall A \text{ f.b.f.} \quad \models A \Longrightarrow \vdash A$$

THEOREME III.3 (Décidabilité)

Le problème de la validité dans le CP1 est semi-décidable et non décidable.

Il existe cependant certaines classes de formules du CP1 pour lesquelles le problème de la validité (ou, ce qui revient au même dans le CP1, celui du théorème) est décidable.

Exemple de sous-classes décidables du CP1 (ACKERMANN 1954, NGOA 1967) -

La forme prénexe et la forme clausale d'une formule seront définies en III.6.

- la classe EA sans symbole fonctionnel autre que les constantes.
Une formule appartient à la classe EA si, lorsqu'elle est sous forme prénexe, aucun quantificateur existentiel n'est gouverné par un quantificateur universel.

- les classes EA_1E, EA_2E sans symbole fonctionnel autre que les constantes.
Une formule appartient à la classe EA_1E (resp. EA_2e) si, lorsqu'elle est sous forme prénexe, son préfixe est une suite finie de quantificateurs existentiels suivie d'un (resp. de deux) quantificateur(s) universel(s), suivi(s) de quantificateurs existentiels.

- la classe conjonctive.
Une formule appartient à cette classe, si et seulement si les clauses de sa forme clausale sont toutes unitaires.

THEOREME III.4

Une f.b.f. est logiquement valide si et seulement si sa fermeture l'est.

Démonstration en utilisant les théorèmes III.1, III.2, le schéma d'axiome (4) MP et GEN.

Nous allons définir maintenant le problème P1 qui représente la classe de problèmes à laquelle nous nous intéresserons.

Problème P1

Etant donnés un ensemble Γ de f.b.f. du CP1 et une f.b.f. A du CP1, démontrer que A est une conséquence logique de Γ , i.e. montrer que $\Gamma \models A$.

Par abus de langage, nous dirons que A est le théorème à démontrer et que Γ est l'ensemble des axiomes du problème. (Ne pas confondre avec les théorèmes et axiomes définis précédemment en III.2). De plus, il est parfois utile de distinguer dans Γ deux types d'axiomes. Les axiomes spécifiques, et les axiomes logiques (ou de base) du problème. Le théorème de la déduction va nous permettre de transformer P1 en un problème équivalent.

III.5 - Le théorème de la déduction - Le problème P2

THEOREME III.5

$A_1, A_2, \ldots, A_m$, B étant des f.b.f., $m \geq 1$, on a

$A_1, A_2, \ldots, A_m \models B$ si et seulement si

$\models (A_1 \wedge A_2 \wedge \ldots \wedge A_m) \supset B$

La démonstration de ce théorème peut être trouvée, par exemple, dans KLEENE (1967).

COROLLAIRE III.6

Pour $m \geq 1$ $A_1, A_2, \ldots, A_m \models B$ si et seulement si

$(A_1 \wedge A_2 \wedge \ldots \wedge A_m \wedge \neg B)$ est insatisfaisable.

DEMONSTRATION

D'après le théorème III.5, B est conséquence logique de $A_1, A_2, \ldots, A_m$ si et seulement si $\neg [(A_1 \wedge A_2 \wedge \ldots \wedge A_m) \supset B]$ est insatisfaisable.

Or $\neg [(A_1 \wedge A_2 \wedge \ldots \wedge A_m) \supset B] = (A_1 \wedge A_2 \wedge \ldots \wedge A_m \wedge \neg B)$ c.q.f.d.

Le théorème III.4 permet de ne considérer que des f.b.f.f. ; ce que nous ferons dans la suite. En utilisant, en outre, le corollaire III.6, on peut transformer P1 en un problème équivalent P2.

Problème P2

Démontrer qu'une f.b.f.f. est insatisfaisable.

III.6 - La forme clausale - Le problème P3 -

Une f.b.f.f. est sous forme prénexe si elle est de la forme

$(Q_1 x_1)\ (Q_2 x_2) \ldots (Q_n x_n)\ M$

où M est une formule sans quantificateur, et les Q_i représentent des quantificateurs universels ou existentiels. M s'appelle la matrice ; l'ensemble des $(Q_i x_i)$ constitue le préfixe.

Une formule A est sous forme normale conjonctive si A s'écrit

$$A = C_1 \wedge C_2 \wedge \cdots \wedge C_k \qquad k \geq 0$$

et si tous les C_i sont de la forme

$$C_i = L_1 \vee L_2 \vee \cdots \vee L_{P_i} \quad , \qquad P_i \geq 1$$

les L_j étant des littéraux. Chaque disjonction C_i de littéraux constitue une clause ; la matrice est donc une conjonction de clauses.

Une formule est sous forme clausale si elle est sous forme prénexe, normale conjonctive, sans quantificateur existentiel.

A toute formule sous forme clausale, on peut faire correspondre l'ensemble de ses clauses en faisant abstraction des symboles $\wedge$ qui y figurent ; inversement, à tout ensemble S de clauses, on peut faire correspondre une formule sous forme clausale, en ordonnant arbitrairement les clauses de S et en les connectant par des symboles $\wedge$.

Le théorème qui suit rend légitime le passage d'une formule sous forme clausale $\mathcal{F}$ à un ensemble de clauses, lorsque l'on s'intéresse à l'insatisfaisabilité de $\mathcal{F}$.

THEOREME III.7

$C_1, C_2, \ldots, C_n$ étant des f.b.f., pour toute permutation $(i_1, i_2, \ldots, i_n)$ de l'ensemble d'indices $\{1, 2, \ldots, n\}$, on a

$$\models C_1 \wedge C_2 \wedge \cdots \wedge C_n \sim C_{i_1} \wedge C_{i_2} \wedge \cdots \wedge C_{i_n}$$

la démonstration peut se faire par récurrence sur n, en utilisant la propriété vraie pour n = 2 (cf par exemple KLEENE 1971).

COROLLAIRE III.8

Une formule sous forme clausale est insatisfaisable si et seulement si l'ensemble de clauses qui lui correspond l'est aussi.

Rappelons (cf. II.1) qu'un ensemble de clauses est insatisfaisable si il n'existe aucune interprétation pour laquelle toutes les clauses soient vraies simultanément.

Une clause positive (respectivement négative) est une clause qui ne contient que des littéraux positifs (respectivement négatifs).

THEOREME III.9

Tout ensemble de clauses insatisfaisable contient au moins une clause positive et une clause négative.

DEMONSTRATION

Un ensemble de clauses qui ne contient aucune clause positive (respectivement négative) est satisfait par un modèle constitué uniquement de littéraux négatifs (respectivement positifs).

THEOREME III.10

Si un ensemble S contient une clause C logiquement valide on a

$$S \text{ insatisfaisable} \iff S - \{C\} \text{ insatisfaisable.}$$

DEMONSTRATION

$\Leftarrow$

Si S est satisfaisable, S - {C} l'est aussi par définition

$\Rightarrow$

Soit J une interprétation qui satisfait S - {C} ; J satisfait C, donc S.

Deux clauses identiques, au nom de leurs variables près, sont dites variantes l'une de l'autre. La relation "être une variante de" est une relation d'équivalence entre les clauses. Sauf spécification contraire (nous parlerons alors d'occurrence de clause), toute clause dans la suite représentera en fait sa classe d'équivalence par rapport à cette relation.

Un théorème analogue au théorème III.7, mais concernant le symbole $\vee$, permet de considérer, le cas échéant, une clause C comme l'ensemble de ses littéraux noté C^*. Nous appellerons longueur d'une clause C, notée $\|C\|$, le nombre d'éléments de C^*.

Une clause unitaire (ou singleton) est une clause de longueur 1 ; une clause vide, notée $\square$, est une clause de longueur nulle. Nous verrons en IV.7, dans quelles conditions une telle clause peut figurer dans un ensemble de clauses S ($\square$ reflètera une contradiction ; par exemple, $\square$ serait engendrée si S contient les 2 clauses unitaires P(x) et $\neg$ P(x)). Bien entendu, un ensemble de clauses correspondant à la mise sous forme clausale d'une formule quelconque A, ne contient pas de clause vide.

Le théorème qui suit, ainsi que le corollaire III.8, vont nous permettre de transformer le problème P2 en un problème P3 équivalent.

THEOREME III.11

Il existe un algorithme permettant de transformer une f.b.f.f. quelconque A en une f.b.f.f. A' sous forme clausale logiquement équivalente, i.e. telle que

$$\models A \sim A'$$

Une démonstration de ce théorème peut être trouvée dans MENDELSON (1964).

Avant de formuler le problème P3, nous allons décrire un algorithme de transformation d'une f.b.f.f. en un ensemble de clauses.

III.6.1 - Algorithme de mise sous forme de clauses (NILSSON, 1971)

Nous illustrerons l'algorithme sur un exemple en considérant la f.b.f.f. suivante

$$(\phi) \qquad (\exists x)\{(\forall y)[(\exists z)\, P(x,y,z) \supset Q(x)]\} \sim (\forall x)\, Q(x)$$

On effectue la succession de pas suivants :

pas 1 : élimination de signes d'équivalence

On remplace toute sous formule de la forme $B \sim C$ par

$$(\neg B \vee C) \wedge (\neg C \vee B)$$

(ϕ) devient

$$(\phi_1) \quad (\neg (\exists x) \{(\forall y) [(\exists z) P(x,y,z) \supset Q(x)]\} \vee (\forall x) Q(x)) \wedge$$
$$(\neg (\forall x) Q(x) \vee (\exists x) \{(y) [(\exists z) P(x,y,z) \supset Q(x)]\})$$

pas 2 : élimination des signes d'implication

On remplace toute sous formule de la forme $B \supset C$ par $\neg B \vee C$

(ϕ_1) devient

$$(\phi_2) \quad (\neg (\exists x) \{(\forall y) [\neg (\exists z) P(x,y,z) \vee Q(x)]\} \vee (\forall x) Q(x)) \wedge$$
$$(\neg (\forall x) Q(x) \vee (\exists x) \{(\forall y) [\neg (\exists z) P(x,y,z) \vee Q(x)]\})$$

pas 3 : réduction de la portée des signes de négation

On remplace toute sous formule d'une formule contenue dans la colonne de gauche par celle de la colonne de droite correspondante :

$\neg (A \wedge B)$	est remplacée par	$\neg A \vee \neg B$
$\neg (A \vee B)$	"	$\neg A \wedge \neg B$
$\neg \neg A$	"	A
$\neg (\forall x) A$	"	$(\exists x) \neg A$
$\neg (\exists x) A$	"	$(\forall x) \neg A$

à l'issue de ce pas, il ne doit figurer, dans la portée d'un signe de négation, qu'une formule atomique.

(ϕ_2) devient :

$$(\phi_3) \quad ((\forall x) \{(\exists y) [(\exists z) P(x,y,z) \wedge \neg Q(x)]\} \vee (\forall x) Q(x)) \wedge$$
$$((\exists x) \neg Q(x) \vee (\exists x) \{(\forall y) [(\forall z) \neg P(x,y,z) \vee Q(x)]\})$$

pas 4 : standardisation des variables

Pour tout quantificateur Q, on crée une nouvelle variable x_i et on remplace, dans la portée de Q, toute occurrence de la variable quantifiée par Q, par x_i

(ϕ_3) devient

$$(\phi_4) \quad ((\forall x_1) \{(\exists x_2) [(\exists x_3) P(x_1,x_2,x_3) \wedge \neg Q(x_1)]\} \vee (\forall x_4) Q(x_4)) \wedge$$
$$((\exists x_5) \neg Q(x_5) \vee (\exists x_6) \{(\forall x_7) [(\forall x_8) \neg P(x_6,x_7,x_8) \vee Q(x_6)]\})$$

pas 5 : élimination des quantificateurs existentiels (ou Skolemisation)

La formule $A = (\forall x) (\exists y) P(x,y)$ affirme l'existence d'un y pour un x donné, tel que le couple (x,y) vérifie P. Ceci définit une correspondance implicite : $x \to y$. Explicitons cette correspondance en la nommant $x \xrightarrow{f} y$; f est un nouveau symbole et est appelé fonction de Skolem ; la formule $(\forall x) P(x, f(x))$ affirme la même chose que A et lui est donc équivalente.

D'une manière générale, on pourra supprimer tout quantificateur existentiel portant sur une variable x_i, à condition de remplacer chaque occurrence de x_i par $f_i^p (x_{i_1}, x_{i_2}, \ldots, x_{i_p})$, si x_i est dans la portée de p quantificateurs universels portant sur les variables $x_{i_1}, x_{i_2}, \ldots, x_{i_p}$, f_i^p étant un nouveau symbole fonctionnel à p places.

(ϕ_4) devient

$$(\phi_5) \quad ((\forall x_1) [P(x_1, f_1^1(x_1), f_2^1(x_1)) \wedge \neg Q(x_1)] \vee (\forall x_4) Q(x_4)) \wedge$$
$$(\neg Q(a) \vee (\forall x_7) [(\forall x_8) \neg P(b,x_7,x_8) \vee Q(b)])$$

pas 6 : conversion en forme prénexe

Puisqu'il ne reste plus que des quantificateurs universels et que ceux-ci portent sur des variables différentes, on peut accroître la portée de chacun d'eux à l'ensemble de la formule, en les repoussant à gauche.

(ϕ_5) devient

$$(\phi_6) \quad (\forall x_1)\ (\forall x_4)\ (\forall x_7)\ (\forall x_8)\ \{([P(x_1, f_1^1(x_1), f_2^1(x_1)) \wedge \neg Q(x_1)] \vee Q(x_4)) \wedge (\neg Q(a) \vee [\neg P(b,x_7x_8) \vee Q(b)])\}$$

pas 7 : conversion de la matrice sous forme normale conjonctive

On remplace, dans la matrice, toute sous formule de la forme

$$A \vee (B \wedge C) \quad \text{par} \quad (A \vee B) \wedge (A \vee C)$$

et toute sous formule de forme

$$(A \wedge B) \vee C \quad \text{par} \quad (A \vee C) \wedge (B \vee C)$$

ϕ_6 devient, après suppression des parenthèses inutiles,

$$(\phi_7) \quad (\forall x_1)\ (\forall x_4)\ (\forall x_7)\ (\forall x_8)\ \{[P(x_1,f_1^1(x_1), f_2^1(x_1) \vee Q(x_4)] \wedge [\neg Q(x_1) \vee Q(x_4)] \wedge [\neg Q(a) \vee \neg P(b,x_7,x_8) \vee Q(b)]\}$$

Le passage de la forme clausale à un ensemble de clauses se fait en supprimant les quantificateurs et les symboles $\wedge$.

On obtient alors l'ensemble de 3 clauses suivant :

$$S = \{P(x_1, f_1^1(x_1), f_2^1(x_1)) \vee Q(x_4)\ ,\ \neg Q(x_1) \vee Q(x_4)\ ,\ \neg Q(a) \vee \neg P(b,x_7,x_8) \vee Q(b)\}$$

A partir de maintenant, nous représenterons parfois les clauses, en sous-entendant les symboles $\vee$: P(a) P(b) sera mis pour P(a) $\vee$ P(b).

En utilisant le corollaire III.8 et le théorème III.11, on peut transformer le problème P2 en un problème équivalent.

III.6.2 - Le problème P3

Etant donné un ensemble S de clauses, montrer que S est insatisfaisable.

Les problèmes P1, P2, P3 étant successivement équivalents, nous pourrons donc supposer, sans perte de généralité, les problèmes posés directement sous la forme P3.

Remarques -

- toute variable d'une clause est implicitement quantifiée universellement
- si 2 variables appartiennent à des clauses distinctes, elles sont différentes, même si elles sont désignées par un même nom

  Exemple - dans $S = \{P(x)\ Q(y)\ ,\ Q(x)\}$ les 2 occurrences de x désignent des variables différentes x_1 et x_2.
- un ensemble vide de clauses est logiquement valide
- soit S un ensemble de clauses insatisfaisable ; si $\forall C \in S$, $S - \{C\}$ est satisfaisable, S est minimalement insatisfaisable.

Nous allons, dans le chapitre suivant, donner un aperçu des résultats de HERBRAND, qui servent de fondement théorique à notre approche de la démonstration automatique, et nous décrirons des procédures permettant d'utiliser ces résultats, dans la pratique.

IV - THEOREME ET PROCEDURES DE HERBRAND

Pour démontrer qu'un ensemble de clauses S est insatisfaisable, il faudrait, si l'on s'en tenait à la définition, examiner toutes les interprétations possibles de S et prouver l'insatisfaisabilité de S pour chacune. Cette façon de procéder, envisageable pour le calcul propositionnel (tables de vérité) doit être immédiatement écartée dans le cas du CP1 ; en effet, l'existence de variables parcourant des domaines éventuellement infinis, rend le cas du CP1 beaucoup plus complexe que celui du calcul propositionnel.

Le théorème de Herbrand est un résultat fondamental qui permet de passer d'une recherche théoriquement infinie à une recherche finie, dans le cas où S est effectivement insatisfaisable.

Avant d'énoncer le théorème, donnons quelques définitions. Dans tout ce chapitre, S désigne un ensemble fini de clauses quelconque.

IV.1 - Univers de HERBRAND

On appelle univers de HERBRAND associé à S, l'ensemble H_S de termes constants construit récursivement de la façon suivante :

- Si l'ensemble des constantes (symboles fonctionnels à 0 argument) contenus dans S est vide, on introduit dans H_S une constante arbitraire, a.

- Pour tout symbole fonctionnel f_i^n contenu dans S et pour toute suite $t_1, t_2, \ldots, t_n$ (vide si n = 0), on introduit dans H_S $f_i^n(t_1, t_2, \ldots, t_n)$.

Exemple -

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

$$H_S = \{a, b, f(a) , f(b) , f(f(a)) , f(f(b)) , \ldots\}$$

IV.2 - Composant de substitution

C'est un couple (x,τ), noté $x.\tau$, où x est une variable et τ un terme. x est la variable du composant, τ est le substitut du composant.

IV.3 - Substitution

C'est un ensemble fini (eventuellement vide) de composants de substitutions, dont les variables sont toutes différentes. ε désigne la substitution vide.

IV.4 - Instanciation

Soient une expression ou un ensemble d'expression non constant E_o, une substitution $\theta = \{x_1.\ \tau_1, \ldots, x_k.\ \tau_k\}$.

L'instanciation de E_o par θ , notée $E_o\theta$, s'effectue de la façon suivante :

Pas 1 : $E_1 \leftarrow E_o$; aller au pas 2

Pas 2 : Tant que E_1 contient une occurrence de x_i, $1 \leq i \leq k$, la remplacer dans E_1 par le substitut τ_i ; aller au pas 3

Pas 3 : E_1 est le résultat $E_o\theta$.

Remarque - L'algorithme précédent peut ne pas se terminer.

Exemple : $E_o = x$; $\theta = \{x.f(x)\}$

Nous énoncerons dans la suite du chapitre, quelques propriétés supplémentaires sur les substitutions.

Une expression E' est une instance (ou une déployée) d'une expression E, s'il existe une substitution θ telle que $E' = E\theta$.

Remarques -

. Deux clauses variantes l'une de l'autre (cf. III.6) sont des instances l'une de l'autre.

. On peut définir une relation d'ordre partielle entre les instances d'une expression.

Exemple -

$E(x,y) \prec E(f(z),y) \prec E(f(a),y)$

IV.5 - Saturation - Base de HERBRAND

La saturation de S par un ensemble P de termes, est l'ensemble P(S) contenant toutes les instances des clauses de S obtenues avec des substitutions utilisant des termes de P comme substituts de composant.

Soit S' l'ensemble des formules atomiques contenues dans S ; la saturation de S' par H_S s'appelle la base de HERBRAND (ou l'expansion de HERBRAND) de S, notée B_S. Les éléments de B_S sont appelés atomes de S.

Exemple -

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

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

$H_S = H_{S'} = \{a, b, f(a) , f(a) , f(b) , f(f(a)) , f(f(b)),....\}$

$H_S(S') = B_S = \{P(a) , P(b) , Q(a) , Q(b) , P(f(a)) , P(f(b)) ....\}$

IV.6 - H - Interprétation - Arbre sémantique

On appelle H-interprétation de S, toute interprétation de S qui satisfait aux conditions suivantes :

- le domaine D de l'interprétation est H_S
- la correspondance entre les constantes de S et les éléments de D est l'identité
- à chaque symbole fonctionnel f_i^n de S, on fait correspondre la fonction f_i^{*n} de $D^n \to D$ telle que

$$(h_1,\ldots,h_n) \in D^n \xrightarrow{f_i^{*n}} f_i^n(h_1,\ldots,h_n) \in D$$

Une H-interprétation de S est donc complètement déterminée par l'affectation d'une valeur de vérité à chaque atome de S ; cette opération s'appelle valuation des atomes de S.

Soit I une H-interprétation de S. I sera caractérisée par l'ensemble des atomes, ou négations d'atomes qui ont la valeur vraie pour I.

Si α est la cardinalité de B_S, la cardinalité de l'ensemble $\mathcal{H}_S$ des H-interprétations de S est 2^{α}. $\mathcal{H}_S$ peut être représenté par un arbre binaire appelé arbre sémantique complet de S (voir exemple). I est en correspondance biunivoque avec un chemin Ch_I de l'arbre issu de la racine. Tout littéral constant de S, ou son complément (mais pas les 2) est représenté une (et une seule) fois dans Ch_I. Numérotons les noeuds de Ch_I à partir de la racine. A tout ensemble {0, 1,...., i} correspond une H-interprétation partielle de S $I_i \subset I$; un noeud j tel que I_j falsifie une instance constante d'une clause de S, et tel que I_o, I_1,...., I_{j-1} n'en falsifie aucune, s'appelle un noeud d'échec pour S. Un arbre sémantique est fermé pour S, si pour tout $I \in \mathcal{H}_S$, Ch_I contient un noeud d'échec.

Remarques -

. Les définitions de fermeture et de noeud d'échec sont relatives à un ensemble de clauses, car un arbre sémantique complet peut correspondre à plusieurs ensembles de clauses.

. Il existe plusieurs arbres sémantiques complets correspondant à un ensemble $\mathcal{H}_S$.

Exemple -

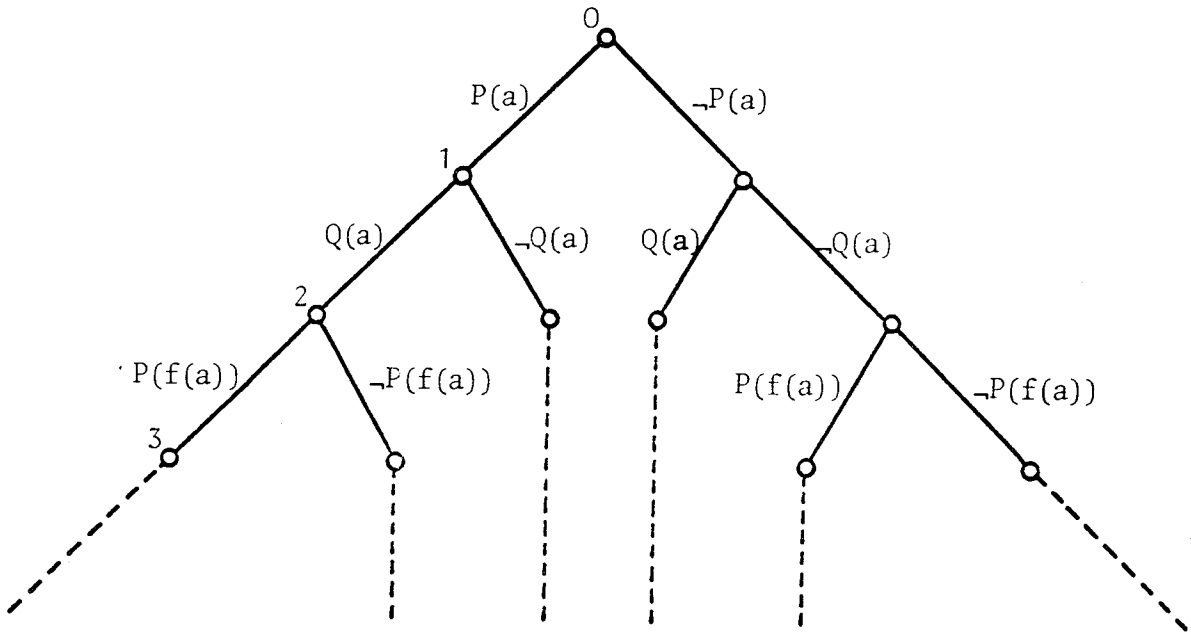


Figure IV.1 : Un arbre sémantique complet de S = {¬P(a), Q(f(a))}

1 est un noeud d'échec pour S.

Les trois résultats qui suivent permettent de ne considérer que les H-interprétations pour démontrer l'insatisfaisabilité de S ; ils sont en rapport avec le théorème de HERBRAND.

LEMME IV.1

A toute interprétation I de S, on peut faire correspondre une H-interprétation.

DEMONSTRATION -

Soient I, une interprétation de S, D le domaine de I. Pour définir une H-interprétation, il suffit de définir une valuation des atomes de B_S, la base de HERBRAND de S. A tout élément h de H_S on peut faire correspondre par I un élément d_h de D (si l'ensemble des symboles constants $\{f_i^o\}$ est vide, H_S contient par construction une constante arbitraire a ; on fait alors correspondre à a un élément quelconque d_a de D).

Soit $P_i^n (h_1, h_2, \ldots, h_n)$ un atome quelconque de B_S ; sa valuation sera celle de $P_i^{*n} (d_{h_1}, d_{h_2}, \ldots, d_{h_n})$ par I.

Dans le cas où $\{f_i^o\} = \emptyset$, il peut donc correspondre plusieurs H-interprétations à une interprétation.

COROLLAIRE IV.2

Si une interprétation I satisfait S, toute H-interprétation I^* correspondante satisfait S.

THEOREME IV.3

S est insatisfaisable si et seulement si S n'est vrai pour aucune H-interprétation de S.

Nous pouvons énoncer maintenant le théorème de HERBRAND sous deux formes équivalentes :

THEOREME de HERBRAND lère version -

Un ensemble de clauses S, fini, est insatisfaisable si et seulement si il possède un arbre sémantique complet et fermé pour S.

La démonstration s'effectue en utilisant le théorème IV.3, le lemme de KONIG sur les arbres (cf. par exemple KNUTH 1968) et les définitions précédentes.

THEOREME de HERBRAND 2ème version -

Un ensemble de clauses S, fini, est insatisfaisable si et seulement si il existe un ensemble fini d'instances constantes de clauses de S insatisfaisable.

Soient S, un ensemble de clauses insatisfaisable, S_1 un ensemble insatisfaisable d'instances constantes de clauses de S ; l'ensemble des termes de S_1 s'appelle ensemble de preuve de S.

La suite du chapitre et les chapitres suivants sont consacrés à l'utilisation pratique du théorème de HERBRAND.

Nous allons décrire des procédures, utilisables sur un ordinateur, pour prouver l'insatisfaisabilité d'un ensemble de clauses donné.

La 2ème version du théorème de HERBRAND suggère directement la procédure suivante :

Construire , de façon exhaustive, des ensembles croissants S_o, S_1, S_2,... d'instances constantes de clauses de S et tester, à l'aide de procédures du calcul propositionnel, chaque ensemble S_i, jusqu'à ce que l'un d'eux soit insatisfaisable. Les premiers démonstrateurs (GILMORE, DAVIS et PUTNAM, PRAWITZ, WANG) étaient basés sur ce principe. Hélas, les ensembles S_o, S_1, S_2 croissent, en général, de façon exponentielle et leur construction ainsi que leur mémorisation (sans parler du test d'insatisfaisabilité !) posent de gros problèmes.

Le principe de résolution (ROBINSON 1965) s'appuie sur la 1ère version du théorème de HERBRAND. Il consiste à utiliser une règle d'inférence, la résolution, qui permet de construire de nouvelles clauses à partir de l'ensemble S de clauses de départ, le test d'insatisfaisabilité de S se réduisant au

test d'appartenance de la clause vide à un nouvel ensemble de clauses ainsi engéndré.

Nous allons voir, sur un exemple, comment le concept d'arbre sémantique peut suggérer l'idée de cette règle d'inférence, puis nous en décrirons le fonctionnement et nous formulerons le problème P4.

IV.7 - Noeud d'inférence d'un arbre sémantique

Soit l'ensemble

$$S = \{P(x) \neg Q(a) = C_1 \ , \ Q(a) \neg R(x) = C_2 \ , \ \neg P(x) = C_3 \ , \ R(x) = C_4\}$$

La figure IV.2 représente un arbre sémantique complet T de S. 3 est un noeud d'échec pour S car $I_3 = \{\neg P(a) \ , \ R(a) \ , \ Q(a)\}$ falsifie une instance constante de C_1 et I_0, I_1, I_2 ne falsifient aucune instance constante de clause de S. De façon analogue, 3' est un noeud d'échec pour S car $I_{3'}$ falsifie une instance constante de C_2. (En fait, T est fermée pour S car 1' et 2' sont aussi des noeuds d'échec pour S).

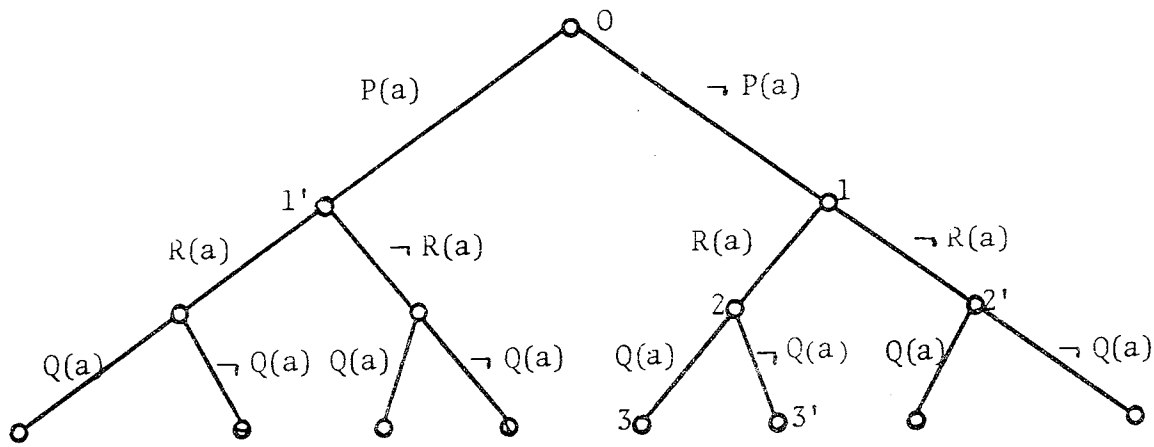


Figure IV.2 - Un arbre sémantique complet

de $S = \{P(x) \vee \neg Q(a) \ , \ Q(a) \vee \neg R(x) \ , \ \neg P(x) \ , \ R(x)\}$

la nouvelle clause $C_5 = P(x) \neg R(x)$, obtenue à partir de C_1 et C_2 est telle que :

(i) $C_1, C_2 \models C_5$

(ii) C_5 falsifie I_2, donc 2 est un noeud d'échec de T pour $S \vee \{C_5\}$

(iii) Si T était fermé pour S, T est fermé pour $S \vee \{C_5\}$

Le noeud 2 est appelé noeud d'inférence de T pour S.

D'après le théorème II.1 et (i), pour démontrer l'insatisfaisabilité de S, il suffit de démontrer celle de $S \vee \{C_5\}$.

De plus, le nombre de noeuds d'échec de T pour $S \vee \{C_5\}$ est inférieur d'une unité au nombre de noeuds d'échec de T pour S ; le noeud 1 est devenu un noeud d'inférence de T pour $S \vee \{C_5\}$; la clause $C_6 = P(x)$, obtenue à partir de C_5 et de C_4, est telle que 1 est un noeud d'échec de T pour $S \vee \{C_5, C6\}$; pour cet ensemble, 0 est devenu un noeud d'inférence ; la clause $C_7 = \square$, obtenue à partir de C_6 et de C_3, est telle que 0 est un noeud d'échec de T pour $S \vee \{C_5, C_6, C_7\}$.

On a : $C_3, C_6 \models \square$; on remarque donc que $\square$ ne peut être satisfaite par aucune interprétation (c'est la mise en évidence d'une contradiction).

THEOREME IV.4

Si un ensemble de clauses S contient la clause vide $\square$, tout arbre sémantique de S possède un unique noeud d'échec pour S : sa racine.

COROLLAIRE IV.5

Un ensemble de clauses contenant $\square$ est insatisfaisable.

En utilisant le théorème de HERBRAND et les résultats précédents, il suffira donc, pour montrer l'insatisfaisabilité d'un ensemble de clauses S, de construire un ensemble S' tel que :

- T fermé pour S' $\Rightarrow$ T fermé pour S
- S' contient $\square$.

Remarque -

D'après (i), la règle d'inférence qui permet d'obtenir C_5 à partir de C_1 et C_2 est cohérente (cf. remarque en II.2).

Nous donnons maintenant une description plus formelle du principe de résolution, précédée de quelques définitions et propriétés complémentaires des substitutions.

IV.8 - Le principe de résolution - Le problème P4

IV.8.1 - Composition de substitutions

Soient 2 substitutions $\theta = \{x_1 . \tau_1, \ldots, x_k . \tau_k\}$

et $\lambda = \{y_1 . \tau'_1, \ldots, y_p . \tau'_p\}$,

Soient $\theta' = \{x_i . \tau_i \lambda / \ 1 \leq i \leq k \ , \ x_i \neq \tau_i \lambda\}$,

$\lambda' = \{y_j . \tau'_j \ / \ y_j \neq x_i \ , \ \forall_{i,j} \quad 1 \leq i \leq k \ , \quad 1 \leq j \leq p\}$

la composition (ou le produit) de θ et λ , notée $\theta\lambda$ est la substitution $\theta' \vee \lambda'$.

Remarques -

Pour toutes substitutions θ , λ , μ et pour toute expression E, on a :

- $\varepsilon\theta = \theta\varepsilon = \theta$
- $(\theta\lambda)\mu = \theta(\lambda\mu)$
- $(E\,\theta)\lambda = E(\theta\lambda)$

IV.8.2 - Substitutions équivalentes

2 substitutions θ_1 et θ_2 sont équivalentes, notées $\theta_1 \equiv \theta_2$, si et seulement si, pour toute expression E, $E\,\theta_1 = E\,\theta_1$

IV.8.3 - Substitution réduite

Une substitution est réduite si elle ne contient pas simultanément 2 composants $x_i.x_j$ et $x_j.\tau_j$, $\forall_i,j$.

THEOREME IV.6

Pour toute substitution θ, il existe une substitution réduite équivalente à θ. Il suffit, pour la construire, de supprimer les composants $x_i.x_j$ et de remplacer $\{x_i.x_j , x_j.\tau_j\}$ par $\{x_i.\tau_j , x_j.\tau_j\}$ $\forall_i,j$.

IV.8.4 - Substitution plus générale

Une substitution λ est plus générale (ou moins instanciée) qu'une substitution θ, notée $\lambda > \theta$, si et seulement si

$\lambda' \subset \theta'$, λ' et θ' étant des substitutions réduites de θ et λ .

Remarques -

. $>$ est une relation d'ordre partielle sur l'ensemble des substitutions

. l'ordre défini en IV.4 sur les expressions aurait pu être défini comme étant l'ordre induit par l'ordre inverse sur les substitutions

. $\forall \lambda$, θ telles que $\lambda > \theta, \exists \mu$ telle que $\lambda \mu \equiv \theta$

IV.8.5 - Unificateur d'un ensemble d'expressions

Soient E un ensemble d'expressions, θ une substitution. Si $E\theta$ ne contient qu'une expression, on dit que θ est un unificateur de E ou que θ unifie E et que E est unifiable (par θ).

Remarque -

De façon générale, toute substitution θ induit une partition K de E en classes d'équivalence, la relation d'équivalence $\mathcal{P}$ étant définie par :

$$\forall E_1 , E_2 \qquad E_1 \mathcal{P} E_2 \Leftrightarrow E_1\theta = E_2\theta$$

Si θ est un unificateur de E, K n'a qu'un élément : E.

IV.8.6 - Plus grand unificateur (p.g.u.)

Si λ unifie un ensemble E et si tout unificateur θ de E est tel que $\lambda \quad \theta$, λ est le plus grand unificateur (p.g.u.) de E (ou encore, l'unificateur le plus général de E).

THEOREME IV.7

Tout ensemble unifiable possède un p.g.u. Une démonstration et un algorithme de construction d'un éventuel p.g.u. sont dans ROBINSON (1965).

IV.8.7 - Substitutions compatibles

Deux substitutions θ_1 et θ_2 sont compatibles si et seulement si pour toute expression E, $E\theta_1$ et $E\theta_2$ sont unifiables ; pour cela, il suffit que τ_i et τ_j soient unifiables pour tous composants $x_i.\tau_i \quad \theta_1'$ et $x_j.\tau_j \quad \theta_2'$ tels que $x_i = x_j$, θ_1' et θ_2' étant des substitutions réduites de θ_1 et θ_2. La substitution notée $\theta_1 * \theta_2$, contenant tous les composants de θ_1' et θ_2' dont les variables sont différentes, ainsi que les p.g.u. des substituts des composants de θ_1' et θ_2' dont les variables sont les mêmes, est appelée p.g.u. de θ_1 et θ_2.

Exemples -

- $\theta_1 = \{x.a , y.b\}$ et $\theta_2 = \{x.c , z.d\}$ sont incompatibles
- $\theta_1 = \{x.f(y)\}$ et $\theta_2 = \{x.f(a) , u.b\}$ sont compatibles ;

  $\theta_1 * \theta_2 = \{y.a , u.b\}$

IV.8.8 - Littéraux potentiellement complémentaires ou identiques

Considérons les 3 littéraux suivants :

$L_1 = P(t_1, t_2,\ldots,t_n) \qquad L_2 = P(t_1', t_2',\ldots, t_n') , \qquad L_3 = \neg P(t_1', t_2',\ldots, t_n')$

Si $E = \{L_1, L_2\}$ est unifiable (par un p.g.u.θ), L_1 et L_2 sont dits potentiellement identiques (mod. θ), L_1 et L_3 sont dits potentiellement complémentaires (mod. θ).

Une clause contenant deux littéraux (potentiellement) complémentaires est une tautologie (potentielle).

THEOREME IV.8

Soit S un ensemble de clauses contenant une tautologie C (non potentielle) ; on a

$$S \text{ insatisfaisable} \Leftrightarrow S - \{C\} \text{ insatisfaisable.}$$

Démonstration en utilisant le théorème III.10 et le fait qu'une tautologie est logiquement valide (car elle est satisfaite par toute interprétation).

IV.8.9 - Sous-clause d'une clause

Soit $C = L_1 \vee \ldots \vee L_k$ une clause quelconque.

Si $L_{i_1}, \ldots L_{i_p}$ représentent des occurrences de littéraux de C, $1 \leq i_1 < i_2 \ldots < i_p \leq k$

$C - \{L_{i_1}, \ldots, L_{i_p}\}$ représente la sous-clause de C, obtenue en supprimant dans C les occurrences des littéraux $L_{i_1}, \ldots, L_{i_p}$.

IV.8.10 - Facteur d'une clause, factorisation d'un ensemble

Soit $C = L_1 \vee \ldots \vee L_k$ une clause telle que

$\forall i_1, i_2 \quad 1 \leq i_1 < i_2 \ldots < i_p \leq k$, $P \geq 1$, $\{L_{i_1}, \ldots, L_{i_p}\}$ est unifiable par θ.

La sous-clause $C\theta - \{L_{i_2}\theta, \ldots, L_{i_p}\theta\}$ de C est un facteur (ou est obtenue par factorisation) de C (vers $L_{i_1}\theta$).

Cas particuliers

. p = 2 : $C\theta - \{L_{i_2}\theta\}$ est appelé facteur binaire de C (vers $L_{i_1}\theta$)

. p = 1 : la définition implique que C est un facteur de C.

Factoriser un ensemble S de clauses consiste à construire un ensemble de clauses, noté $\mathcal{F}(S)$, fermé pour l'opération de factorisation. $\mathcal{F}(S)$ est dit factorisé. Donc, si une clause C appartient à un ensemble factorisé S, tout facteur de C appartient à S. Le lemme suivant découle des propriétés sur les substitutions.

LEMME IV.9

Tout facteur d'une clause peut être obtenue par une suite de factorisations binaires.

IV.8.11 - Résolvant de 2 clauses

Considérons deux clauses $C_1 = L_1 \vee L_2 \vee \cdots \vee L_k$

et $C_2 = L'_1 \vee L'_2 \vee \cdots \vee L'_n$

contenant respectivement deux occurrences L et L' de littéraux potentiellement complémentaires (mod. θ).

On appelle résolvant binaire de C_1 et C_2 (sur L et L' avec θ) , la clause C de la forme suivante

$$C = (C_1 - \{L\})\theta \vee (C_2 - \{L'\})\theta$$

L et L' s'appellent les littéraux résolus ; $R(C_1, C_2)$ désigne l'ensemble des résolvants binaires de C_1 et C_2.

Plus généralement, on appelle résolvant de C_1 et C_2, tout résolvant binaire sur $L\theta_1$ et $L'\theta_2$ (avec λ) de 2 clauses obtenues respectivement par factorisation de C_1 vers $L\theta_1$ et de C_2 vers $L'\theta_2$.

Pour tout couple de clauses (C_1, C_2), il existe un nombre fini (éventuellement nul) de résolvants.

Remarques -

. La définition que nous donnons ici du résolvant diffère de la définition originale de ROBINSON (1965), mais elle lui est équivalente (voir, par exemple, KOWALSKI 1970 b, pour une justification). Dans la suite, nous utiliserons les mots résolvant, résolution, en sous-entendant l'adjectif binaire).

. La transitivité de l'implication et le Modus Ponens sont des cas particuliers d'utilisation de résolvant ; en effet

$A \supset B$, $B \supset C \vdash A \supset C$ correspond à $\neg A \vee C \in R\ (\neg A \vee B, \neg B \vee C)$

$A \supset B$, $A \vdash B$ correspond à $B \in R\ (\neg A \vee B\ ,\ A)$

. En faisant abstraction du processus d'unification, l'opération de calcul d'un résolvant correspond à l'opération de consensus en algèbre de boole.

IV.8.12 - Résolution d'un ensemble de clauses

Soit S, un ensemble de clauses. La résolution de S est un ensemble de clauses, noté $\mathcal{R}(S)$ obtenu en rajoutant à l'ensemble $\mathcal{F}(S)$ des facteurs de S, tous les résolvants possibles de ces facteurs.

La résolution de niveau n de S est l'ensemble $\mathcal{R}^n(S)$ défini par les équations suivantes :

$$\mathcal{R}^0(S) = S$$
$$\mathcal{R}^{n+1}(S) = \mathcal{R}(\mathcal{R}^n(S)) \quad , \quad n \geq 0$$

Une clause C est de niveau (ou de profondeur) i , $i \neq 0$, si

$$C \in \mathcal{R}^i(S) \quad \text{et} \quad C \notin \mathcal{R}^{i-1}(S) \ ;$$

les clauses de S sont de niveau 0.

THEOREME DE GENERALISATION (ROBINSON, 1965)

Soient S un ensemble de clauses et P un sous-multiple de l'univers de HERBRAND de S ; on a $\mathcal{R}(P(s)) \subseteq P(\mathcal{R}(S))$.

Rappelons que la notation P(X) dénote la saturation de l'ensemble de clauses X sur P. Ce théorème signifie que tout résolvant de 2 instances de 2 clauses, C_1 et C_2 est une instance d'un résolvant de C_1 et C_2 ; il est connu en anglais sous le nom de "lifting theorem".

THEOREME DE RESOLUTION (ROBINSON, 1965)

Un ensemble S de clauses est insatisfaisable si et seulement si $\mathcal{R}^n(S)$ contient la clause vide $\square$, pour un certain $n > 0$.

COROLLAIRE IV.10

Si un ensemble de clauses est tel que $\mathcal{R}^{n+1}(S) = \mathcal{R}^n(S)$, pour un certain $n \geq 0$, et si $\square \notin \mathcal{R}^n(S)$, alors S est satisfaisable.

L'indécidabilité du CP1 permet d'affirmer que la réciproque de ce corollaire est fausse.

Remarque -

Il existe des résultats similaires à ceux énoncés ci-dessus, dans le cadre de l'algèbre des fonctions booléennes ; LABORDE (1974) poursuivant les travaux de PICHAT (1970) et de TISON, a démontré la convergence d'un "algorithme d'exclusion" pour le calcul des monômes premiers d'une fonction booléenne qui présente des similitudes avec la résolution.

Le théorème de résolution permet de transformer le problème P3 en un nouveau problème équivalent.

Problème P4

Etant donné un ensemble de clauses S, fini, existe-t-il un entier n, $n \geq 0$, tel que : $\square \in \mathcal{R}^n(S)$?

IV.9 - Déduction - Réfutation

Soient S un ensemble de clauses et C une occurrence d'une clause. Une déduction (ou une dérivation) δ de C à partir de S est une suite finie de clauses $C_1, C_2, \ldots, C_n$ vérifiant les conditions suivantes

(i) $\forall_i$, $1 \leq i \leq n$ C_i est soit

(a) dans S

(b) un facteur de C_j , $j < i$

(c) un résolvant de C_j et C_k j , $k < i$

(ii) $C_n = C$

C_1 est la clause initiale de δ ; n est la longueur de la déduction δ.

Soient i et j les nombres de résolutions et factorisations effectuées dans δ ; le couple (i,j) est le format de δ ; i+j est le coût de δ(et de C) ; le niveau de δ est celui de C ; nous appelerons complexité d'une preuve δ une fonction pouvant avoir comme arguments : le format, la longueur, le niveau de δ , le niveau d'imbrication fonctionnel maximum d'une clause de δ , la longueur maximum d'une clause de δ etc ....

Dans les cas (b) et (c) respectivement de la définition ci-dessus C_i et C_j, C_k sontles ancêtres directs (ou parents) de C_i. Un ancêtre de C_i est, soit un ancêtre direct de C_i, soit l'ancêtre direct d'un ancêtre de C_i. Si C_i est un ancêtre (direct) de C_j, C_j est un descendant (direct) de C_i.

Une preuve (ou réfutation) de S est une déduction de $\square$ à partir de S.

On peut faire correspondre à une preuve un arbre de preuve de la façon suivante :

. Les noeuds de l'arbre sont les clauses de la preuve
. □ est la racine de l'arbre
. 2 noeuds sont reliés par un lien si l'un est ancêtre direct de l'autre.

Les clauses de la preuve qui appartiennent à S sont les feuilles de l'arbre.

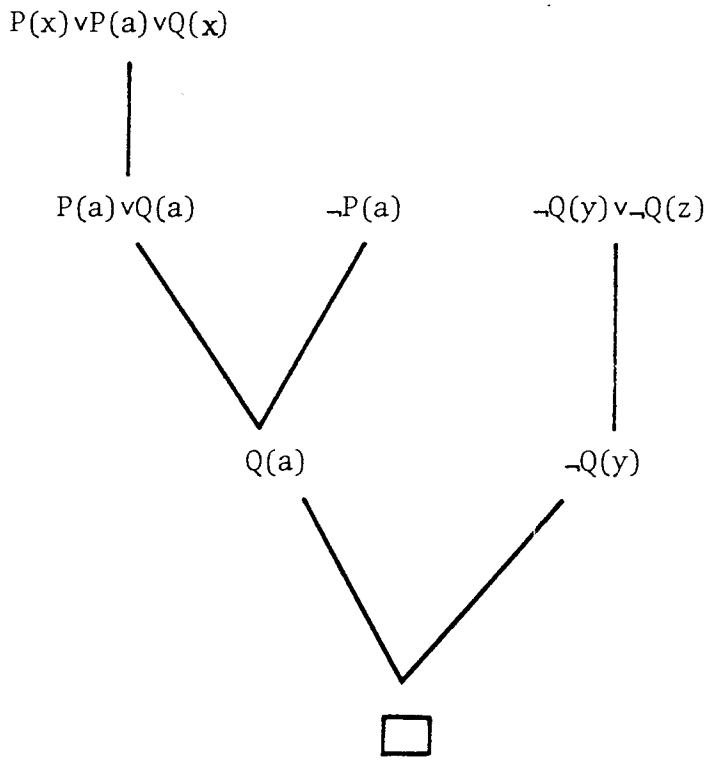


Figure IV.3 : Arbre de preuve de l'ensemble {P(x)vP(a)vQ(x), ¬P(a), ¬Q(y)v¬Q(z)}

Le chapitre suivant est consacré à certaines restrictions qu'il est possible d'imposer lors de l'élaboration d'une réfutation.

DEUXIEME PARTIE

PROCEDURES DE PREUVE

- Méthodes de déduction
- Stratégies de recherche de preuve
- S-L graphes

Nous allons rappeler brièvement la forme du problème auquel nous nous intéressons en définitive.

Soit S un ensemble de clauses (i.e. un ensemble de disjonctions de formules atomiques positives ou négatives) ; les opérations de factorisation et de résolution binaire permettent d'inferrer des clauses à partir de S ; le problème est le suivant : peut-on obtenir la clause vide ($\square$) en appliquant de façon répétitive ces deux opérations à l'ensemble S augmenté au fur et à mesure des clauses engendrées par ces opérations ? Dans l'affirmative, S est insatisfaisable (i.e. quelle que soit l'interprétation I, il existe au moins une clause de S fausse pour I).

Exemple :

$S = \{C_1 = P(x)\ Q(x)\ ,\ C_2 = \neg Q(f(z)),\ C_3 = \neg P(f(z))\ R(z),\ C_4 = \neg R(v)\ \neg R(w)\}$

Par résolution sur C_1, C_2 on obtient $C_5 = P(f(z))$

" C_5, C_3 " $C_6 = R(z)$

Par factorisation sur C_4 on obtient $C_7 = \neg R(w)$

Par résolution sur C_6, C_7 on obtient $C_8 = \square$

S est donc insatisfaisable.

V - METHODES DE DEDUCTION

Le principe de résolution représente une amélioration sensible par rapport à une application directe du théorème de HERBRAND ; cependant, utilisé sans restriction, ce principe conduit à la génération d'ensembles de clauses $\mathcal{R}^i(S)$ volumineux. Dans ce chapitre nous examinerons quelques méthodes qui permettent d'améliorer l'efficacité du principe de résolution. Donnons d'abord quelques définitions inspirées de MELTZER (1970) et de KOWALSKI (1970).

V.1 - Définitions

D'une manière générale, nous appelerons méthode de déduction (ou système d'inférence), ou plus brièvement méthode $\mathcal{R}_\rho$, tout ensemble ρ de règles et/ou de principes permettant de restreindre le nombre de résolutions qu'il est possible d'effectuer dans un ensemble quelconque S de clauses. Dans le cas particulier où on impose aucune limitation, on a : $\rho = \emptyset$ et $\mathcal{R}_\emptyset = \mathcal{R}$. On définit les ensembles $\mathcal{R}_\rho^i(S)$ de clauses de la même façon que les ensembles $\mathcal{R}^i(S)$.

L'espace de recherche de S relatif à $\mathcal{R}_\rho$, noté $S^{\mathcal{R}_\rho}$, correspond à l'ensemble des clauses que l'on peut obtenir à partir de S en utilisant $\mathcal{R}_\rho$.

Une méthode de déduction $\mathcal{R}_\rho$ est complète pour S si l'on a :

S insatisfaisable $\Rightarrow_\infty \square \in S^{\mathcal{R}_\rho}$

Remarquons que $S^{\mathcal{R}_\rho} = \bigcup_{i=0}^{\infty} \mathcal{R}_\rho^i$ et que le théorème de résolution du chapitre précédent établit la complétude de $\mathcal{R}_\emptyset$

Etant données deux méthodes de déduction $\mathcal{R}_\rho$ et $\mathcal{R}_{\rho'}$, nous dirons que $\mathcal{R}_\rho$ est plus raffinée que (ou est un raffinement de) $\mathcal{R}_{\rho'}$, si et seulement si : $S^{\mathcal{R}_\rho} \subset S^{\mathcal{R}_{\rho'}}$

$\mathcal{R}$ est la méthode de déduction la moins raffinée.

Toute dérivation constructible à l'aide de $\mathcal{R}_\rho$ est dite admissible par $\mathcal{R}_\rho$.

Une stratégie de recherche Σ_ρ , relative à une méthode de déduction $\mathcal{R}_\rho$, est un algorithme qui construit des dérivations admissibles par $\mathcal{R}_\rho$ (en vue d'obtenir éventuellement une preuve d'un théorème).

Remarque -

Etant donné un ensemble S de clause, toute stratégie Σ_ρ induit un ordre partiel sur l'espace $\mathcal{S}^{\mathcal{R}_\rho}$

Une stratégie de recherche Σ_ρ est complète (ou exhaustive) pour une méthode de déduction $\mathcal{R}_\rho$ si elle engendre toutes les dérivations admissibles par $\mathcal{R}_\rho$. La complétude de Σ_ρ ne dépend pas de celle de $\mathcal{R}_\rho$, mais celle de $\mathcal{R}_\rho$ peut dépendre de celle de Σ_ρ (cf. V.3).

On appelle procédure de preuve $\mathcal{P}_\rho$ un couple $(\mathcal{R}_\rho , \Sigma_\rho)$.

Une procédure de preuve $\mathcal{P}_\rho$ est complète pour S si l'on a :

S insatisfaisable $\Rightarrow \Sigma_\rho$ produit une preuve de S admissible par $\mathcal{R}_\rho$.

Remarquons que la complétude d'une procédure de preuve $\mathcal{P}_\rho$ pour S implique celle de $\mathcal{R}_\rho$, mais n'implique pas celle de Σ_ρ pour $\mathcal{R}_\rho$.

On peut parler de la difficulté qu'éprouve une procédure de preuve $\mathcal{P}_\rho$ pour un ensemble S, comme étant une mesure de "l'effort" produit par $\mathcal{P}_\rho$ dans l'obtention d'une première preuve $\delta_\square$ de S ; cette difficulté peut être évaluée à plusieurs niveaux, selon les critères de calcul retenus. Voici quelques éléments que l'on peut utiliser pour une telle évaluation :

- nombre de clauses construites,
- nombre de tests effectués sur ces clauses (subsomption , tautologie, support, etc... cf. sections suivantes),
- nombre de clauses acceptées après ces tests,
- nombre de résolutions, de factorisations effectuées,
- nombre d'unifications effectuées,

- niveau maximum d'imbrication fonctionnel,
- longueur de la plus grande clause construite,
- profondeur maximum d'une déduction,
- temps et espace-mémoire utilisés.

Le dernier critère a l'inconvénient de dépendre aussi de facteurs étrangers à la procédure de preuve (programmation, langage, système et ordinateurs utilisés).

L'efficacité d'une procédure de preuve $\mathcal{P}_\rho$ pour S est fonction à la fois de la difficulté qu'éprouve $\mathcal{P}_\rho$ pour S et de la complexité (cf. IV.9) de la première preuve de S obtenue par $\mathcal{P}_\rho$.

Les notions de méthode de déduction et de stratégie de recherche se confondent parfois dans la littérature ; en effet, certaines restrictions peuvent être considérées comme appartenant soit à la méthode de déduction, soit à la stratégie. KOWALSKI (1970) suggère d'incorporer dans la méthode de déduction les restrictions de nature logique, en laissant dans la stratégie celles de nature heuristique.

Nous allons énoncer d'abord quelques principes de simplification (qui pourraient constituer, à eux seuls, des méthodes de déduction) puis quelques méthodes fondées sur des restrictions imposées dans le choix des résolvants à effectuer.

Dans les sections suivantes, S désigne un ensemble quelconque fini de clauses.

V.2 - Principe de pureté

Soit C une clause de S. Un littéral L de C est pur dans S, s'il n'existe pas de littéral L' et de clause C' tels que : $L' \in C'$, C et C' se résolvent sur L et L', $C' \neq C$.

THEOREME V.1 (de pureté)

Etant donnée une clause C de S, contenant un littéral pur dans S, S est satisfaisable si et seulement si S - {C} l'est.

La démonstration des théorèmes V.1 et V.2 se trouvent dans ROBINSON (1965).

Le principe de pureté est le suivant :

Dans un ensemble fini de clauses, on peut supprimer toute clause contenant un littéral pur (en vue de montrer qu'il est insatisfaisable).

Le principe d'instanciation est un cas particulier du principe de pureté :

Soit C une clause de S contenant un littéral L potentiellement complémentaire seulement d'un nombre fini d'instances de littéraux $L_1,\ldots, L_k$ respectivement (mod. θ_1),....,(mod. θ_k) ; on peut remplacer dans S, C par les instances $C\theta_1,\ldots,C\theta_k$.

V.3 - Principe de subsomption

Soient C_1 et C_2 2 clauses non vides distinctes ; C_1 subsume C_2 si il existe une substitution θ telle que $(C_1\theta)^* \subseteq C_2^*$.

Rappel - C^* désigne l'ensemble de littéraux correspondant à la clause C.

C_1 subsume strictement C_2, si C_1 subsume C_2 et C_2 ne subsume pas C_1 (i.e. C_1 et C_2 ne sont pas des variantes l'une de l'autre).

THEOREME V.2 (de subsomption)

Soit C_2 une clause de S subsumée par une clause C_1 de S - $\{C_2\}$;

S insatisfaisable $\iff$ S - $\{C_2\}$ insatisfaisable.

Le principe de subsomption est le suivant.

On peut supprimer toute clause C_2 de S subsumée par une clause C_1 de S, $C_1 \neq C_2$ (en vue de montrer que S est insatisfaisable).

Les restrictions suivantes utilisées seules, ou combinées entre elles) conduisent à des formes réduites du principe de subsomption :

(i) On n'utilise que la subsomption stricte

(ii) On ne supprime que les variantes de clauses existantes

(iii) On n'effectue le test C_1 subsume (strictement) C_2 que si :

- C_2 est une clause nouvellement construite (subsomption nouvelle
- C_1 est unitaire (subsomption unitaire, cf. REBOH et al. 1972)
- C_1 est de longueur 2 (subsomption double).

KOWALSKI (1970) a montré que ces principes devaient être utilisés avec précaution, et il a fourni des contre-exemples qui montrent comment une utilisation inconsidérée de ces principes peut retarder, parfois indéfiniment, l'obtention de la clause vide ; il a étudié la complétude (et l'efficacité) d'une procédure de preuve $\mathcal{P}_\rho = (\mathcal{R}_\rho, \Sigma_\rho)$ en fonction de Σ_ρ, en ayant incorporé dans $\mathcal{R}_\rho$, une version du principe de subsomption ; il est donc préférable, dans ce cas, de parler de compatibilité d'un principe de subsomption avec une stratégie (plutôt que de complétude de ce principe).

V.4 - Principe de saturation (COLMERAUER, KOWALSKI)

Soient C_1 une clause de S, L un littéral de C_1. L est saturé dans S si, pour toute clause C_2 de S contenant un littéral L', potentiellement complémentaire (mod. θ) de L, le résolvant de C_1 et C_2 sur L et L' est une variante d'une clause de S.

Remarques

- C_2 peut être, le cas échéant, une variante de C_1 ; dans ce cas on dit que C_1 est auto-résolvante

- un littéral pur dans S est saturé dans S.

Le principe de saturation (cf. GUIZOL 1972) est le suivant :

Soient C_1 une clause de S, L un littéral de C_1, S_2 l'ensemble des clauses de S contenant un littéral L' potentiellement complémentaire de L ; on peut supprimer C_1 de S après y avoir introduit tous les résolvants $R(C_1, C_2)$ sur L et L' (C_2 parcourant S_2), c'est-à-dire après avoir saturé L.

Enoncé tel quel, ce principe n'est pas complet, comme le montre le contre-exemple suivant :

Soit $S = \{Q(x) = C_1 , \neg Q(a) \neg Q(b) = C_2\}$; S est insatisfaisable ; $R(C_1, C_2) = \{ \neg Q(a) , \neg Q(b)\}$; on ne peut obtenir $\square$, à partir de $S - \{C_1\} \vee R(C_1, C_2)$; pour le rendre complet, il suffit d'inclure dans S, au départ, un nombre fini déterminé de variantes de clauses de S. Ce principe rappelle à la fois le principe de pureté et le principe de subsomption.

V.5 - Principes de suppression sémantiques

Ces principes utilisent le théorème III.10, le premier utilise aussi le théorème IV.8.

V.5.1 - Suppression des tautologies

Dans un ensemble fini de clauses, on peut supprimer toute tautologie (en vue de montrer qu'il est insatisfaisable). Dans certains cas particuliers (par exemple, prédicat d'égalité), il est possible de déterminer la valeur de vérité d'un littéral en fonction de critères sémantiques. Ce qui conduit au principe suivant.

V.5.2 - Suppression par évaluation de littéral

Si un littéral L d'une clause C de S s'évalue à vrai, on peut supprimer C de S; s'il s'évalue à faux, on peut supprimer L de C (en vue de montrer que S est insatisfaisable).

Le principe de suppression des tautologies peut être considéré comme un cas particulier du principe précédent.

V.6 - Graphes de classification (KOWALSKI, 1973)

Cette méthode utilise le principe de saturation et le principe de suppression des tautologies et propose, en outre, une représentation partielle de l'espace de recherche correspondant a la méthode. Elle utilise aussi une forme plus restrictive de l'opération de factorisation : la m-factorisation, notée $\mathcal{F}_m$, qui évite de construire des variantes de clauses déjà construites.

Un m-facteur d'une clause C est défini ainsi :

- Si C est un facteur de S, un m-facteur de C est un facteur de C
- Si C_2 est un facteur d'un résolvant C_1, un m-facteur de C_2 est un facteur de C_2 qui ne contient pas de littéral descendant à la fois de 2 littéraux (ou plus) d'un même parent de C_1.

Exemples :

. $S = \{C_1 = \neg P\ Q(a,y,c)\ Q(z,b,c)\ ,\ C_2 = Q(a,b,x)\ P\}$

C_1 a 2 m-facteurs : C_1 et $\neg P\ Q(a,b,c)$

C_2 a 1 m-facteur : C_2

$C_3 = Q(a,y,c)\ Q(z,b,c)\ Q(a,b,x) \in R(C_1, C_2)$ a 3 m-facteurs :

C_3, $Q(a,b,c)\ Q(z,b,c)$, $Q(a,y,c)\ Q(a,b,c)$.

$Q(a,b,x)\ Q(a,b,c)$ n'est pas un m-facteur.

. le résolvant $Q(a,b,c)$ de $\neg Q(a,b,x)$ et $Q(a,y,c)\ Q(a,b,c)$ n'a aucun m-facteur.

La méthode des graphes de classification est la suivante :

(1) Construire l'ensemble $\mathcal{F}_m(S) = \mathcal{F}(S)$ des m-facteurs de S et construire un graphe G dont les noeuds sont les occurrences de littéraux de $\mathcal{F}_m(S)$, et dont les liens sont de deux types :

(1.1) Liens clausaux

Pour toute clause $C = L_1 L_2 \ldots L_k$ de $\mathcal{F}(S)$, on définit un ordre total dans l'ensemble $C^* = \{L_1, L_2, \ldots, L_k\}$ en reliant L_i et L_{i+1} $(1 \leq i < k)$ par un lien clausal orienté de L_i vers L_{i+1}.

(1.2) Liens résolvants

Connecter toute paire de littéraux potentiellement complémentaires (mod. θ), appartenant à des clauses distinctes de $\mathcal{F}(S)$, par un lien résolvant non orienté étiqueté par θ .

(2) Choisir un lien résolvant λ dans G, entre 2 littéraux de 2 clauses C_1 et C_2 ; construire le résolvant correspondant C_λ et l'ensemble $\mathcal{F}_m(\{C_\lambda\})$ des m-facteurs de C_λ, puis supprimer λ dans G ; inclure dans G les occurrences des littéraux de $\mathcal{F}_m(\{C_\lambda\})$ en créant des liens clausaux comme en (1.1).

(3) Soient $L\theta$ une occurrence quelconque d'un littéral d'un m-facteur $C_\lambda\mu$ de C_λ , et L un littéral de C_1 ou C_2, ancêtre de $L\theta$; ajouter les liens résolvants connectant $L\theta$ aux autres noeuds de G de la façon suivante :

(3.1) Si un lien résolvant étiqueté par σ relie L à un littéral P, et si θ et σ sont compatibles (cf. IV.8.7), connecter $L\theta$ et P par un lien résolvant étiqueté par le p.g.u. $\theta * \sigma$ de θ et σ (et de $L\theta$ et P).

(3.2) Si $L\theta$ est potentiellement complémentaire (mod. θ) d'un littéral P de C_1 , C_2 ou d'un m-facteur distinct de $C_\lambda\mu$, connecter $L\theta$ et P par un lien résolvant étiqueté par θ.

(4) Supprimer dans G tous les littéraux d'une clause C avec leurs liens si :

(4.1) C est une tautologie

(4.2) un B-litteral de C ne possède aucun lien résolvant (littéral saturé).

La complétude de cette méthode est un problème ouvert ; nous n'avons cependant trouvé aucun contre-exemple qui infirmerait cette hypothèse.

V.7 - Méthode du support (WOS et al. 1965)

Supposons que l'on connaisse, a priori, un sous-ensemble satisfaisable S' de S.

Puisqu'on cherche à montrer que S est contradictoire, il semble naturel "d'orienter" les résolutions vers l'ensemble de clauses complémentaire de S' par rapport à S, et d'interdire les résolutions entre clauses de S'.

La méthode du support repose sur cette idée. Elle est définie de la façon suivante :

(i) On choisit un sous-ensemble K de clauses de S, tel que S-K soit satisfaisable ; K s'appelle l'ensemble de support de S ; on dit que les clauses de K ont le support.

(ii) Pour effectuer le résolvant C de C_1 et C_2, il faut que C_1 ou C_2 au moins aient le support ; C hérite alors du support.

(iii) Si C_1 est un facteur de C_2, C_1 a le support si et seulement si C_2 l'a.

Cette méthode est complète, mais conduit à des procédures d'efficacité différente selon le choix de K (cf. REBOH et al., 1972).

Il est courant de choisir pour K un sous-ensemble de l'ensemble des clauses provenant de la négation du théorème à démontrer et des axiomes spécifiques du problème.

V.8 - Déduction P1 (ROBINSON)

Cette méthode consiste à n'effectuer le résolvant de deux clauses que si l'une d'elles au moins est positive (cf. III.6).

Cette méthode est complète ; il y a des méthodes dérivées de celle-ci qui utilisent un changement de nom des prédicats (MELTZER, 1966).

Les deux méthodes précédentes peuvent être considérées comme des cas particuliers de la méthode qui suit.

V.9 - Méthode du modèle

Nous avons vu en III.3 qu'un modèle de S est un ensemble M (éventuellement infini) de littéraux tel que, pour toute instance constante L d'un littéral L de S :

$$L\sigma \in M \iff \text{complément de } L\sigma \notin M$$

Si l'ensemble $\mathcal{L}$ de toutes les instances constantes d'un littéral L est inclus dans M, on peut représenter M de façon compacte, en remplaçant $\mathcal{L}$ par L dans M.

Exemple -

$M = \{P(x), \neg Q(a), Q(b)\}$ est mis pour $\{P(a), P(b), \neg Q(a), Q(b)\}$.

La méthode du modèle est la suivante :

Soit M un modèle de S. On n'effectue le résolvant de deux clauses que si l'une des clauses, au moins, n'est pas satisfaite par M. Cette méthode est complète (LUCKHAM, 1968).

Remarques -

- Si M est l'ensemble des littéraux négatifs, on obtient une déduction P1.
- Une preuve de S avec ensemble de support K peut être obtenue en choisissant un modèle qui satisfait S-K, et qui ne satisfait pas les clauses de K.

V.10 - Résolution linéaire (LUCKHAM, 1968)

Soit δ une dérivation d'une clause C dans S.

$$\delta = C_1, C_2, \ldots, C_n$$

δ est linéaire si les conditions suivantes sont vérifiées.

- $C_1 \in S$
- Si C_{i+1} est obtenue par résolution, l'un des parents de C_{i+1} est C_i (parent proche), le 2e parent de C_{i+1}, est soit :

(i) une clause, ou un facteur d'une clause, de S (parent d'entrée),

(ii) un ancêtre de C_i (parent éloigné)

Dans le cas (i) C_{i+1} est obtenue par résolution d'entrée

Dans le cas (ii) C_{i+1} est obtenue par résolution ancestrale.

La méthode qui consiste à ne construire que des dérivations linéaires est complète (LUCKHAM, 1968). Elle est aussi appelée méthode du filtre par les ancêtres. La méthode qui suit est un raffinement de celle-ci. Elle présente beaucoup de ressemblance avec la méthode de Model-élimination (LOVELAND, 1969) dont elle dérive.

V.11 - SL- résolution (KOWALSKI et KUEHNER, 1970)

Dans cette section K désigne un ensemble de support de S.

Soit $\delta = C_1, C_2, \ldots, C_n$ une déduction d'une clause C de S.

C_j est un A-ancêtre de C_i ($j < i$), si et seulement si

- C_{j+1} est obtenue par résolution (d'entrée) de C_j avec une clause de S sur des littéraux L et $\neg$ L
- Il existe une substitution θ, telle que pour tout littéral P de C_j :

$$P \neq L \Rightarrow P\theta \text{ est un littéral de } C_i$$

L est le A-littéral de C_i provenant de C_j.

La S-L résolution est la méthode obtenue en imposant les restrictions suivantes :

- On ne construit que des dérivations linéaires
- les résolutions ancestrales ne sont faites qu'avec des A-ancêtres

et elles sont faites en priorité par rapport aux autres

- les A-littéraux provenant d'A-ancêtres différents ont des valeurs absolues distinctes
- la clause initiale d'une dérivation a le support
- il n'y a pas de tautologie dans une dérivation
- dans toute clause C, on choisit un littéral (le littéral sélectionné) qui est le seul sur lequel on puisse résoudre C lors d'une résolution d'entrée utilisant C comme parent proche.

Nous allons décrire maintenant un formalisme permettant de construire des dérivations qui satisfont à toutes ces restrictions.

Rappelons que $\mathcal{F}(S)$ désigne la fermeture de S par l'opération de factorisation. Tout littéral du système a un statut de A-littéral ou de B-littéral. Nous distinguerons les A-littéraux des B-littéraux en les entourant d'un carré.

Une chaîne est une suite ordonnée C de A- et B-littéraux ; la relation d'ordre entre les littéraux d'une même chaîne sera notée $<$.

Toute sous-suite maximale de B-littéraux de C est un maillon de la chaîne. Deux chaînes qui ne diffèrent que par l'ordre des B-littéraux dans un ou plusieurs maillons sont équivalentes. (P $\boxed{Q}$ RS est équivalente à P $\boxed{Q}$ SR mais n'est pas équivalente à PR $\boxed{Q}$ S). Tous les littéraux de $\mathcal{F}(S)$ sont des B-littéraux. A toute clause C de $\mathcal{F}(S)$, on fait correspondre une chaîne d'entrée C' (qui ne contient qu'un seul maillon) en ordonnant arbitrairement les B-littéraux de C. Le maillon le plus à gauche d'une chaîne C s'appelle le maillon actif de C, noté μ_C. Si le littéral le plus à gauche d'une chaîne n'est pas un B-littéral, son maillon actif est vide.

Exemples :

- la chaîne $C_1 = P(x)\ \boxed{Q(y)}\ \boxed{R(x)}\ P(a)$ possède deux maillons ; le maillon actif μ_{C_1} de C_1 est $P(x)$.

- la chaîne C_2 = $\boxed{P(b)}$ Q(a) possède un maillon ; μ_{C_2} est vide.

Soit ϕ une fonction qui fait correspondre à toute chaîne C, telle que μ_C soit non vide, un B-littéral $\phi(c)$ de μ_C. ϕ est une <u>fonction de sélection</u> ; $\phi(c)$ est le <u>littéral sélectionné</u> dans C par ϕ.

Une chaîne d'entrée a le support, si elle provient d'une clause de $\mathcal{F}(K)$.

<u>Remarque</u> -

Les chaînes sont représentées dans l'ordre inverse de celui présenté habituellement (i.e. μ_C = maillon le plus a droite) par commodité vis à vis de la représentation en liste de LISP (le fait que l'auteur soit gaucher est peut-être une autre raison !).

<u>Une SL-dérivation</u> de S est une suite de chaînes $\delta = C_1, C_2, \ldots, C_n$ qui satisfait aux 3 conditions suivantes,

(1) C_1 est une chaîne d'entrée qui a le support

(2) C_{i+1} est obtenue à partir de C_2 par l'une des 3 opérations décrites ci-après (extension, réduction, tronquation)

(3) Deux littéraux de C_i ont des valeurs absolues différentes (<u>condition d'admissibilité</u>) sauf si C_{i+1} est obtenue par réduction de C_i.

C_{i+1} est obtenue par <u>tronquation</u> de C_i si et seulement si :

1 - Le littéral le plus à gauche de C_i est un A-littéral,

2 - C_{i+1} est la plus grande sous-chaîne de C_i dont le maillon actif soit non vide. Les littéraux de C_{i+1} ont le même statut que celui qu'ils avaient dans C_i.

C_{i+1} est obtenue par <u>réduction</u> de C_i si et seulement si

1 - C_i possède un maillon actif non vide

2 - C_i n'a pas été obtenue par tronquation

3 - Le maillon actif de C_i contient un B-littéral L, et soit

3.1 - un maillon non actif de C_i contient un B-littéral P (m-factorisation), soit

3.2 - C_i contient un A-littéral $\neg P$ qui n'est pas le A-littéral le plus à gauche de C_i (résolution ancestrale).

4 - L et P sont unifiables, avec un plus grand unificateur θ .

5 - $C_{i+1} = C'_i \theta$, où C'_i est obtenue à partir de C_i par suppression de L.

Les littéraux de C_{i+1} ont le même statut que celui qu'ils avaient dans C_i.

C_{i+1} est obtenue par <u>extension</u> de C_i avec une <u>chaîne d'entrée</u> B si et seulement si :

1 - C_i possède un maillon actif non vide

2 - C_i et B n'ont aucune variable commune

3 - Le littéral L sélectionné dans C_i est potentiellement complémentaire (mod. θ) d'un littéral P de B.

4 - Soient B' et C'_i les chaînes obtenues par suppression, respectivement de P dans B et de L dans C_i ; C_{i+1} est la chaîne (B' L C'_i) θ. Le littéral Lθ de C_{i+1} devient un A-littéral ; tous les autres littéraux de C_{i+1} ont le statut qu'ils avaient dans C_i ou B.

<u>Remarque</u> -

Toutes les chaînes d'une SL - dérivation de S ont le support. Cette méthode est <u>fortement complète</u> : ce qui signifie que l'on est assuré qu'une réfutation existe, si toutes les clauses de K appartiennent à un ensemble minimalement insatisfaisable (cf. III.6.2.).

Avant de présenter la méthode des S-L graphes obtenue à partir de la SL -résolution et des graphes de classification, nous décrivons des représen-

tations arborescentes des espaces de recherche correspondant à ces méthodes ainsi qu'une stratégie de parcourt d'arbres, la stratégie diagonale montante ; ce type de stratégie a été étudié par HART, NILSSON et RAPHAEL (1968), puis par POHL (1969) et KOWALSKI (1969, 1972).

VI - STRATEGIES DE RECHERCHE DE PREUVE

VI.1 - Représentation arborescente d'un espace de recherche

Les espaces de recherche que nous considérerons peuvent être représentés par des arbres, que nous appelerons arbres de contrôle.

Les résultats relatifs à la complétude d'une méthode de déduction nous permettent de définir deux types de noeuds dans ces arbres:

- les noeuds de non-retour-arrière (en abrégé "noeuds N.R.A")
- les noeuds de retour-arrière (en abrégé "noeuds R.A.").

Dans un arbre de contrôle, le type d'un noeud correspond au type de décision que doit prendre un algorithme qui parcourt cet arbre en allant de la racine vers une éventuelle solution.

- Lorsqu'il atteint un noeud N.R.A. il lui suffit de choisir un seul descendant pour poursuivre la recherche vers une éventuelle solution.
- Lorsqu'il atteint un noeud R.A. , il peut lui être nécessaire d'examiner, le cas echéant, plus d'un descendant direct de ce noeud car certains de ces descendants peuvent conduire à une solution, tandis que d'autres peuvent ne pas y conduire. On ne peut éliminer un noeud R.A. qu'apres avoir exploré tous ses descendants directs.

Les noeuds R.A. feront donc l'objet, le cas échéant, d'un retour-arrière et nécessitent la mise en oeuvre d'un processus d'empilage. Les branches issues d'un noeud N.R.A. correspondent à des directions de recherche indépendantes dans l'espace de recherche.

VI.2 - Arbre de recherche de la SL-résolution

L'espace de recherche correspondant à la SL-résolution d'un ensemble S de clauses avec un ensemble de support S_0 peut être représenté par un arbre

de contrôle, de la façon suivante :

. la racine R de l'arbre est un noeud N.R.A. et correspond à l'ensemble S_o des chaînes d'entrée qui ont le support.

. Chaque noeud descendant de R correspond à une chaîne de S_o.

. Si une opération d'extension s'applique à une clause C et si il existe plusieurs fonctions de sélection possibles pour C, un noeud N.R.A. correspond à C et chaque descendant de ce noeud correspond au choix d'une certaine fonction de sélection.

. Dans tous les autres cas, une chaîne C correspond à un noeud R.A. ayant pour descendant les noeuds correspondant aux chaînes que l'on peut obtenir en appliquant à C toutes les opérations (applicables) de la SL - résolution.

Les feuilles de l'arbre correspondent, soit à des chaînes inadmissibles, soit à $\square$.

Exemple : Cf. figure VI.1.

VI.3 - Arbre de recherche des graphes de classification.

On peut représenter l'espace de recherche correspondant à la méthode des graphes de classification par un arbre dont les noeuds sont des graphes. Cet arbre est défini ainsi :

. Tous ses noeuds sont des noeuds N.R.A.

. Sa racine est le graphe initial correspondant à $\mathcal{F}(S)$

. A un lien quelconque λ d'un noeud N de l'arbre correspond un descendant de N qui est le graphe obtenu par la transformation de N consécutive à la suppression de λ.

. Une feuille de l'arbre correspond soit à un graphe contenant une clause vide, soit à un graphe vide (i.e. ne contenant aucune clause).

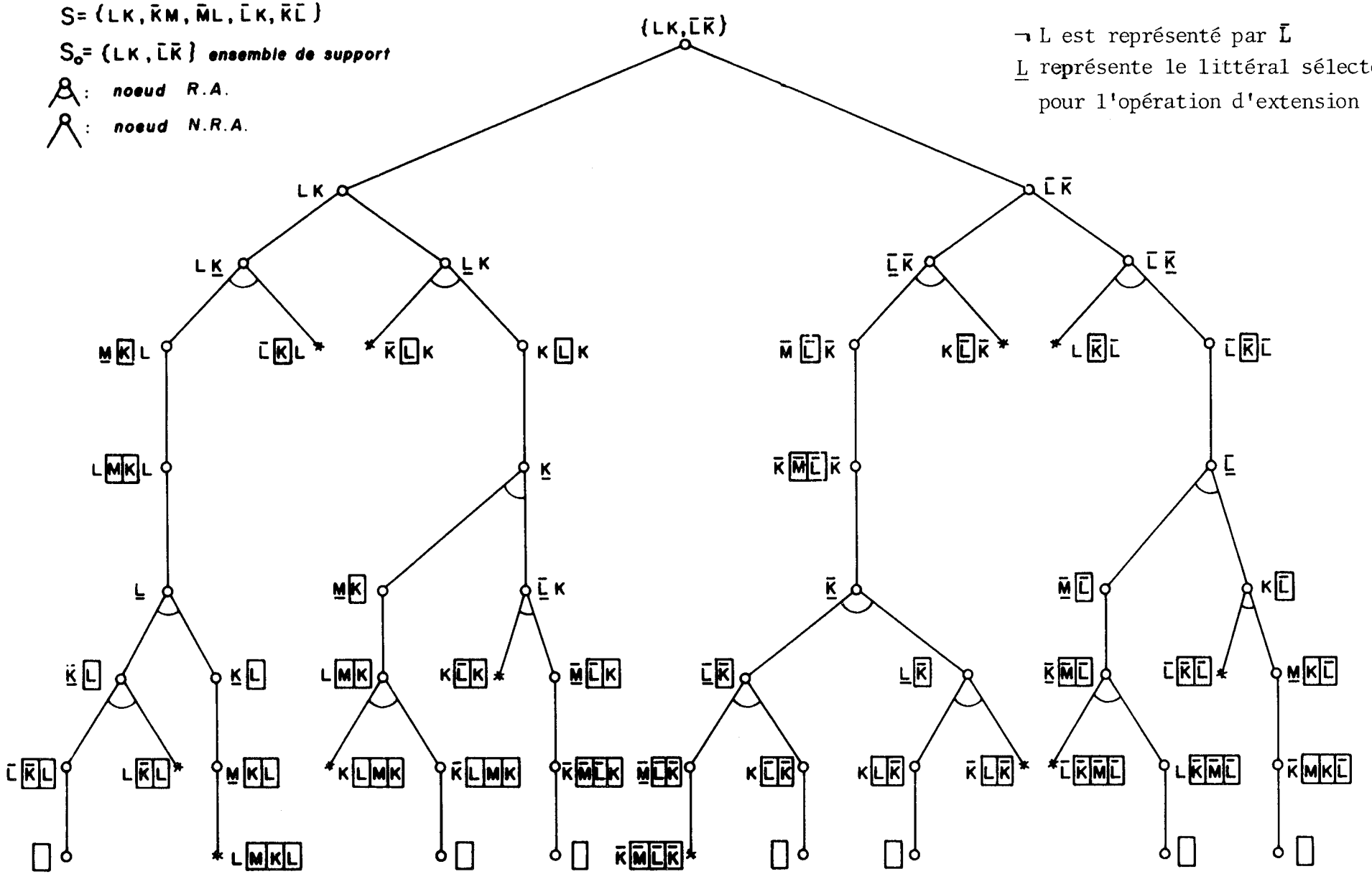


FIGURE VI.1 : Arbre de recherche associé à la SL - résolution de S avec ensemble de support S_o

VI.4 - Stratégie de recherche

Il existe une correspondance biunivoque entre un chemin issu de la racine d'un arbre de recherche, et une dérivation dans S. Une stratégie de recherche (cf. V.1) peut donc être considérée comme un algorithme d'exploration d'un arbre de recherche.

Le principe d'une stratégie est de définir un ordre de mérite sur l'ensemble des dérivations de S et d'engendrer ces dérivations par ordre de mérite décroissant. Si pour toute dérivation D, l'ensemble des dérivations de meilleur mérite que D est fini, on dit que le mérite est δ-fini.

VI.5 - La stratégie diagonale montante

Soit D une dérivation dans S.

Soient g, une fonction de coût de D, $g(D) \geq 0$ et h une fonction heuristique de D , $h(D) \geq 0$. La valeur h(D) est une estimation du coût qu'il faut ajouter à g(D) pour obtenir une preuve contenant D. La fonction $f(D) = g(D) + h(D)$ est donc une estimation du coût d'une preuve contenant D ; f(D) s'appelle fonction d'évaluation de D.

Exemples de fonctions g et h -

- g(D) peut être le niveau de D
- h(D) peut être le nombre de littéraux de la dernière clause C de D, (D est une dérivation de C dans S).

Une stratégie Σ est diagonale montante si elle utilise l'ordre de mérite défini par les relations suivantes :

D a meilleur mérite que D' si et seulement si

- soit $f(D) < f(D')$
- soit $f(D) = f(D')$ et $h(D) < h(D')$

Le couple (h(D), g(D)) est le mérite de D

f est monotone, si pour toute sous-dérivation D' d'une dérivation quelconque D

$$f(D') \leq f(D) \quad , \quad \text{et si} \quad h(D^*) = 0 \quad \text{lorsque } D^* \text{ est une preuve de S.}$$

THEOREME VI.1 (KOWALSKI, 1969)

Une stratégie diagonale montante dont le mérite est δ-fini et dont la fonction d'évaluation est monotone, trouve une solution de coût optimal, lorsqu'il existe une solution.

La dénomination"diagonale montante" provient du fait que si on représente sur un plan une dérivation D par un point de coordonnées (g(D),h(D)), l'axe des h étant orienté de gauche à droite, l'axe des g étant orienté de haut en bas, Σ engendre des dérivations, en parcourant, de manière ascendante, des diagonales de plus en plus éloignées de l'origine.

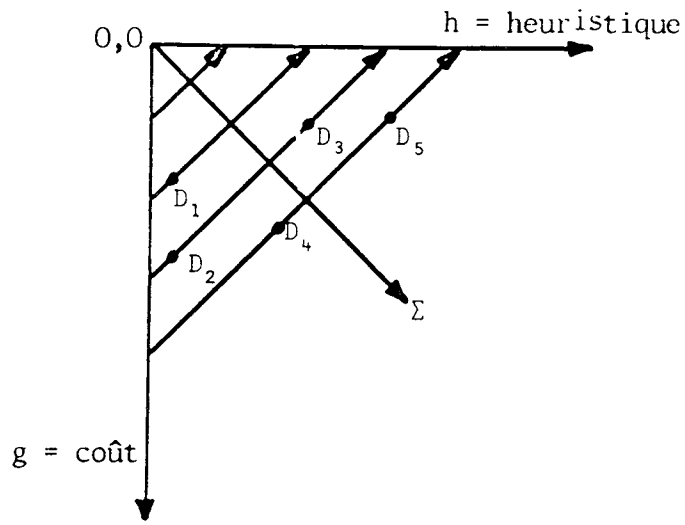


Figure VI.2 : Ordre de génération des dérivations par une stratégie diagonale montante.

Cas particuliers -

(i) La fonction heuristique est constante, i.e. $\forall D \quad h(D) = A$, la stratégie est dite à saturation de coût. En outre,

(a) Si $g(D)$ = niveau de D, la stratégie est dite à saturation de niveau ("Breadth first")

(b) Si $g(D)$ = inverse du niveau de D, la statégie est dite à saturation inverse de niveau ou de la plus profonde descente ("Depth first")

(ii) Soit N_{Max}, une borne supérieure (choisie à l'avance) du niveau des clauses ; supposons que la fonction de coût soit définie par :

. Si niveau (D) $< N_{Max}$, $g(D) = B$ (= constante)

. sinon $g(D) = \infty$

Soit D une dérivation quelconque d'une clause C contenant k littéraux

(a) On a une stratégie à préférence des clauses courtes lorsque $h(D)$ est définie par

$$h(D) = k$$

(b) On a une stratégie à préférence unitaire (WOS et al. 1964) lorsque $h(D)$ est définie par

. Si $k = 1$, $h(D) = 1$

. Sinon $h(D) = \infty$

(iii) POHL (1969) propose une fonction d'évaluation plus générale de la forme

$$f(D) = g(D) + \omega \cdot h(D) \quad , \quad 0 \leq \omega \leq \infty$$

avec la convention suivante : si $\omega = \infty$, $f(D) = h(D)$.

Ce type de fonction permet d'effectuer une pondération de la fonction heuristique selon la confiance qu'on accorde à cette dernière.

Remarque -

La stratégie de la plus profonde descente est intéressante car elle peut être mise en oeuvre simplement à l'aide d'une pile ; malheureusement, il existe des cas dans lesquels on peut ne pas parvenir à la solution : ce sont ceux où l'arbre de recherche possède une branche infinie et où la stratégie explore cette branche.

Nous allons décrire, maintenant, la méthode des S-L graphes.

VII - S-L GRAPHES

Dans tout le chapitre, S désigne un ensemble fini de clauses et K un ensemble de support de S. Nous décrivons d'abord les modifications que nous avons introduites dans le SL - résolution en vue de l'adapter à la méthode des graphes de classification ; nous définirons ensuite la structure de S-L graphes et une méthode de déduction utilisant cette structure ; nous exposerons enfin quelques avantages du système que nous proposons, par rapport à chacune des méthodes dont il provient.

VII. 1 - SL - résolution étendue

Les différences entre cette méthode et la SL- résolution (V.11) résident uniquement dans la définition des opérations de réduction et d'extension que nous avons appelées e-réduction et e-extension.

C_{i+1} est obtenue par e-réduction de C_i si et seulement si

1. C_i possède un maillon actif non vide
2. C_i n'a pas été obtenue par tronquation
3. Le maillon actif de C_i contient un B-littéral L, et soit
   - 3.1 - un maillon non actif de C_i contient un B-littéral P (m-factorisation), soit
   - 3.2 - C_i contient un A-littéral $\neg P$, qui n'est pas le A-littéral le plus à gauche de C_i (résolution ancestrale), soit
   - 3.3 - il existe une chaîne ne contenant qu'un seul B-littéral $\neg P$ (résolution unitaire).
4. L et P sont unifiables avec un p.g.u. θ
5. $C_{i+1} = C_i'\theta$, où C_i' est obtenue à partir de C_i par suppression de L. Les littéraux de C_{i+1} ont le même statut que celui qu'ils avaient dans C_i.

C_{i+1} est obtenue par e-extension de C_i avec une chaîne C si et seulement si

1. C_i possède un maillon actif non vide ;
2. C_i et C n'ont aucune variable commune ;
3. Le littéral L sélectionné dans C_i est potentiellement complémentaire (mod. θ) d'un B-littéral P de C.

Soient C_i' la chaîne obtenue par suppression de L dans C_i,
C' la chaîne obtenue par suppression dans C de P et des A-littéraux de C ;

C_{i+1}' est la chaîne $(C'\ L\ C_i')\ \theta$.

Le littéral $L\theta$ de C_{i+1} devient un A-littéral ; tous les autres littéraux de C_{i+1} ont le statut qu'ils avaient dans C_i ou dans C.

L'espace de recherche de S relatif à cette méthode contient celui relatif à la SL-résolution. Donc cette méthode est complète puisque la SL-résolution l'est.

Nous allons définir, maintenant, une structure de graphe de classification adaptée à la S résolution étendue : le S-L graphe.

VII. 2 - Définition d'un S-L graphe

Soit $T = \{C_1, C_2, \ldots, C_n\}$ un ensemble de chaînes (cf. V.11), avec $C_i = L_{i_1}\ L_{i_2} \ldots . L_{i_k}$, L_{i_j} A- ou B-littéral, $1 \leq i \leq n$, $1 < j \leq k$.
On a, par construction, $L_{i_1} < L_{i_2} < \ldots . < L_{i_k}$; soit $T^* \subseteq T$ un ensemble de chaînes ayant le support.

Un S-L graphe G_T associé à T et T^* est défini ainsi :

1 - Les noeuds de G_T sont les occurrences des littéraux L_{i_j} , $1 \leq i \leq n$, $1 \leq j \leq k$ des chaînes de T .

2 - Il existe 5 types de liens entre les noeuds :

- les liens de chaîne
- les liens résolvants
- les liens facteurs
- les liens ancestraux
- les liens d'admissibilité.

A l'exception des liens de chaînes, les liens sont bi-directionnels et étiquetés par un plus grand unificateur. Entre deux noeuds de G_T, il y a au plus un lien bi-directionnel.

(a) Les liens de chaîne

Ils sont analogues aux liens clausaux d'un graphe de classification ; ils relient deux littéraux consécutifs L_i et L_{i+1} d'une chaîne $L_1\ L_2 \ldots\ L_p$, $1 \leq i < p$ et sont orientés de L_i vers L_{i+1}.

(b) Les liens résolvants

De même que dans les graphes de classification, ils relient toute paire de B-littéraux potentiellement complémentaires (mod. θ), appartenant à des chaînes distinctes de T, par un lien étiqueté par θ. Ces liens seront utilisés pour effectuer des e-extensions ou des e-réductions (en cas de résolution unitaire).

(c) Les liens facteurs

Ils relient dans B-littéraux potentiellement identiques (mod. θ), L_1 et L_2 d'une clause C et sont étiquetés par θ ; L_1 ou L_2 appartient au maillon actif μ_C de C. Ces liens seront utilisés pour effectuer des m-factorisations.

(d) Les liens ancestraux

Ils relient un B-littéral L_1 et un A-littéral L_2 potentiellement complémentaires (mod. θ) d'une chaîne C seulement si

(i) $\exists\, L_3$, $L_1 < L_3 < L_2$, L_3 A-littéral de C

(ii) $L_1 \in \mu_C$

Ils sont étiquetés par θ . Ces liens seront utilisés pour effectuer des résolutions ancestrales.

(e) Les liens d'admissibilité

Ils relient deux littéraux potentiellement complémentaires ou identiques (mod. θ) d'une même chaîne et sont étiquetés par θ. Ces liens serviront à s'assurer que la condition d'admissibilité est respectée (cf. V.11) ; elle l'est si la substitution étiquetant un tel lien n'est pas vide.

Une chaîne du graphe G_T désigne l'ensemble des noeuds de G_T correspondant aux littéraux d'une chaîne de T, ainsi que l'ensemble des liens connectés à ces noeuds.

3 - Une partie de l'ensemble des noeuds de G_T possède un attribut support : les noeuds provenant de T^*. Si un noeud d'une chaîne C de G_T a le support, tous les noeuds de C l'ont aussi ; on dit dans ce cas que C a le support dans G_T.

Exemple -

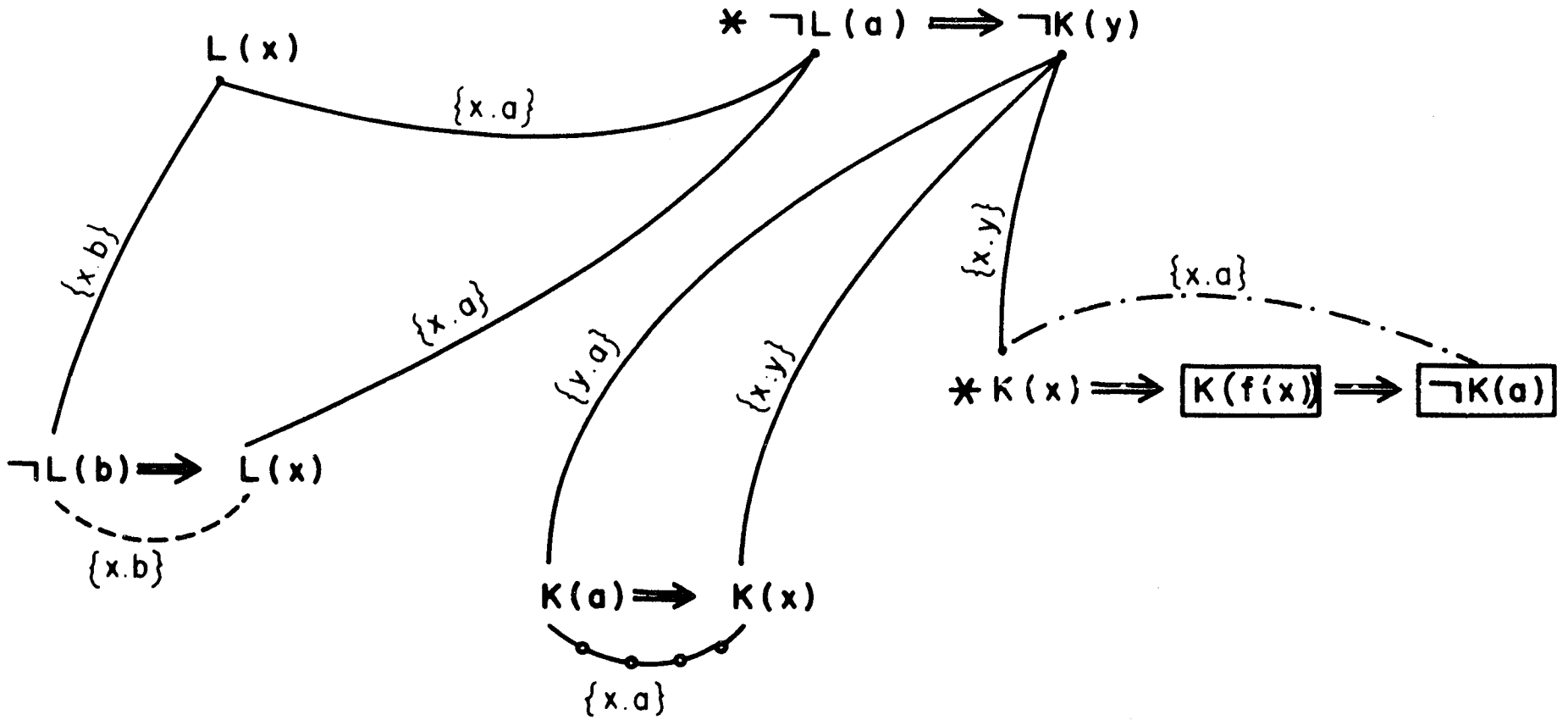
Soient les ensembles de chaînes

$T = \{L(x)\ ,\ \neg L(a)\ \neg K(y)\ ,\ \neg L(b)\ L(x)\ ,\ K(a)\ K(x)\ ,$

$K(x)\ \boxed{K(f(x))}\ \boxed{\neg K(a)}\ \}$, $T^* = \{\neg L(a)\ \neg K(y)\ ,\ K(x)\ \boxed{K(f(x))}\ \boxed{\neg K(a)}$

Les littéraux encadrés sont des A-littéraux.

Le graphe G_T est ainsi représenté :



⟹ : *lien de chaine*
——— : *lien résolvant*
-·-·- : *lien ancestral*
----- : *lien d'admissibilité*
-o-o-o- : *lien facteur*

Figure VII.1 : Le S-L graphe G_T

Les chaines qui ont le support sont précédées d'un *.

Dans la suite, les liens de chaîne seront sous-entendus (la juxtaposition de 2 littéraux en tenant lieu), tous les autres liens seront représentés par un trait continu (leur type étant déterminé par le contexte).

Remarques -

1 - Les littéraux d'une tautologie potentielle sont reliés par un lien d'admissibilité.

2 - Un B-littéral qui ne possède ni lien résolvant ni lien ancestral est saturé ; toute chaîne de G_T contenant un B-littéral saturé sera supprimée de G_T.

3 - Entre 2 noeuds de G_T on a parfois la possibilité d'existence d'un lien facteur ou d'un lien d'admissibilité. La détermination du type du lien est faite lors de l'utilisation de G_T par la méthode de déduction.

4 - Nous envisageons la création d'un nouveau type de lien (lien de subsomption) bi-directionnel, qui relierait 2 B-littéraux potentiellement identiques (mod. θ) de 2 chaînes différentes et qui serait étiqueté par θ. Ces liens seraient utilisés pour des tests de subsomption.

VII.3 - La méthode de déduction des S-L graphes

Comme pour la SL -résolution, on attribue à tout littéral un statut de A-littéral ou de B-littéral ; les littéraux de $\mathcal{F}(S)$ sont des B-littéraux.

Soient T un ensemble de chaînes obtenues à partir des clauses de S en rangeant dans un ordre arbitraire les littéraux de ces clauses, T^* le sous-ensemble de T provenant d'un ensemble de support K de S.

En procédant comme en VII.2, on associe à T et T^* un S-L graphe G_T en créant, lorsque les deux possibilités existent, des liens facteurs au lieu de liens admissibles.

La méthode des S-L graphes est la suivante :

1 - Construire un S-L graphe correspondant à l'ensemble $\mathcal{F}_m(S) = \mathcal{F}(S)$ des m-facteurs de S

2 - Choisir dans le S-L graphe une chaîne C ayant le support

3 - Construire à partir de C une chaîne C' en utilisant l'une des trois opérations suivantes : e-extension, e-réduction, tronquation

4 - Inclure C', si elle est admissible, dans le S-L graphe en créant les liens correspondant aux littéraux de C' ; C' a le support.

Nous allons reprendre ces opérations dans le détail.

VII.3.1 - Construction du S-L graphe

Pour construire le S-L graphe correspondant à l'ensemble $\mathcal{F}_m(S)$, on peut opérer de la façon suivante en partant de G_T.

1 - Choix d'un lien facteur

Choisir un lien facteur λ entre 2 littéraux d'une chaîne C de G_T

2 - Construction d'une chaîne C'

Construire la chaîne C' obtenue à partir de C par la m-factorisation correspondant à λ et inclure dans G_T les noeuds correspondant aux occurrences de ses littéraux ; transformer λ en lien d'admissibilité ; C' a le support si C l'a

3 - Construction des liens de C'

Soit $L\theta$ une occurrence quelconque d'un B-littéral de C' dont l'ancêtre dans C est le B-littéral L ; construire les liens connectant $L\theta$ aux autres noeuds de G_T de la façon suivante :

3.1 - Liens résolvants hérités des parents

Si un lien résolvant étiqueté par σ relie L à un littéral P, et si θ et σ sont compatibles, connecter L et P par un lien résolvant étiqueté par le p.g.u. $\theta * \sigma$ de θ et σ.

3.2 - Liens résolvants avec le parent

Si L est potentiellement complémentaire (mod. σ) d'un B-littéral de C, connecter $L\theta$ et P par un lien résolvant étiqueté par σ.

3.3 - Liens non résolvants

Construire les liens facteurs, puis les liens d'admissibilité de $L\theta$ avec les littéraux de C' (autres que $L\theta$). Si un lien d'admissibilité est étiqueté par une substitution vide, supprimer la chaîne C' de G_T.

4 - Si G_T contient encore un lien facteur, recommencer au pas 1- ; sinon G_T est le S-L graphe recherché.

Remarque -

Cet algorithme peut engendrer plusieurs fois un même facteur ; on peut le compléter par un test à la fin du pas 2 qui détermine si la nouvelle chaîne C' est une variante d'une chaîne de G_T.

Les trois sous-sections qui suivent sont consacrées à la description des opérations de la méthode des S-L graphes ; on utilise, pour les définir, les opérations de la SL -résolution étendue.

Les chaînes auxquelles s'appliquent ces opérations sont appellées chaînes principales, par opposition aux autres chaînes utilisées (le cas échéant) par ces opérations ; ces dernières sont appellées chaînes auxiliaires.

VII.3.2 - L'opération de e-réduction

Soit C une chaîne dont le maillon actif μ_C est non vide telle que C ait le support dans le S-L graphe.

Pour construire une chaîne C' par e-réduction de C, on opère de la façon suivante:

1 - Choix d'un B-littéral et d'un lien

Choisir, dans μ_C, un B-littéral Q ; choisir dans Q un lien λ facteur (respectivement ancestral, résolvant - seulement si λ relie Q à un B-littéral qui est le seul B-littéral de sa chaîne -) ; λ est étiqueté par une substitution θ ;

2 - Construction de la chaîne

Construire la chaîne C' obtenue à partir de C par la m-factorisation (respectivement résolution ancestrale unitaire) correspondant à λ et supprimer λ du graphe ; inclure C' dans le graphe en lui attribuant le support ;

3 - Construction des liens

Soit $L\theta$ une occurrence quelconque d'un B-littéral de C' dont l'ancêtre dans C est le B-littéral L ;

3.1, 3.2 - Liens résolvants

Construire les liens résolvants de $L\theta$ de la même façon qu'en 3.1 et 3.2 du paragraphe VII.3.1

3.3 - Liens non résolvants

Soient $L_1\theta$ et $L_2\theta$ deux occurrences de littéraux de C' descendant de 2 littéraux L_1 et L_2 de C reliés par un lien λ étiqueté par σ ; si θ et σ

sont compatibles, connecter $L_1\theta$ et $L_2\theta$ par un lien de même nature que λ , étiqueté par $\theta * \sigma$. Si C' possède un lien d'admissibilité étiqueté par une substitution vide, supprimer C' du graphe.

VII.3.3 - L'opération de e-extension

Soit C une chaîne dont le maillon actif μ_C est non vide, telle que C ait le support dans le S-L graphe. Pour construire une chaîne C' par e-extension de C, on opère de la façon suivante:

1 - Sélection d'un B-littéral et choix d'un lien résolvant

Sélectionner un B-littéral P du maillon actif de C ; choisir, dans P, un lien résolvant λ étiqueté par une substitution θ (à condition que λ relie P à un B-littéral Q d'une chaîne non unitaire).

2 - Construction de la chaîne C'

Construire la chaîne C' obtenue à partir de C par la e-extension correspondant à λ et supprimer le lien λ; inclure C' dans le graphe en lui attribuant le support.

3 - Construction des liens de C'

Soit $L\theta$ une occurrence quelconque d'un B-littéral de C' dont l'ancêtre dans C ou C_1 est le B-littéral L.

3.1 - Liens résolvants hérités des parents

Adjoindre à L_θ des liens résolvants de la même façon qu'en 3.1 du paragraphe VII.3.1.

3.2 - Liens résolvants avec les parents

Si $L\theta$ est potentiellement complémentaire (mod. σ) d'un B-littéral P de C ou C_1, connecter $L\theta$ et P par un lien résolvant étiqueté par σ .

3.3 - Liens non résolvants hérités des parents

Soient $L_1\theta$ et $L_2\theta$ 2 occurrences de littéraux de C' descendant de 2 littéraux L_1 et L_2 de C ou de C_1, reliés tous deux par un lien λ étiqueté par σ ; si θ et σ sont compatibles, connecter L_1 et L_2 par un lien admissible, étiqueté par $\theta * \sigma$.

3.4 - Liens non résolvants non hérités des parents

Construire les liens facteurs et ancestraux puis les liens d'admissibilité reliant les B-littéraux du maillon actif de C' aux autres littéraux de C'. Si C' possède un lien d'admissibilité étiqueté par une substitution vide, C' est supprimée du graphe.

VII.3.4 - L'opération de tronquation

La tronquation d'une chaîne C du graphe s'effectue en supprimant du graphe les A-littéraux les plus à gauche de C, tant que le maillon actif de la chaîne ainsi obtenue demeure vide.

VII.4 - Arbre de recherche de la méthode des S-L graphes

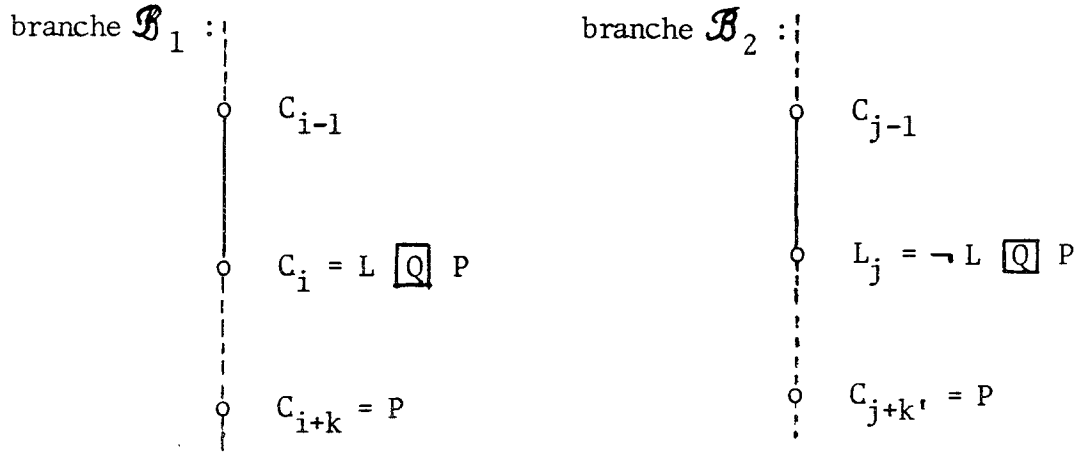
Sa définition est analogue à celle d'un arbre de recherche correspondant à la méthode des graphes de classification (cf. VI.3) ; notons toutefois que les descendants directs d'un noeud correspondent à des liens permettant d'effectuer une e-extension ou une e-réduction ; tous les noeuds sont de type N.R.A. et correspondent à des S-L graphes.

VII.5 - Avantages des S-L graphes par rapport à la SL - résolution pure

VII.5.1 - Redondance

La méthode des S-L graphes permet d'éviter un certain type de redondance (qui apparaît dans la SL -résolution), concrétisé par l'exemple suivant :

Soient S un ensemble de clauses, S_o un ensemble de support de S, ϕ une fonction de sélection ; supposons que l'espace de recherche E correspondant à la SL - résolution de S (avec S_o et ϕ contienne deux branches $\mathcal{B}_1$ et $\mathcal{B}_2$ de la forme suivante :



les maillons actifs de C_i et C_j contiennent des B-littéraux complémentaires ; la clause unitaire P pourrait être obtenue comme résultat d'une e-extension (de C_i avec C_j ou de C_j avec C_i) et d'une factorisation ; avec la SL - résolution, la clause unitaire P ne sera obtenue qu'après un parcours "aveugle" de $\mathcal{B}_1$ (ou de $\mathcal{B}_2$), c'est-à-dire sans tenir compte des chaînes engendrées dans des branches parallèles de E.

Ceci est encore plus regrettable dans le cas où C_i et C_j sont elles-même des clauses unitaires complémentaires qui pourraient directement engendrer la clause vide.

Cependant, en fonction de remarques faites par KOWALSKI et des résultats d'une expérimentation de la méthode des S-L graphes, nous avons été amené à limiter une telle utilisation des chaînes produites dans des branches parallèles de E ; en effet, l'utilisation inconsidérée de telles chaînes dans des opérations de e-extension peut, dans certains cas, accroître de façon sensible le nombre de chaînes produites et donc, le nombre de chaînes inutiles, ce qui contrebalance l'avantage décrit plus haut. Nous avons, en définitive, choisi un moyen terme qui consiste à n'utiliser que les clauses unitaires produites dans des branches parallèles de l'arbre E. Nous reviendrons dans la

section VII.7, sur la méthode de déduction qui en découle et que nous avons appelée méthode des S-L graphes simplifiée.

VII.5.2 - Principes de simplification

La méthode des S-L graphes incorpore de façon implicite trois des principes de simplification énoncés dans le chapitre V :

. Le principe de pureté

. Le principe de saturation

. Le principe d'instanciation.

Ces principes ne font pas partie de la SL -résolution et devraient lui être rajoutés, le cas échéant.

VII.5.3 - Recherche multi-directionnelle

Une stratégie Σ qui parcourt un arbre de recherche E correspondant à la SL - résolution d'un ensemble S de clauses avec ensemble de support S_o et fonction de sélection ϕ y effectue une recherche unidirectionnelle ; cela signifie que lorsque Σ est arrivé à un noeud N.R.A. de l'arbre E, Σ choisit une direction de recherche (cf. VI.1 et VI.2), dans E ; dans le cas où Σ engendrerait des dérivations dans des directions différentes, elle ne ferait qu'effectuer plusieurs recherches unidirectionnelles en parallèle ; on peut qualifier une telle recherche "multi-directionnelle" ; avec les S-L graphes, une stratégie peut tirer parti des résultats obtenus dans les autres directions ; cette possibilité est intéressante, par exemple, dans le cas de directions de recherche multiples associées à des clauses distinctes du support S_o.

Remarque -

Le terme "multi-directionnel" est un abus de langage. En effet, il s'agit ici de multi-direction en référence à l'espace de recherche de la SL-résolution. Bien entendu, par rapport à l'espace de recherche des S-L graphes,

la recherche ne peut qu'être unidirectionnelle, ou à la rigueur, multi-unidirectionnelle.

VII.5.4 - Structuration de l'information

Ce qui suit ne constitue pas, stricto sensu, un avantage des S-L graphes par rapport à la SL-résolution, mais constitue plutôt un avantage en soi.

Le fait de structurer les informations sous forme de S-L graphes évite de faire à chaque fois une recherche systématique dans tout l'ensemble de clauses, lors d'une extension ; les m-factorisations, les résolutions ancestrales et la vérification de la condition d'admissibilité s'en trouvent également simplifiées. De plus, un S-L graphe contient assez d'information pour permettre l'interruption et la reprise ultérieure de la recherche d'une preuve.

La structure de graphe présente un autre avantage : elle permet, éventuellement, de raffiner la fonction heuristique utilisée par une stratégie Σ diagonale. Montrons-le sur un exemple simple.

Supposons que Σ ait à choisir entre une dérivation D_1 de C_1 et une dérivation D_2 de C_2 (voir figure).

Une fonction heuristique h_o utilisant seulement la longueur de C_1 et C_2 serait telle que

$$h_o(D_1) < h_o(D_2)$$

La fonction heuristique h_1 qui tiendrait compte, en plus, de la longueur des chaînes reliées aux littéraux de C_1 et C_2 serait telle que :

$$h_1(D_1) > h_1(D_2)$$

Il est, dans ce cas, évidemment plus judicieux de dériver, en priorité, C_2 malgré sa longueur.

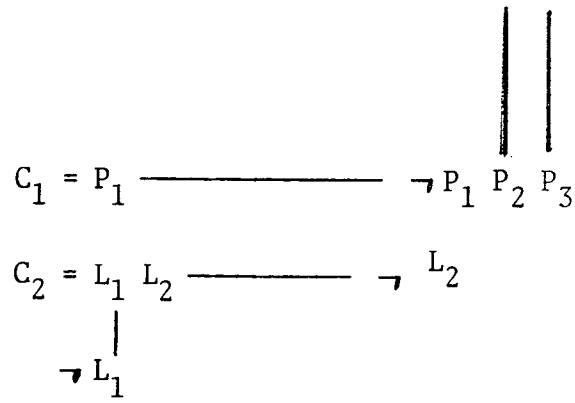


FIGURE VII.2 - C_2 conduit (en 2 résolutions) plus rapidement que C_1 à □.

La structure de graphe facilite ce genre de prédiction.

VII.6 - Avantages des S-L graphes par rapport aux graphes de classification

L'existence dans les S-L graphes de 2 types de littéraux, ainsi que de liens facteurs, ancestraux et d'admissibilité, permet de garder les avantages essentiels de la SL - résolution par rapport aux graphes de classification ; d'une part, le taux de branchement de l'arbre de recherche correspondant aux S-L graphes est inférieur à celui des graphes de classification ; d'autre part, le fonctionnement de la m-factorisation, de la résolution ancestrale sont rendus possibles. En outre, la simulation de la SL- résolution (étendue) permet d'utiliser la méthode du support.

La m-factorisation permet d'éviter la construction d'un même facteur plusieurs fois, ce qui supprime des redondances.

La résolution ancestrale (qui, dans ce cas et dans celui de la SL-résolution, est plutôt la résolution avec une instance d'un ancêtre), permet d'engendrer des clauses plus courtes que leurs parents, et équivaut à un test de subsomption rapide pour une certaine classe de chaînes.

Nous allons, dans la section qui suit, présenter une méthode de déduction définie à partir de la méthode des S-L graphes ; ces deux méthodes ont été programmées sur ordinateur et ont fait l'objet d'une expérimentation.

VII.7 - La méthode des S-L graphes simplifiée

Cette méthode a été obtenue en imposant certaines restrictions à la méthode des S-L graphes ; un grand nombre d'entre elles sont inspirées du système de FLEISIG et al. (1974) qui utilise la méthode de Model-elimination. Ces restrictions sont les suivantes :

- Une chaîne non unitaire, qui n'est pas une chaîne d'entrée, ne peut pas être utilisée comme chaîne auxiliaire lors d'une opération de e-extension ; ceci entraîne des modifications dans la structure de S-L graphe (nous appelerons cette nouvelle structure "S-L graphe simplifié").
- La fonction de sélection est déterminée une fois pour toute : le littéral sélectionné est toujours celui le plus à gauche du maillon actif.
- De même, l'opération de e-réduction n'est applicable qu'au B-littéral le plus à gauche du maillon actif.
- La factorisation non basique (i.e. nécessitant l'application d'une substitution) est supprimée ; ce qui conduit à définir une nouvelle opération (la post-factorisation), à modifier le test d'admissibilité ainsi que les conditions d'application de ce test.

VII.7.1 - Définition d'un S-L graphe simplifié

Soient $T = \{C_1, C_2, \ldots, C_n\}$ un ensemble de chaînes, $T^* \subseteq T$ un ensemble de support de T (cf. V.11).

Un S-L graphe simplifié G_T associé à T et T^* est défini ainsi :

1 - Les noeuds de G_T sont les occurrences des littéraux des chaînes de T.

2 - Il existe 5 types de liens entre les noeuds :

- les liens de chaîne
- les liens résolvants

- les liens facteurs
- les liens ancestraux
- les liens d'admissibilité.

Tous les liens sont uni-directionnels ; les liens résolvants et ancestraux sont étiquetés par un plus grand unificateur ; les liens facteurs et les liens d'admissibilité sont étiquetés par un indicateur booléen. Entre deux noeuds de G_T, il y a au plus un lien étiqueté.

(a) les liens de chaîne

Ils sont définis de la même façon qu'en VII.2.

(b) les liens résolvants

Soit une paire de B-littéraux L_1, L_2 de T satisfaisant aux conditions suivantes :

- L_1 et L_2 sont potentiellement complémentaires (mod. θ) ;
- A moins que L_1 appartienne à une chaîne ayant le support, L_1 n'est pas le littéral le plus à gauche de sa chaîne ;
- L_2 appartient à une chaîne d'entrée C et est le littéral le plus à gauche de C.

Un lien résolvant orienté de L_1 vers L_2 et étiqueté par θ, relie ces 2 littéraux.

(c) Les liens facteurs

Soit une paire de littéraux L_1, L_2 de T satisfaisant aux conditions suivantes :

- L_1 et L_2 sont potentiellement identiques (mod. θ) et appartiennent à une même chaîne C : $L_1 < L_2$ (i.e. L_1 plus à gauche que L_2) ;
- L_1 est un A-littéral ou un B-littéral ; L_2 est un B-littéral ; un

lien facteur orienté de L_1 vers L_2 et étiqueté par la valeur booléenne du prédicat $\theta = \varepsilon$, relie L_1 à L_2.

(d) Les liens ancestraux

Ils sont définis de la même façon qu'en VII.2, mais ils sont unidirectionnels et orientés du B-littéral vers le A-littéral.

(e) Les liens d'admissibilité

Soit une paire de littéraux L_1, L_2 appartenant à une chaîne de T, qui n'est pas une chaîne d'entrée, telle que $L_1 < L_2$ et vérifiant l'une des conditions suivantes:

(i) L_1 est un A-LITTERAL
ou L_1 est un B-LITTERAL et L_2 est un A-LITTERAL potentiellement IDENTIQUE (mod. θ) ;

(ii) L_1 est un A-LITTERAL et L_2 est un A-LITTERAL potentiellement COMPLEMENTAIRE (mod. θ) ;

(iii) L_1 est un B-LITTERAL et L_2 est un B-LITTERAL potentiellement COMPLEMENTAIRE (mod. θ) ;

(iiii) L_1 est un A-LITTERAL et L_2 est un B-LITTERAL potentiellement COMPLEMENTAIRE (mod. θ).

Un lien d'admissibilité orienté de L_1 vers L_2 et étiqueté par la valeur booléenne du prédicat $\theta = \varepsilon$ relie L_1 à L_2.

3 - On définit comme en VII.2 une chaîne de G_T, une chaîne ayant le support dans G_T.

VII.7.2 - La méthode de déduction

Soit $C = L_1\ L_2 \ldots L_{i-1}\ L_i\ L_{i+1} \ldots L_p$; toute chaîne $C_i = L_i\ L_1\ L_2 \ldots L_{i-1}\ L_{i+1} \ldots L_p$ obtenue à partir de C en rangeant à gauche un littéral quelconque L_i de C sera appelée pseudo-permutation de C ;

il en existe p ; soit G_T le S-L graphe simplifié construit à partir de l'ensemble des pseudo-permutations des chaînes d'entrée de S et de K, l'ensemble de support de S. Les clauses ayant le support dans G_T correspondent, soit aux chaînes provenant de K, soit à l'ensemble $\tilde{K}$ des pseudo-permutations des chaînes provenant de K, soit à un sous-ensemble de $\tilde{K}$ (ce choix affine les possibilités du support et est un paramètre de la méthode).

La méthode des S-L graphes simplifiée est la suivante :

1 - Choisir dans le S-L graphe simplifié une chaîne C ayant le support

2 - Construire à partir de C une chaîne C' en utilisant l'une des quatre opérations suivantes : extension, e-réduction, tronquation, post-factorisation

3 - inclure C', si elle est admissible, dans le S-L graphe simplifié, en créant les liens correspondant aux littéraux de C' ; C' a le support.

VII.7.2.1 - L'opération de e-réduction

Soit C une chaîne dont le maillon actif μ_C est non vide, telle que C ait le support dans le S-L graphe simplifié ; soit Q le B-littéral le plus à gauche de μ_C. Pour construire une chaîne C' par e-réduction de C, on opère de la façon suivante:

1 - Choix d'un lieu

1.1 - Factorisation basique. Q possède un lien facteur étiqueté par la valeur booléenne VRAI ; la chaîne C' est la chaîne C amputée de Q ; l'opération est terminée.

1.2 - Résolution ancestrale ou unitaire d'entrée. Q possède un lien λ ancestral (respectivement résolvant, seulement si λ relie Q à un B-littéral d'une chaîne unitaire) ; λ est étiqueté par une substitution θ ;

1.3 - Résolution unitaire non-d'entrée . Il existe dans le S-L graphe simplifié, une chaîne unitaire (qui n'est pas une chaîne d'entrée) dont le B-littéral P est potentiellement complémentaire (mod. θ) de Q ; soit λ le lien de Q vers P, construit pour la circonstance ;

2 - Construction de la chaîne C'

- Construire la chaîne C' obtenue à partir de C par la résolution ancestrale (respectivement unitaire) correspondant à λ , et supprimer λ du graphe ; inclure C' dans le graphe en lui attribuant le support.

3 - Construction des liens

3.1 - Liens résolvants

On ne construit que les liens résolvants hérités des parents (cf. 3.1 dans le paragraphe VII.3.1).

3.2 - Liens non-résolvants (cf. 3.3 dans le paragraphe VII.3.2)

. Si C' possède un lien facteur, reliant un A-littéral à un B-littéral, étiqueté par la valeur booléenne VRAI, C' est post-factorisée ; sinon,

. Si C' possède un lien d'admissibilité étiqueté par la valeur booléenne VRAI, C' est supprimée du graphe.

VII.7.2.2 - L'opération d'extension

Soit C une chaîne dont le maillon actif μ_C est non vide, telle que C ait le support ; soit Q le B-littéral le plus à gauche de μ_C.

Pour construire une chaîne C' par extension de C, on opère de la façon suivante:

1 - Choix d'un lien résolvant

Choisir dans Q un lien résolvant λ étiqueté par une substitution θ (à condition que λ relie Q à un B-littéral P d'une chaîne non unitaire).

2 - Construction de la chaîne C'

Construire la chaîne C' obtenue à partir de C par l'extension correspondant à λ (cf. V.11) et supprimer le lien λ ; inclure C' dans le graphe en lui attribuant le support.

3 - Construction des liens de C'

3.1 - Liens résolvants

On ne construit que les liens résolvants hérités des parents (cf. 3.1 dans le paragraphe VII.3.1).

3.2 - Liens non-résolvants (cf. 3.3 et 3.4 dans le paragraphe VII.3.3)

- Si C' possède un lien facteur, reliant un A-littéral à un B-littéral, étiqueté par la valeur booléenne VRAI, C' est post-factorisée ; sinon,
- Si C' possède un lien d'admissibilité étiqueté par la valeur booléenne VRAI, C' est supprimée du graphe.

VII.7.2.3 - L'opération de tronquation (cf. VII.3.4)

VII.7.2.4 - L'opération de post-factorisation (FLEISIG et al. 1974)

Pour post-factoriser une chaîne C on supprime du graphe les littéraux de C, en parcourant C de gauche à droite, jusqu'à ce que la chaîne ainsi obtenue ne contienne plus de lien facteur, reliant un A-littéral à un B-littéral, étiqueté par la valeur booléenne VRAI.

Cette opération peut être considérée comme une factorisation effectuée rétroactivement. En effet, la chaîne C_3 obtenue par post-factorisation de C_2 est une instance de la chaîne qui aurait été obtenue par factorisation d'un

ancêtre C_1 de C_2 (ancêtre dans lequel le B-littéral le plus à gauche a pour descendant dans C_1, le A-littéral le plus à droite supprimé par la post-factorisation de C_1).

Exemple -

C_1 = P(x) P(a)

si C_2 = Q(a) $\boxed{P(a)}$ P(a) est un descendant de C_1,

alors C_3 = P(x) est obtenue par post-factorisation de C_2

VII.7.3 - Remarques sur la complétude de la méthode

. Si le test d'admissibilité était réduit aux cas (i) et (ii) énoncés dans le sous-paragraphe (e) du chapitre VII.7.1, la méthode serait complète d'après le résultats de LOVELAND (1970 b- Model-elimination).

.. Lorsque la factorisation (non basique) est permise, la méthode est complète avec un test d'admissibilité comportant tous les cas possibles (cf. V.11 SL -résolution).

. L'introduction de l'opération de post-factorisation ainsi que le fait que son applicabilité soit testée avant la condition d'admissibilité, sont des conditions nécessaires à la complétude de la méthode des S-L graphes simplifiée.

La question de savoir si ces conditions sont suffisantes pour la complétude de cette méthode n'a pas été résolue.

VII.7.4 - Arbre de recherche de la méthode des S-L graphes simplifiée

Sa définition est analogue à celle d'un arbre de recherche correspondant à la méthode des S-L graphes (VII.4). Les noeuds sont de type N.R.A; et correspondent à des S.L graphes simplifiés.

TROISIEME PARTIE

MISE EN OEUVRE D'UN DEMONSTRATEUR SUR ORDINATEUR

- Description du système réalisé
- Expérimentation du système
- Extensions et modifications du système

VIII - DESCRIPTION DES SYSTEMES REALISES

Les démonstrateurs que nous avons réalisés utilisent une stratégie diagonale montante pour parcourir des arbres de recherche relatifs à la méthode de déduction des S-L graphes et à celle des S-L graphes modifiée. Nous exposons tout d'abord quelques principes généraux de programmation qui se sont imposés au fil de notre expérience, puis nous donnerons les raisons qui nous ont conduit au choix du langage LISP ; nous décrirons ensuite les algorithmes de mise en oeuvre de la stratégie diagonale montante et la représentation interne des données ; nous terminerons le chapitre par les modifications introduites dans le système pour la méthode des S-L graphes modifiée.

VIII.1 - Méthodologie

Pour la programmation de ce système nous avons adopté les principes généraux suivants :

- Toute modification de la représentation interne de la structure des données ne doit entraîner qu'un minimum de modifications dans le programme.
- Un algorithme doit pouvoir être modifié facilement ; ce qui s'est traduit par le choix des options suivantes:

1 - Le programme communique avec ses données uniquement par l'intermédiaire de fonctions d'accès ; ces fonctions sont caractéristiques de la structure de données et toute modification de cette structure n'entraîne des modifications que sur ces fonctions.

2 - Le programme est très structuré ; cela signifie qu'une fonction qui effectue une opération complexe utilise pour cela des fonctions plus simples qui, à leur tour, en utilisent d'autres encore plus simples, etc... ce qui entraîne une hiérarchisation des actions.

On peut objecter à ce choix le fait qu'il constitue un frein à l'efficacité des programmes, car il implique un grand nombre d'appel de fonctions. Nous pensons néanmoins que ce choix est justifié, car :

- Cette façon de programmer augmente notablement la lisibilité des programmes (ce qui facilite la mise au point du programme ainsi que sa compréhension par autrui) : les noms des fonctions sont autant de commentaires implicites; les divers pas d'un algorithme sont mis en évidence.

- Les modifications ne se font, en général, que localement.

- Le gain de temps et d'effort, lors de la mise au point, justifie largement l'effort supplémentaire lors de la conception et de l'écriture des programmes.

- La transformation du programme sous une forme plus efficace est toujours possible (et simple à effectuer) une fois que celui-ci est au point.

VIII.2 - Choix du langage

Le système est programmé dans une version de LISP 1.5 (MAC CARTHY, 1961) utilisée dans l'environnement conversationnel et à temps partagé CP-CMS (SIRET, 1967) fonctionnant sur l'ordinateur IBM 360/67 du CICG de Grenoble. Nous avons choisi LISP pour de multiples raisons.

1 - Ce langage est spécialisé dans le traitement des listes. La representation interne des expressions (clauses, termes), ensembles d'expression et de leurs attributs, se fait commodément (cf. VIII.4), à l'aide de listes.

2 - LISP traite efficacement la récursivité, que nous utilisons dans plusieurs fonctions du système ; cependant, dans un souci d'optimisation, nous avons, dans certains cas, préféré la forme itérative à la forme récursive.

3 - LISP est un langage interprété (bien que l'on ait la possibilité de compiler des fonctions), et de plus, l'interpréteur LISP est facilement modifiable.

Cette dernière possibilité (cf. LUX, 1974) a été largement utilisée, notamment en ce qui concerne :

. l'impression de listes circulaires (possibilité d'imposer à l'interpréteur son propre programme d'impression)

. la gestion de blocs de mémoires consécutives (tableaux)

. l'utilisation de fonctions codées en langage machine (et, en particulier, de fonctions précompilées).

Le fait que LISP soit un interpréteur présente les avantages suivants :

. mise au point des programmes : il est, en effet, facile (et commode) de définir une fonction spéciale que l'on insère provisoirement dans la ou les parties douteuses d'un programme, et qui permet d'en interrompre l'exécution à un endroit choisi, de vérifier et/ou modifier le contenu des variables, et de reprendre l'exécution du programme (pas nécessairement au même endroit), toutes ces opérations étant effectuées interactivement dans le cadre d'un système conversationnel

. évaluation des littéraux (cf. X.6) ; elle peut se faire par un appel (récursif) à l'interpréteur qui considère alors le littéral comme une fonction à évaluer, le résultat (booléen) étant transmis au démonstrateur.

4 - Il est possible de définir des "MACROS" en utilisant des fonctions spéciales.

Exemples -

- (WHILE B I_1 I_2 .....I_n)

sera remplacé par le groupe d'instructions :

ETI_1

(CØND [(NOT B) (GØ ETI_2)])

I_1

I_2

:

I_n

(GØ ETI_1)

ETI_2

- (SETCDR X) sera remplacé par (SETQ X (CDR X))

5 - LISP est un langage qui se prête assez bien à la validation de programmes (cf. BOYER et MOORE, 1973).

VIII.3 - Mise en oeuvre de la stratégie diagonale montante

Soit D, une dérivation d'une chaîne quelconque C.

La fonction heuristique h actuellement utilisée, fait correspondre à D, le nombre h(D) de B-littéraux de C.

La fonction de coût g actuellement utilisée, fait correspondre à D, le nombre g(D) calculé récursivement de la façon suivante :

- Si C est un axiome, g(D) = 0
- Si C est obtenue par résolution de C_1 et C_2,
  $g(D) = \text{Max}\,[g(D_1)\ ,\ g(D_2)] + 1$, où D_1 et D_2 sont respectivement des dérivations de C_1 et C_2 (et des sous-dérivations de D).
- Si C est obtenue par m-factorisation de C_1
  $g(D) = g(D_1) + 1$, où D_1 est une dérivation de C_1 (et une sous-dérivation de D).

Toute chaîne est représentée par une liste qui contient une sous-liste

constituée par le ou les ancêtres immédiats de cette chaîne (cf. VIII.4) ; la liste qui représente C est donc en même temps une représentation de D. TAB est un tableau à 2 dimensions rempli de telle façon que l'élément TAB (i,j) contienne la liste des chaînes C_k telles que $h(C_k) = i$ et $g(C_k) = j$, i.e. telles que C_k soit de mérite (i,j).

La bijection $\mathbb{N}^2 \xrightarrow{\beta} \mathbb{N}$ définie par :

$$\beta(i,j) = \frac{(i+j)\ (i+j+1)}{2} + j+1$$

est telle que :

(i,j) de meilleur mérite que (i',j') $\iff$ $\beta(i,j) < \beta(i',j')$

il suffit de vérifier que $\beta(i,j)$ est une énumération diagonale montante (cf. Fig. VIII.1) des points de $\mathbb{N}^2$

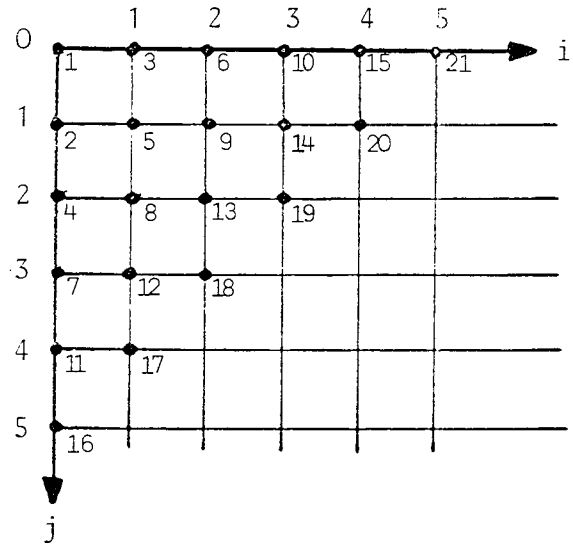


Figure VIII.1 : énumération par la bijection β des points de $\mathbb{N}^2$

La fonction β est utilisée comme fonction d'adresse de TAB. Au départ, TAB est initialisé avec les chaînes d'entrée (cf. V.11) .

Exemple -

Si $P(a) \underset{V}{} P(x) \in S$, $P(a)\ P(x) \in TAB(2,0)$ et $P(a) \in TAB(1,1)$

La stratégie Σ remplit TAB par ordre de mérite décroissant, c'est-à-dire que lorsqu'une chaîne de mérite (i,j) est construite et introduite dans TAB(i,j), toute autre chaîne, de mérite (i',j'), constructible à partir du S-L graphe est telle que, $\beta(i,j) \leq \beta(i',j')$. Le fonctionnement de Σ est illustré par un exemple dans l'appendice B.

Pour remplir le tableau, Σ utilise deux sous-fonctions : DIAGAMONT et DIAGAVAL.

DIAGAMONT est la fonction principale ; elle a deux arguments entiers i et j ; DIAGAMONT (i,j) remplit TAB (i,j) en effectuant des e-extensions ou des e-réductions (seulement dans le cas de la résolution unitaire).

Soient C_1 une chaîne de $TAB(i_1, j_1)$, C_2 une chaîne de $TAB(i_2,j_2)$ telles que :

- Soit l'opération de e-extension avec C_2 s'applique à C_1 ,
- Soit C_2 est unitaire de niveau $j_2 = \max(j_1,j_2)$ et C_1 se résoud avec C_2 (e-réduction de C_1).

Soient $i = i_1 + i_2 - 2$ et $j = \max(j_1, j_2) + 1$

La chaîne C résultante est construite et introduite dans TAB (i,j) par DIAGAMONT (i,j) ; lorsque toutes les chaînes possibles ont été introduites dans TAB (i,j), DIAGAMONT (i',j') est activée, avec i',j' tels que :

$$\beta(i',j') = 1 + (i,j) \quad \text{si} \quad j \neq 1 ,$$

$$\beta(i',j') = 3 + (i,j) \quad \text{si} \quad j = 1$$

Dans le 2ème cas, les 2 éléments sautés par Σ sont TAB (i+1, 0) et TAB(0, i+2) ; le 1er élément correspond aux chaînes d'entrée introduites lors de l'initialisation de TAB ; le second élément est sauté, car l'élément TAB (1, i+1) doit être nécessairement rempli avant lui.

DIAGAVAL est une fonction récursive qui a 3 arguments : une chaîne C et 2 entiers i et j ; si C est de mérite (i+1,j-1) et si l'opération de e-réduction s'applique à C, DIAGAVAL (C,i,j) engendre des chaînes de mérite (i,j) (meilleur que (i+1,j-1)) obtenues par e-réduction de C et les introduit dans TAB(i,j) ; de plus, DIAGAVAL est réactivé sur ces chaînes. DIAGAVAL(C,i,j) engendre donc des chaînes en descendant la diagonale sur laquelle se trouve C ; la profondeur de la récursion est bornée par i+j. D'une manière générale, dès qu'une chaîne C de mérite (i,j) est engendrée par DIAGAMONT ou DIAGAVAL, DIAGAVAL (C,i-1,j+1) est appelée pour construire des chaînes de meilleur mérite que C.

Pour construire les chaînes de mérite (i,j), DIAGAMONT effectue des e-extensions ou des e-réductions entre les chaînes de TAB (i-ℓ+2,j-1) et des chaînes de TAB (ℓ,k) avec $0 \leq k \leq j-1$ et $2 \leq \ell \leq i+1$.

Σ, DIAGAMONT et DIAGAVAL se décrivent en LISP de la façon suivante : (les identificateurs qui apparaissent dans les programmes LISP qui suivent ne sont pas toujours identiques à ceux du programme implanté sur l'ordinateur).

```
DEFINE2 ((
 (SIGMA
   (LAMBDA (DMAX)       (PROG  (D I J J1  N1 EXPN)
       (SETQ    N1  (TAB 1 0))
   (WHILE  N1
        (COND [ (DIAGAVAL  N1 0 1 ) (RETURN T)] )
        (SETSUIV  N1 ))
        (SETQ  D  1)
        (SETQ  EXPN T)
(WHILE  (LESSP  D  DMAX)
     (SETQ  J1  (SUB1 (SETQ  J  D )))
     (SETQ  D   (ADD1  D ))
     (SETQ  I  1)
(WHILE   (NOT (ZEROP  J ))
      (COND [(DIAGAMONT  I  J ) (RETURN  T )] )
      (SETQ  I  (ADD1  I ))
      (SETQ  J1 (SUB1 (SETQ  J  J1 ) ))   ))
)))
```

```
(DIAGAMONT
    (LAMBDA (I J) (PROG ( L I1 LT LNS LIEN N1  N2  N )
        (SETQ L (ADD1 I) )
        (SETQ I1    1 )
(REPEAT     (GREATERP [ PROG2 (SETQ  I1 (ADD1 I1) ) (SETQ L (SUB1 L) ) ] 1)
        (SETQ  N1 (TAB I1  J1) )
    (WHILE  N1
          (COND [(NULL (SETQ  LT (SELECT  N1) ) (GO SUIVANT) ])
          (SETQ    LNS    (CDR LT) )
          (SETQ    LT     (CAR LT) )
    (WHILE  LNS
          (SETQ  LIEN  (CAR  LNS) )
          (COND  [ (AND  ( ADDCHAINE  N1  LT  LIEN  I J )
                         (DIAGAVAL  (TRONQUATION  N) (SUB 1 I) (ADD1  J) ))
                   (RETURN  T) ] )
          (SETCDR  LNS) )
    SUIVANT
          (SETSUIV  N1) ))
         )))
(DIAGAVAL
   (LAMBDA  (N1  I  J) (PROG (N  LNS  LN  LT  EXPN)
       (COND [(MINUSP  I) (RETURN (CSETQ  CHAINEVIDE  N1) )])
       (SETQ LT (LITS  N1) )
   (WHILE (AND  LT (NOT (ALIT  LT)))
       (SETQ  LNS  (LIENS  LT ))
```

```
(WHILE LNS
       (SETQ LN (CAR LNS) )
       (COND [ (AND (OR (REDUC LN)
                        (AND (RES LN)
                             (EQUAL (HEURIST (SETQ N (CHAINE LN) )) 1)
                             (LESSP (COUT N) J) ))
                    (ADDCHAINE N1 LT LN  I  J)
                    (DIAGAVAL (TRONQUATION N) (SUB1 I) (ADD1 J) ))
               (RETURN T) ])
       (SETCDR  LNS) )
 (SETSUIV  LT) )
)))     ))
```

DEFINE2 est une fonction analogue à DEFINE qui permet en outre le traitement des macros (cf. VIII.2).

REPEAT est une macro identique à WHILE sauf en ce qui concerne le test booléen qui est engendré après le groupe d'instructions.

SETSUIV est une macro ; (SETSUIV N) engendre : (SETQ N (SUIV N)).

Soient $\mathcal{C}$ une suite de chaînes, N une chaîne de $\mathcal{C}$, SUIV est une fonction d'accès qui fournit comme résultat la chaîne qui suit N dans $\mathcal{C}$.

SELECT (N) sélectionne un B-littéral L dans le maillon actif de N, en vue d'effectuer des e-extensions de N, et choisit les liens résolvants $\lambda_1, \lambda_2, \ldots, \lambda_k$ de L qui conviennent pour cela ; le résultat est le doublet $(L . (\lambda_1, \lambda_2, \ldots, \lambda_k))$.

ADDCHAINE (N1 LT LN I J) effectue, selon la valeur booléenne de l'indicateur EXPN, une e-expansion ou une e-réduction de mérite (I,J),de la chaîne N1 sur le littéral LT en utilisant le lien LN.

CHAINEVIDE est une variable globale qui est utilisée en fin de programme, pour construire la preuve.

Le paramètre DMAX de SIGMA est utilisé pour arrêter le programme si SIGMA n'a pas trouvé de solution jusqu'à la diagonale DMAX.

L'appendice A contient une description sommaire des autres fonctions du système.

VIII.4 - Représentation interne du S-L Graphe

Dans la suite, lorsque nous parlerons d'un doublet, nous ferons référence à la structure de données élémentaire de LISP constituée par un couple de mémoires, D = (M_1, M_2) construit par un appel de CONS (M_1, M_2) et tel que M_1 = CAR (D), M_2 = CDR (D).

Les chaînes du S-L graphe sont classées selon leur mérite, dans le tableau TAB. Chaque chaîne C est représentée par une paire de doublets dont le premier contient un pointeur vers la liste des littéraux de C, et le second contient un pointeur vers la liste d'identification de C ; cette liste contient un numéro de sérialisation de C, des indications sur son origine (ancêtre (s) immédiat (s) ou nom de l'axiome), les valeurs de g (D) et h (D), D étant une dérivation de C, deux valeurs booléennes indiquant respectivement si C a le support et si C fait encore partie du S-L graphe (indicateur "MORT") ; de même qu'une chaîne, un littéral est représenté par une paire de doublets ; le premier contient le nom du prédicat, le second contient un pointeur vers la liste d'identification du littéral. Cette liste contient le signe du littéral, son statut (A ou B), un pointeur vers la liste de ses arguments et un pointeur vers la liste des liens qui lui sont connectés dans le S-L graphe. Un lien d'un littéral L est représenté par 3 doublets ; le premier indique la nature du lien (dans le cas d'un lien résolvant, pointeur vers la chaîne qui contient le littéral connecté à L), le second contient un pointeur vers le littéral connecté à L, le 3ème contient un pointeur vers le plus grand unificateur des 2 littéraux connectés. Il contient également un indicateur spécial (* LIEN *) utilisé par la fonction d'impression des chaînes, pour détecter le caractère circulaire d'une liste.

Dans une chaîne C quelconque, les occurrences distinctes d'une variable X sont représentées par des doublets distincts (occurrences secondaires de X) contenant un pointeur vers un même doublet : l'occurrence principale de X ; lors du calcul ou de l'utilisation, d'une substitution, seules sont éven-

tuellement modifiées les parties CAR des occurrences principales des variables. Une suite de termes est représentée par un pseudo-arbre binaire comme nous allons le voir sur un exemple.

Exemple -

Soit la suite de 3 termes :

$$\mathcal{S} = h(x) , f(h(x) , g(b,a)) , b$$

$\mathcal{S}$ est représentée par le pseudo-arbre binaire suivant :

x_o représente l'occurrence principale de x.
A chaque occurrence (principale ou secondaire) d'une variable, correspond 1 doublet ; a chaque symbole fonctionnel correspondent 2 doublets.

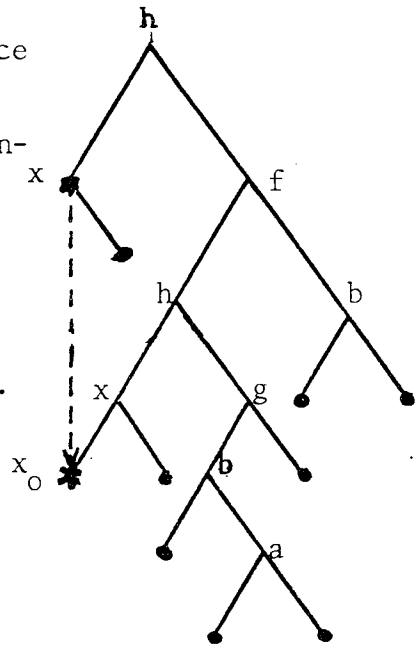


Figure VIII.2 : Pseudo-arbre binaire correspondant à $\mathcal{S}$.

Une substitution unificatrice σ est représentée par une liste d'un nombre pair de doublets ; un doublet de rang impair correspond à une occurrence principale de variables, le doublet, de rang pair suivant, correspondant au terme substitué à cette variable dans σ .

Exemple de représentation (partielle) d'un S-L graphe

Supposons qu'un S-L graphe contienne les 2 chaînes suivantes :

$C_1 = D(a,x)$

$C_2 = \neg D\ (a,f(a,x))$ | $D(x,y)$ |

Supposons que C_1 possède le support et soit une chaîne d'entrée. Posons $h(C_1) = i_1$, $g(C_1) = j_1$, $h(C_2) = i_2$, $g(C_2) = j_2$

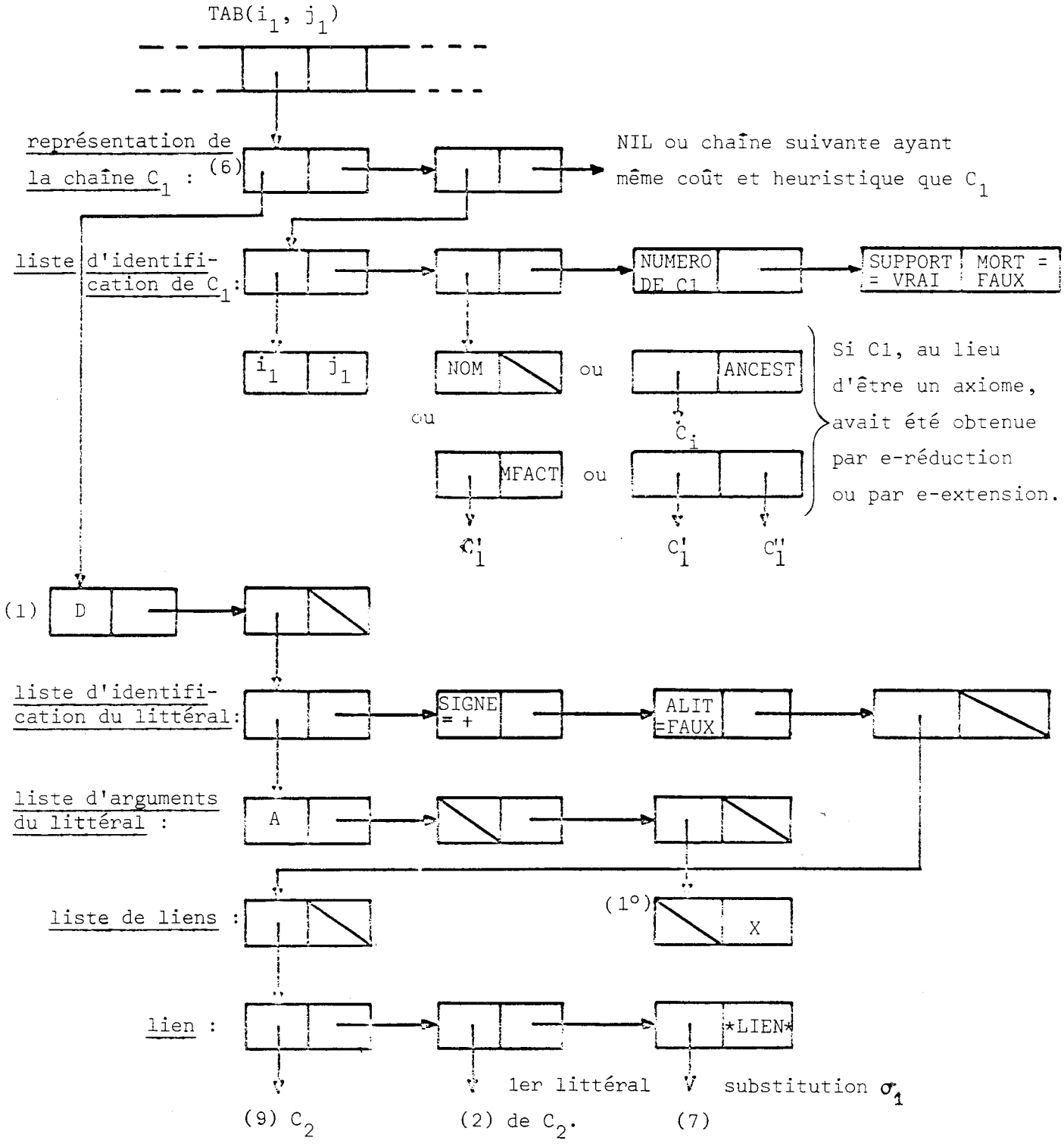


Figure VIII.3 : Représentation d'un S-L graphe - (1ère partie)

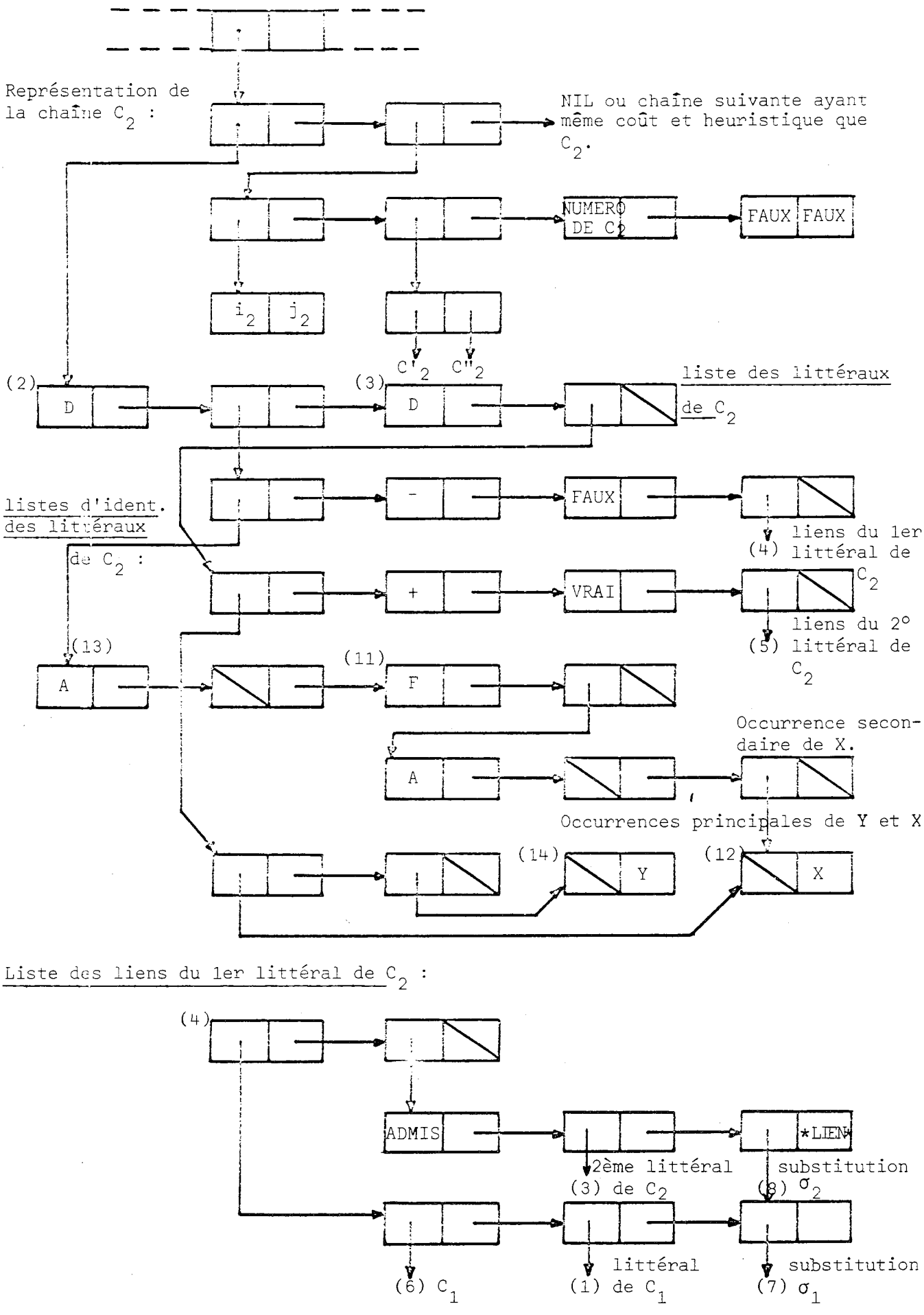
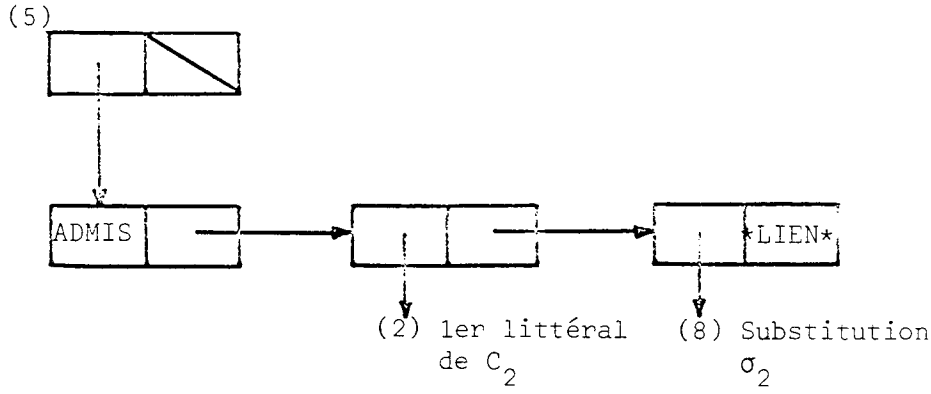
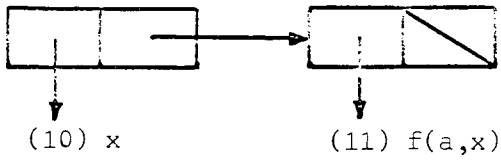


Figure VIII.3 : Représentation interne d'un S-L graphe (2ème partie)

Liste des liens du 2ème littéral de C_2 :



Substitution σ_1 :



Substitution σ_2 :

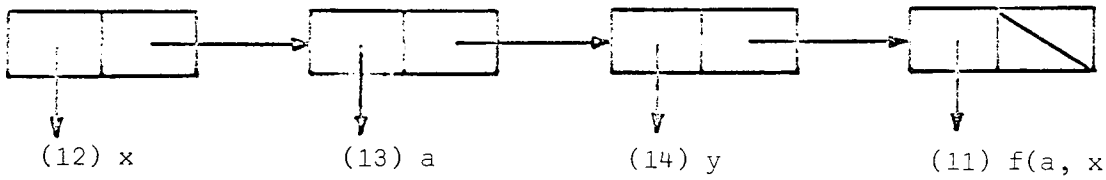


Figure VIII.3 : représentation interne d'un S-L graphe - (3ème partie)

Considérons la liste de termes $\mathcal{L}$ = (x,x,y) et la substitution σ = {x.b} représentées de la façon suivante :

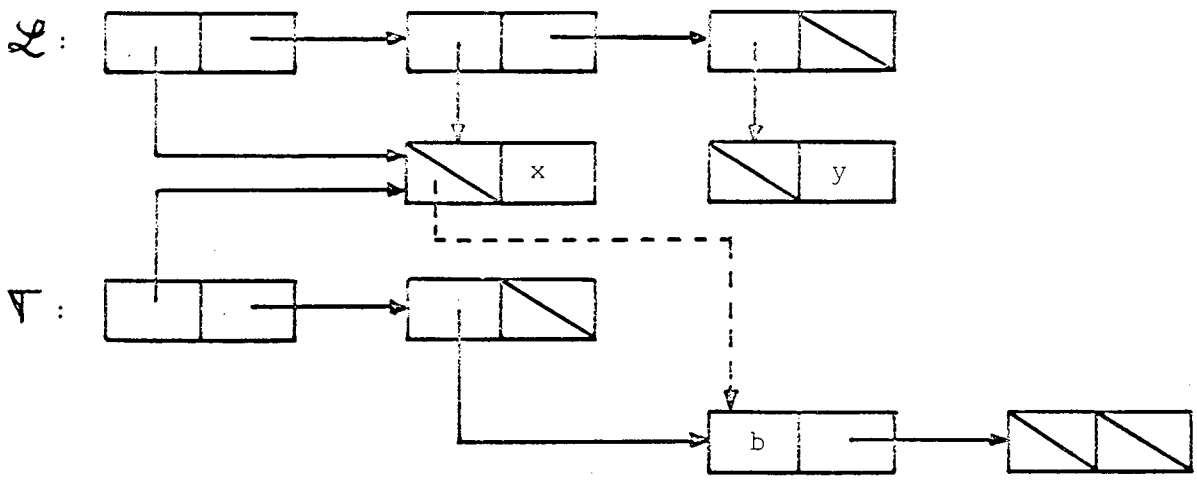


Figure VIII.4 : Utilisation de la substitution σ pour la génération de la liste $\mathcal{L}_\sigma$.

$\mathcal{L}_\sigma = (b , b , y)$.

Pour générer $\mathcal{L}_\sigma$, on remplace le NIL de l'occurrence principale de x par un pointeur vers le terme b et on recopie la liste $\mathcal{L}$. Les listes constantes de termes ne sont représentées qu'une seule fois ; elles sont rangées dans une liste spéciale, par l'intermédiaire d'une fonction d'adressage dispersé (hash-coding). Cette représentation des termes est inspirée de ROUSSEL (1972).

VIII.5 - Principes de simplification du S-L graphe

Il est possible de supprimer certains liens du S-L graphe, et donc, le cas échéant, de supprimer certaines chaînes, par un examen approprié des substitutions qui étiquettent ces liens. Nous étudions ici deux types de simplification :

- La suppression des tautologies
- La suppression des liens incompatibles.

Ces principes de suppression ne concernent que des liens résolvants et sont des principes provenant de la méthode des graphes de classification.

VIII.5.1 - Suppression des tautologies

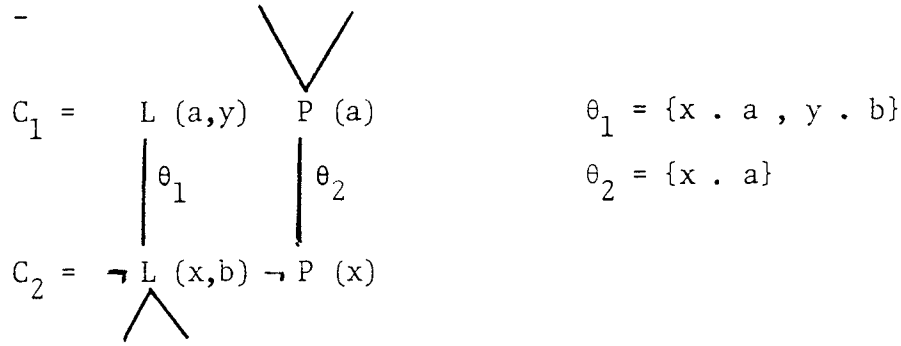
Si le graphe contient deux liens résolvants λ_1 et λ_2 étiquetés respectivement par les substitutions θ_1 et θ_2, et tels que :

1 - λ_1 et λ_2 connectent tous deux des littéraux d'une chaîne C_1 à des littéraux d'une chaîne C_2

2 - θ_2 est plus générale que θ_1

On peut supprimer θ_1 du graphe.

Exemple -



On peut supprimer θ_1, car le résolvant correspondant serait une tautologie ; par contre, on ne peut pas supprimer θ_2, car le résolvant correspondant ne serait qu'une tautologie potentielle (mod. {y . b}). Dans l'exemple présent, la suppression de θ_1 entraîne celle de C_1 (L saturé), puis celle de C_2. Les fonctions SUPTAUTOLN et SUPTAUTOLNS effectuent la suppression des tautologies.

VIII.5.2 - Suppression des liens incompatibles

Deux liens sont incompatibles si les substitutions qui les étiquettent sont incompatibles (cf. IV.8.7).

On peut représenter par un arbre ET/OU l'ensemble des substitutions étiquetant les liens résolvants des B-littéraux L_1, L_2,...., L_k d'une chaîne C. La racine de l'arbre est un noeud ET et correspond à C ; à chaque B-littéral L_i de C, correspond un descendant de C qui est un noeud OU ; à chaque lien résolvant λ_j de L_i correspond un descendant de L_i qui est un noeud ET ; à chaque composant de la substitution étiquetant λ_j, correspond un descendant de λ_j qui est une feuille de l'arbre.

Notons F(N) l'ensemble des feuilles du sous-arbre dont la racine est N. Nous dirons qu'un composant x,τ est incompatible avec un ensemble E de composants si :

1 - Il existe au moins un composant dans E dont la variable soit x.

2 - Pour tout composant $x.\tau'$ de E, τ et τ' ne sont pas unifiables.

La substitution θ d'un lien d'un littéral L d'une chaîne C est incompatible avec les substitutions des liens des autres littéraux de C si et seulement si

$\exists x \,.\, \tau \in \theta$, $\exists L' \in C : L \neq L'$ et $x \,.\, \tau$ incompatible avec $F(L')$.

Tout lien incompatible peut être supprimé, car le résolvant qui lui correspond possèderait au moins un littéral saturé.

Exemple -

$$
\begin{array}{lll}
 & \neg L\ (b,y)\ Q_1 & P(a,u)\ Q_2 \\
 & \lambda_1 \mid \theta_1 \qquad & \lambda_4 \mid \theta_4 \\
C = & L(x,a) & \neg\ P(x,b) \\
 & \lambda_2 \mid \theta_2 \qquad & \lambda_3 \mid \theta_3 \\
 & \neg\ L(a,a)\ Q_3 & P(c,b)\ Q_4
\end{array}
$$

L'arbre ET/OU des substitutions correspondant à C est le suivant :

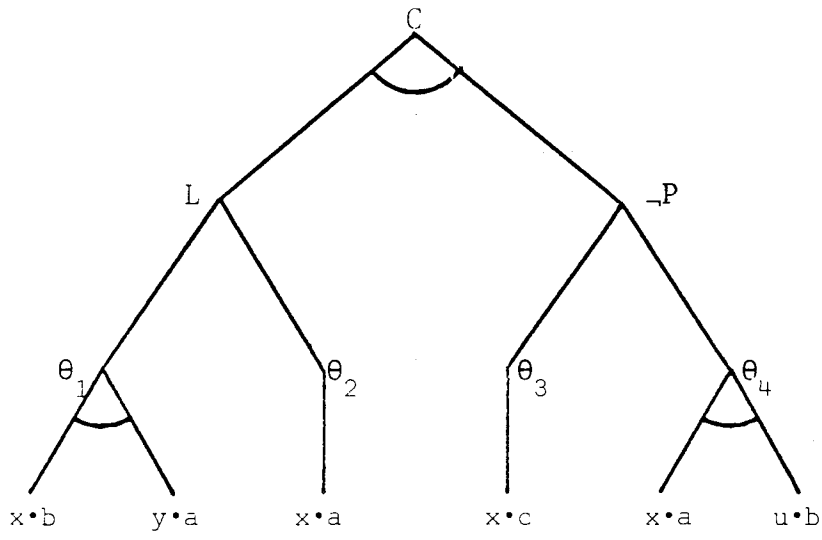


Figure VIII.5 : Arbre ET/OU des substitutions de C.

x.b est incompatible avec $F(\neg P)$

x.c est incompatible avec $F(L)$

Dans cet exemple, on peut supprimer λ_1 et λ_3.

La fonction qui met en oeuvre le principe précédent est récursive car toute suppression de lien peut impliquer de nouvelles incompatibilités. Cette fonction peut être appelée chaque fois que l'on ajoute ou que l'on supprime un lien du S-L graphe. Cependant, cette fonction ainsi que celle qui met en oeuvre le 1er principe, sont assez coûteuses ; nous avons donc choisi de rendre leur application optionnelle, en fonction de la valeur booléenne attribuée par l'utilisateur aux 2 indicateurs : SUPTAUTO et SUPINCOMP.

La fonction LISP qui teste l'incompatibilité d'un composant de substitution x . τ appartenant à F(L) avec les substitutions des liens des autres littéraux à 4 arguments : x, τ, L, L', L' étant le premier littéral de la chaîne. Cette fonction s'écrit :

```
DEFINE2 ((
(INCOMPSUB
    (LAMBDA (VAR  TERME1  LT  CURLIT) (PROG (TERME2  LNS)

(WHILE  CURLIT
    (COND [ (OR (ALIT  CURLIT) ( EQ   CURLIT  LT) ) (GO  LITSUIV) ])
    (SETQ  LNS  (LIENS  CURLIT) )
    (REPEAT     (SETCDR  LNS)
          (COND [ (NOT (RES (CAR  LNS) )) NIL ]
                [ (OR (NULL (SETQ  TERME2  (GET2 (SUB (CAR  LNS) )) VAR) ))
                      (NOT (EQ (UNIFTERM  TERME1  TERME2)  INCOMP) ))
                  (GO  LITSUIV) ] ))
    (RETURN T)
LITSUIV
     (SETSUIV  CURLIT) )
 ))) ))
```

GET2 est une fonction utilitaire récursive qui ressemble à la fonction classique GET de LISP : GET2 [(x τ_1 z x y z ....) ; y] = τ_1

INCOMPSUB est appelée par la fonction SUPINCOMPLNS qui supprime les liens incompatibles.

La première version du démonstrateur était composée, d'une part, de fonctions sous forme compilées, d'autre part (et c'est le cas de la plus grande partie des fonctions), de fonctions sous forme de lambda-expressions, c'est-à-dire sous la forme utilisée ci-dessus. Cette version occupait approximativement un volume mémoire de 50 K ; elle utilisait la méthode de déduction des S-L graphes ; on avait la possibilité d'interdire, par l'intermédiaire de variables booléennes globales, la factorisation non basique, la suppression des liens incompatibles, la suppression des liens tautologiques.

La deuxième version du démonstrateur fonctionne avec la totalité des fonctions sous forme compilées ; elle occupe environ 28 K octets de mémoire (le programme source contient environ 1400 cartes). Elle présente plusieurs différences avec la 1ère version, la principale étant qu'elle utilise la méthode des S-L graphes simplifiée. Nous allons détailler ces différences.

VIII.6 - Différences entre les deux versions du démonstrateur

VIII.6.1 - Structure des données

Dans la liste d'identification d'une chaîne (cf. VIII.3), l'indicateur MORT est supprimé. Ce dernier était surtout utilisé par les fonctions de simplification du S-L graphe qui ont été supprimées du système car elles étaient non rentables. La structure d'un LIEN est modifiée ainsi ; la partie SUBSTITUTION du lien est remplacée par un booléen dans le cas d'un lien d'admissibilité ou d'un lien facteur ; le marqueur * LIEN * pourrait en outre être supprimé, ce qui porterait de 3 à 2 le nombre de doublets utilisés pour un lien (* LIEN * est utilisé dans la phase de mise au point du programme pour couper l'impression de listes circulaires).

Dans la liste d'identification d'un littéral P, un doublet est économisé en regroupant les informations relatives au statut de P (A ou B et celles relatives à son signe, dans un entier relatif STATSIGNE : le signe de P est celui de STATSIGNE, P est un B-littéral si | STATSIGNE | = 1, P est un A-littéral si | STATSIGNE | = 2.

VIII.6.2 - Algorithmes

Toutes les opérations (sauf la post-factorisation) opèrent sur le littéral le plus à gauche d'une chaîne : ce qui simplifie la mise en oeuvre de ces opérations ; en particulier, la fonction de sélection d'un B-littéral (SELECT) est supprimée ; la fonction de recopiage d'une chaîne (COPYLITS) est plus simple car il n'est plus nécessaire de lui fournir en paramètre le littéral de la chaîne à ne pas recopier (ou dont le statut doit changer). De plus, le fait que tous les liens soient unidirectionnels allège le travail de recopie d'un lien.

La partie de l'opération de e-réduction d'une chaîne C, qui correspond à la résolution de C avec des chaînes unitaires non d'entrées, n'est pas effectuée en utilisant des liens résolvants de C ; elle est effectuée en recherchant les chaînes unitaires adéquates dans le S-L graphe simplifié * ; cette recherche est facilitée par le fait que toutes les chaînes unitaires construites au cours de la démonstration sont rangées dans une liste spéciale (TAB1J). Cette liste est utilisée d'autre part (en conjonction avec TAB(1,0)), par le test de subsomption unitaire (cf. V.3, X.2).

Un test de subsomption double peut être également effectué ; cependant, pour ne pas trop alourdir les algorithmes, on se limite à ne considérer comme chaînes subsumantes possibles que celles de TAB(2,0). Ces deux tests peuvent être supprimés très facilement ; on utilise pour cela la propriété suivante de LISP : lorsqu'on lit une fonction LISP ayant le même nom qu'une fonction déjà lue, c'est la fonction lue en dernier qui est prise en compte par LISP. Il suffit donc, pour supprimer les tests de subsomption, de lire deux fonctions (de noms UNITSUBS et DBLESUBS) qui sont inopérantes.

Il est possible de dialoguer avec le système pendant une démonstration, si l'indicateur booléen DIALOGUE est à VRAI : dans ce cas, lorsque SIGMA entreprend de remplir une nouvelle diagonale de TAB, SIGMA imprime un

---

* Une autre solution aurait consisté à construire des liens résolvants bidirectionnels pour les seules chaînes unitaires ; nous avons préféré opter pour une solution qui entraînait une structure de données identique pour toutes les sortes de chaînes.

message au télétype et est prêt à imprimer ou modifier les divers paramètres du système, y compris le S-L graphe simplifié.

Les chapitres suivants sont consacrés à l'expérimentation et aux extensions que l'on peut envisager d'effectuer sur le système.

IX - EXPERIMENTATION DU SYSTEME

Nous présentons d'abord les exemples soumis au système ; nous parlons ensuite des paramètres de contrôle de la démonstration et de la possibilité d'extension de l'espace liste ; puis nous illustrons par quelques tableaux des remarques que nous avons faites. Enfin, nous présentons dans un tableau récapitulatif les meilleures preuves obtenues par le système.

IX.1 - Liste des exemples testés

Les exemples que nous avons choisis ont été, pour la plupart, empruntés à la litterature (ROBINSON 1965, REBOH et al. 1972, KOWALSKI 1973, LOVELAND 1969, FLETSIG et al. 1974, LUCKHAM 1968 b, WOS et al. 1965, CHANG 1970,1972). Ils appartiennent à divers domaines : algèbre, théorie des nombres, manipulation d'ensembles, preuves de programmes, calcul des prédicats du 1er ordre, analyse syntaxique.

Nous donnons pour chaque exemple deux formulations : une en français, une autre au moyen d'un ensemble de clauses. Les noms attribués aux exemples sont, en général, destinés uniquement à évoquer le domaine auquel ils se rapportent ; ils ne doivent pas être interprétés à la lettre : par exemple, dans GROUP1, l'ensemble dont il est question n'est pas nécessairement un groupe.

Dans tous les exemples, x, y, z, u, v, w, m, désignent des variables (quantifiées universellement) ; + Q(x) - R(y) est mis pour $Q(x) \vee \neg R(y)$; nous avons donné un nom à certaines clauses courantes afin d'éviter de les reproduire à chaque fois. Rappelons que $- P_1 - P_2 \ldots - P_k + Q_1 \ldots + Q_p$ peut se lire $(P_1 \wedge P_2 \wedge \cdots \wedge P_k) \supset (Q_1 \vee Q_2 \vee \cdots \vee Q_p)$.

a) Algebre

Dans ces exemples, P(x y z) est le prédicat x * y = z

Exemple n° 1 - GROUP1

"Tout ensemble muni d'une loi de composition associative $*$, d'un élément neutre e pour cette loi, et tel que le carré de tout élément soit e , est commutatif".

```
        1   + P(x e x)                    e  est élément neutre à droite ,
        2   + P(e x x)                    et à gauche ;
ASSOC1 : 3  + P(u z w) - P(x v w) - P(y z v) - P(x y u)   la loi *
ASSOC2 : 4  + P(x v w) - P(u z w) - P(y z v) - P(x y u)   est associative ;
        5   + P(x x e)                    hypothèse : x * x = e
        6   + P(a b c)                    négation du théorème ,
        7   - P(b a c)                    pour les éléments a et b.
```

Exemple n° 2 - GROUP2

"Un ensemble, muni d'une loi de composition $*$, ne peut posséder plus d'un élément neutre pour cette loi'.

f(x y) désigne le résultat du produit x * y ;
E(x y) désigne le prédicat x = y

```
        1   + P(x y f(x y))    *  est une loi de composition ,
        2   + P(a x x)
        3   + P(x a x)         a  est élément neutre ;
        4   + P(b x x)
        5   + P(x b x)         b  est élément neutre ;
        6   - E(a b)           négation du théorème : a ≠ b ;
EGAL1 : 7   + E(x x)                          l'égalité est réflexive
EGAL2 : 8   - E(x y) + E(y x)                             symétrique
EGAL3 : 9   - E(x y) - E(y z) + E(x z)                    et transitive ;
       10   - P(x y z) - P(x y u) + E(z u)       * est univoque.
```

Exemple n° 3 - GROUP3

"Tout ensemble, muni d'une loi de composition associative *, tel que les équations a * x = b et x * a = b possèdent une solution, admet un élément neutre".

h(x y) désigne la solution de la 1ère équation
g(x y) désigne la solution de la 2ème équation
j(x) désigne l'elément neutre correspondant à x (on ignore le théorème de l'exemple précédent!)

1 + P(g(x y) x y) * admet une solution à gauche
2 + P(x h(x y) y) * admet une solution à droite
3 + P(x y f(x y))
4 ASSOC1
5 ASSOC2
6 - P(j(x) x j(x)) négation du théorème.

Exemple n° 4 - GROUP4

"Si un ensemble, muni d'une loi de composition associative * avec inverse à gauche, possède un élément neutre à gauche e, e est aussi élément neutre à droite".

f(x) désigne l'inverse à gauche de x ;

1 + P(f(x) x e) x possède un inverse à gauche
2 + P(e x x) e est élément neutre à gauche
3 ASSOC1
4 ASSOC2
5 - P(a e a) négation du théorème.

Exemple n° 5 - GROUP4 bis

"Si un ensemble, muni d'une loi de composition associative * avec inverse à droite, possède un élément neutre à droite e, e est aussi élément neutre à gauche".

g(x) désigne l'inverse à droite de x ;

```
1   + P(x g(x) e)       x  possède un inverse à droite
2   + P(x e x)          e  est élément neutre à droite
3   ASSOC1
4   ASSOC2
5   - P(e a a)          negation du théorème.
```

Exemple n° 6 - GROUP5

"Si un ensemble est muni d'une loi de composition associative * avec inverse à droite et élément neutre à droite, tout élément de l'ensemble possède un inverse à gauche".

```
1   + P(x g(x) e)
2   + P(x e x)
3   ASSOC1
4   ASSOC2
5   - P(x a e)          négation du théorème.
```

Exemple n° 7 - GROUP6

"Soient G un groupe, S un sous-ensemble non vide de G ; si S est tel que pour tout couple (x,y) de S^2 , $x * y^{-1}$ appartienne à S, l'élément neutre e de G appartient à S".

g(x) désigne l'inverse x^{-1} de x ;

S(x) désigne le prédicat x ∈ S ;

```
1   + P(e x x)
2   + P(x e x)
3   + P(x g(x) e)
4   + P(g(x) x e)
5   + P(x y f(x y))
6   ASSOC1
7   ASSOC2
```

8	EGAL1	
9	EGAL2	
10	EGAL3	
11	- P(x y z) - P(x y u) + E(z u)	propriétés
12	- E(x y) - P(z u x) + P(z u y)	de
13	- E(x y) - P(z x u) + P(z y u	l'égalité
14	- E(x y) - P(x z u) + P(y z u)	par rapport
15	- E(x y) + E(f(z x) f(z y))	à
16	- E(x y) + E(f(x z) f(y z))	l'opération
17	- E(x y) + E(g(x) g(y))	*
18	- S(x) - E(x y) + S(y)	et par rapport à S
19	- S(x) - S(y) - P(x g(y) z) + S(z)	$x * y^{-1}$ S
20	+ S(a)	S est non vide
21	- S(e)	négation du théorème.

Exemple n° 8 - GROUP7

"Soient G un groupe, S un sous-ensemble non vide de G ; si S est tel que pour tout couple (x,y) de S^2 , $x * y^{-1}$ appartienne à S, tout élément z de S a son inverse z^{-1} dans S".

1	+ P(e x x)	
2	+ P(x e x)	
3	+ P(x g(x) e)	
4	+ P(g(x) x e)	
5	ASSOC1	
6	ASSOC2	
7	- S(x) - S(y) - P(x g(y) z) + S(z)	
8	+ S(a)	négation
9	- S(g(a))	du théorème.

Exemple n° 9 - ANNEAU

"Si un ensemble E est muni de deux lois de composition, + et *, possédant les propriétés suivantes :

. la loi + est associative, a un élément neutre 0 à gauche et un inverse à gauche,
. la loi * est distributive à gauche par rapport à la loi + ,
alors, quel que soit x appartenant à E, on a : x * 0 = 0".

Q(x y z) désigne le prédicat x + y = z ; g(x) désigne l'opposé de x ; d désigne 0.

1 + Q(d x x)
2 + Q(g(x) x d)
3 ASSOC1 (correspondant au prédicat Q)
4 ASSOC2 (correspondant au prédicat Q)
5 + P(x y f(x y))
6 + Q(u m w) - P(x v w) - Q(y z v) - P(x z m) - P(x y u) distributivité
7 + P(x v w) - Q(u m w) - Q(y z v) - P(x z m) - P(x y u) de * / +
8 - P(a d d) négation du théorème.

Exemple n° 10 - CORPS

"Dans un corps ordonné, x > 0 implique $x^{-1} > 0$".

Les clauses qui suivent ne sont relatives qu'à une partie des propriétés d'un corps ordonné ; le théorème est donc valable pour un ensemble plus général qu'un corps ordonné.

S(x) désigne le prédicat x > 0 ; d désigne 0 ; e désigne 1 ; g(x) désigne l'opposé de x ; f(x) désigne l'inverse de x ;

1 + P(x e x)
2 - P(e e d) $1 \neq 0$
3 + P(x f(x) e) + P(x x d) tout élément $\neq 0$ admet un inverse à droite
4 - P(x y u) + P(y x u) * est commutative
5 - P(x y u) + P(g(x) g(y) u) x * y = - x * -y
6 - P(x y u) + P(x g(y) g(u)) x * -y = - (x * y)
7 - P(x y u) + P(u u d) - P(x x d) 0 * y = 0
8 - P(x x d) - S(x) (x = 0) (x non positif)

9	+ S(x) + P(x x d) + S(g(x))	$(x \neq 0) \supset$ (x ou -x est positif)
10	- S(x) - S(y) - P(x y u) + S(u)	(x et y sont positifs) $\supset$ (x * y positif)
11	+ S(a)	négation
12	- S(f(a))	du théorème

b) Théorie des nombres

Exemple n° 11 - NUM1

"Etant donné un nombre premier a, si les entiers b et c sont dans la relation $a * c^2 = b^2$ avec a, alors a divise b".

Q(x y z) désigne le predicat $x * y = z$,
P(x) désigne le prédicat x est premier ,
D(x y) désigne le predicat x divise y ($x \mid y$) ,
s(x) désigne l'entier x^2

1	+ Q(x x s(x))	définition de x^2
2	+ Q(a s(c) s(b))	hypothèses : $a * c^2 = b^2$
		a est premier
3	+ P(a)	
4	- P(x) - Q(y z w) - D(x w) + D(x y) + D(x z)	si x premier divise y * z, alors x divise y ou z
5	- Q(x y z) + D(x z)	définition de |
6	- Q(x y z) + Q(y x z)	
7	- D(a b)	négation du théorème

Exemple n° 12 - NUM2

"Si a est premier, $\sqrt{a}$ est irrationnel".

f(x y) désigne le produit x * y
h(x y) désigne le quotient de y par x, lorsque x divise y

	1	+ Q(x y f(x y))	
	2	- P(x) - Q(y y z) -D(x z) + D(x y)	
	3	- Q(x y z) + Q(y x z)	
	4	ASSOC2 (correspondant au prédicat Q)	
	5	- Q(x y z) + D(x z)	définition
	6	- D(x y) + Q(x h(x y) y)	de |
DIV1 :	7	- D(x y) - D(y z) + D(x z)	transitivité de |
	8	EGAL2	
	9	- D(x y) - E(z y) + D(x z)	
	10	- Q(x y u) - Q(x z u) + E(y z)	régularité à droite de *
	11	+ P(a)	hypothèse : a est premier
	12	+ Q(b b d)	négation
	13	+ Q(c c g)	du
	14	+ Q(a g d)	
	15	- D(x b) - D(x c)	théorème

Exemple n° 13 - PRIM1

"Tout nombre supérieur à 1, possede un diviseur premier".

L(x y) désigne le prédicat x < y ,
g(x) désigne un diviseur de x, différent de 1 et de x, lorsque x n'est pas premier , a désignant le plus petit entier, supérieur a 1, qui ne possèderait pas de diviseur premier,
f(x) désigne un diviseur premier de x, lorsque 1 < x < a ;

DIV2 :	1	+ D(x x)	réflexivité de |
	2	DIV1	
	3	+ P(x) + D(g(x) x)	si x est non premier,
	4	+ P(x) + L(1 g(x))	il possède un diviseur g(x)
	5	+ P(x) + L(g(x) x)	tel que 1 < g(x) < x
	6	+ L(1 a)	négation du théorème :
	7	- P (x)- D(x a)	clauses 6 - 9
	8	- L(1 x) - L(x a) + P(f(x))	
	9	- L(1 x) - L(x a) + D(f(x) x)	

Exemple n° 14 - PRIM2

"Il existe une infinité de nombres premiers".

f(x) désigne l'entier x ! + 1 ; h(x) désigne un diviseur premier de x (cf. PRIM1), lorsque x n'est pas premier ;

1	- L(x x)	< n'est pas réflexive
2	- L(x y) - L(y x)	< n'est pas symétrique
3	DIV1	
4	DIV2	
5	- D(x y) - L(y x)	x | y x ≤ y
6	+ P(x) + D(h(x) x)	théorème
7	+ P(x) + P(h(x))	de l'exemple PRIM1
8	+ P(x) + L(h(x) x)	appliqué à un nombre non premier
9	+ L(x f(x))	x < x ! + 1
10	+ L(y x) - D(x f(y))	x | (y ! + 1) ⊃ x > y
11	+ P(a)	négation
12	+ L(f(a) x) - L(a x) - P(x)	du théorème

c) Manipulations d'ensembles

Exemple n° 15 - HAS-PARTS1

"Montrer que John a deux mains".

IN(x y) désigne le prédicat : X appartient à l'ensemble Y ,
HP(X U Y) désigne le prédicat : X contient U objets de type Y ;

1	- IN(X BOY) + IN(X HUMAN)	tout garçon est un humain
2	- IN(X HUMAN) + HP(X 2 ARM)	tout humain a 2 bras
3	- IN(X ARM) + HP(X 1 HAND)	tout bras a 1 main
4	- HP(X U Y) - HP(F(V U Z Y X) V Z) + HP(X T(U V) Z)	
5	- HP(X U Y + IN(F(V U Z Y X) Y) + HP(X T(U V) Z)	
6	+ IN (JOHN BOY)	John est un garçon
7	- HP (JOHN T(2 1) HAND)	négation du théorème.

Les clauses 4 et 5 signifient : si X contient U objets de type Y et si Y contient V objets de type Z, alors X contient U * V objets de type Z, F(V U Z Y X) désigne un objet de type Z, T(U V) désigne le produit U * V.

Exemple n° 16 - HAS-PARTS2

"Montrer que John a dix doigts".

1-6 cf. HAS-PARTS1
7 - IN (X HAND)) + HP(X 5 FINGERS) toute main a 5 doigts
8 - HP (JOHN T(T(2 1) 5) FINGERS) négation du théorème

d) Preuves de programmes

Exemple n° 17 - PROG

"Etant donné le programme suivant (calcul de 2^n) :

(P1) J := 0 ;
(P2) K := 1 ;
LOOP : (P3) if J = N then (P4) goto OUT ; (P5)
(P6) K := 2 * K ;
(P7) J := J + 1 ;
(P8) goto LOOP ;
OUT :

montrer que le point (P3) du programme est dans une boucle".

LABEL (X Y) désigne le prédicat : X étiquette l'instruction du point Y du programme ;
HAS (X Y) désigne le prédicat : le point X du programme contient l'instruction Y ;
FOLLOWS (Y X) désigne le prédicat : le point Y suit immédiatement (d'un point de vue statique) le point X du programme ;
SUCCEEDS (Y X) désigne le prédicat : le point Y peut être exécuté après le point X du programme ;
ASS (J N) désigne l'instruction d'affectation : J := N ;
IFTHEN (X Y) désigne l'instruction conditionnelle : si X alors Y ;

GOTO (X) désigne l'instruction de branchement : goto X ;
XPLUS (X Y) , XTIMES(X Y) , XEQUAL (X Y) désigne respectivement les expressions arithmétiques X + Y , X * Y , X = Y ;

```
1   + HAS (P1 ASS (J 0))
2   + FOLLOWS (P2 P1)
3   + HAS (P2 ASS (K 1))
4   + LABELS (LOOP P3)
5   + FOLLOWS (P3 P2)
6   + HAS (P3 IFTHEN (XEQUAL (J N) P4))          les clauses
7   + HAS (P4 GOTO (OUT))                        1 à 14
8   + FOLLOWS (P5 P4)                            décrivent
9   + FOLLOWS (P6 P3)                            le programme
10  + HAS (P6 ASS (K TIMES (2 K)))
11  + FOLLOWS (P7 P6)
12  + HAS (P7 ASS (J XPLUS (J 1)))
13  + FOLLOWS (P2 P7)
14  + HAS (P8 GOTO (LOOP))
15  - HAS (X GOTO(Z)) - LABELS (Z Y) + SUCCEEDS (Y X) sémantique
16  - HAS (X IFTHEN (Z Y) + SUCCEEDS (Y X)            des
17  - FOLLOWS (X Y) + SUCCEEDS (X Y)                  instructions
18  - SUCCEEDS (X Z) - SUCCEEDS (Z Y) + SUCCEEDS (X Y) transitivité de
                                                       SUCCEEDS
19  - SUCCEEDS (P3 P3)                    négation du théorème.
```

EXEMPLE n° 18 - MINIPROG

"Etant donné le programme suivant :

LOOP : (P3) if J = N then (P4) goto OUT ; (P5)
(P8) goto LOOP ;

montrer que le point (P3) du programme est dans une boucle".

Le programme de cet exemple est une partie du programme de l'exemple PROG ; la formalisation de ce problème est effectuée au moyen des clauses 4, 6-8, 14-19 de l'exemple précédent, complétées par la clause :

13 bis + FOLLOWS (P8 P3).

e) Théorème du calcul des prédicats du 1er ordre

Exemple n° 19 - FORM

"Montrer que la formule bien formée fermée suivante :

$\mathcal{A}$: $\exists x \exists y \forall z \; [Pxy \supset (Pxz \vee Qyz)] \wedge [Pxy \vee (Pzz \supset Qzz)] \supset (Qxy \vee Qzz)$

est un théorème (ou, ce qui est équivalent dans le CP1, une formule logiquement valide) du calcul des prédicats du 1er ordre".

La transformation sous forme clausale de $\neg \mathcal{A}$ conduit à l'ensemble de clauses suivant :

1 - P(x y) - P(x g(x y)) + Q(y g(x y))
2 - P(x y) + P(x g(x y)) - Q(y g(x y))
3 - P(x y) - P(g(x y) g(x y)) + Q(g(x y) g(x y))
4 + P(x y) + P(g(x y) g(x y))
5 + P(x y) - Q(g(x y) g(x y))
6 + Q(x y) + Q(g(x y) g(x y))
7 - Q(x y) - Q(g(x y) g(x y))

f) Théorie des langages

Exemple n° 20 - TRANS1, TRANS2, TRANS3, TRANS4

"Etant donné un système génératif comportant les regles de réécriture suivantes :

(1) X + Y → Y + X
(2) X + (Y+Z) → (X+Y) + Z
(3) (X+Y) - Y → X
(4) X → (X+Y) - Y
(5) (X-Y) + Z → (X+Z) - Y
(6) (X+Y) - Z → (X-Z) + Y

Montrer que :

a - (X+Y) + Z peut être transformé en X + (Y+Z)

b - (X-Y) + Z peut être transformé en X + (Z-Y)

c - X + (Y-Z) peut être transformé en (X-Z) + Y

d - (X+Y) - Z peut être transformé en X + (Y-Z)".

plus (x y) désigne l'expression x + y ;
minus (x y) désigne l'expression x - y ;
R(x y) désigne le prédicat x peut être transofrmé en y ;

1 + R (plus(x y) plus(y x))
2 + R (plus(x plus(y z)) plus (plus(x y) z))
3 + R (minus (plus(x y) y) x) les clauses 1-6 décrivent
4 + R (x minus (plus(x y) y)) les règles de réécriture
5 + R (plus (minus(x y) z) minus (plus(x z) y)) du système
6 + R (minus (plus(x y) z) plus (minus(x z) y)) génératif ;
7 + R(x x) les clauses 7-11
8 - R(x y) - R(y z) + R(x z) décrivent
9 - R(x y) - R(u plus(x v)) + R(u plus(y v)) le fonctionnement
10 - R(x y) - R(u minus(x v)) + R(u minus(y v)) du système
11 - R(x y) - R(u minus(v x)) + R(u minus(v y)) génératif ;
12.a - R(plus (plus(a b)c) plus (a plus(b c))) négation du théorème (TRANS1)
12.b - R(plus (minus(a b) c) plus (a minus(c b))) négation du théorème (TRANS2)
12.c - R(plus (a minus(b c)) plus (minus (a c)b)) négation du théorème (TRANS3)
12.d - R(minus (plus(a b) c) plus (a minus(b c))) négation du théorème (TRANS4)

Exemple n° 21 - LANG1, LANG2

"Soient V_T = {a, b, c} , $L_1 \subset V_T^*$ le langage défini par :

$x \in L_1 \iff x$ contient le même nombre de a, b et c ; soit L_2 le langage défini par la grammaire hors-contexte suivante :

S → A B C
A → a A | a
B → b B | b
C → c C | c

Montrer que $L_1 \wedge L_2$ est le langage (contexte-lié) : $\{a^n b^n c^n\}$, $n \geq 1$".

Ce problème se décompose naturellement en 2 parties :

(I) $\{a^n b^n c^n\}$, $n \geq 1 \subset L_1 \wedge L_2$

(II) $L_1 \wedge L_2 \subset \{a^n b^n c^n\}$, $n \geq 1$

Nous ne nous intéressons, ici, qu'à la 1ère partie ; la 2ème partie correspond au problème (indécidable en général) qui consiste à montrer qu'une f.b.f. donnée n'est pas un théorème du C.P. 1 ; cependant, dans ce cas particulier, le problème est probablement décidable (puisqu'il existe des algorithmes permettant de décider si un mot donné appartient ou non au langage $\{a^n b^n c^n\}$, $n \geq 1$).

Nous utilisons, pour la formalisation de cet exemple et pour celle de l'exemple n° 22, le mécanisme des systèmes -Q (COLMERAUER, 1970). Chaque symbole du mot à lire est encadré par des marqueurs numériques : par exemple, 1 a 2 a 3 b 4 b 5 c 6 c 7 correspond au mot $a^2 b^2 c^2$. Nous présentons ci-dessous deux formalisations pour l'exemple n° 21 : LANG1 et LANG2 ;
a(x y) (resp. b(x y), c(x y)) désigne le prédicat : le symbole a (resp. b, c) se trouve entre les marqueurs x et y du mot à lire ;
A(x y u) (resp. B(x y u) , C(x y u) désigne le prédicat : le sous-mot a^u (resp. b^u, c^u) se trouve entre les marqueurs x et y du mot à lire ; S(x y) désigne le prédicat : le sous-mot $a^n b^n c^n$, $n \geq 1$ se trouve entre les marqueurs x et y du mot à lire ; s(u) désigne le successeur u+1 de l'entier u ;
f(x,y,z) désigne le prédicat : le symbole z se trouve entre les marqueurs x et y du mot à lire ; F(x,y,z,u) désigne le prédicat : le sous-mot z^u se trouve entre les marqueurs x et y du mot à lire ;

LANG1 :

1	- a(x y) + A(x y s(o))	A → a
2	- a(x y) - A(y z u) + A(x z s(u)	A → a A
3	- b(x y) + B(x y s(o))	B → b
4	- b(x y) - B(y z u) + B(x z s(u))	B → b B
5	- c(x y) + C(x y s(o))	C → c
6	- c(x y) - C(y z u) + C(x z s(u))	C → c C
7	- A(x y u) - B(y z u) - C(z w u) + S(x w)	S → A B C
8	+ a(1 2) 9 + a(2 3)	le mot
10	+ b(3 4) 11 + b(4 5)	a lire
12	+ c(5 6) 13 + c(6 7)	est a^2 b^2 c^2
14	- S(1 7)	négation du théorème

LANG2 :

1	- f(x y v) + F(x y v s(o))	V → v
2	- f(x y v) - F(y z v u) + F(x z v s(u))	V → v V
3	- F(x y a u) - F(y z b u) - F(z w c u) S(x w)	S → A B C
4	+ f(1 2 a) 5 + f(2 3 a)	le mot
6	+ f(3 4 b) 7 + f(4 5 b)	a lire
8	+ f(5 6 c) 9 + f(6 7 c)	est a^2 b^2 c^2
10	- S(1 7)	négation du théorème.

Exemple n° 22 - LANG3

"Soit L le langage hors-contexte défini par la grammaire suivante :

PH → GN GV
GV → VB | VB GN | PR VB
GN → NOM | ART NOM ADJ | ART NOM
NOM → pilote | porte
VB → ferme | porte
ADJ → ferme
ART → le | la
PR → le

montrer que la phrase : "le pilote ferme la porte" appartient à L".

le (x y) (resp. pilote (x y), ferme (x y), la (x y), porte (x y), ART (x y), NOM (x y), VB (x y), ADJ (x y), PR (x y), GN (x y), GV (x y), PH (x y) désigne le prédicat : entre les marqueurs x et y de la phrase à lire se trouve le mot le (resp. le mot pilote, le mot ferme, le mot la, le mot porte, un article, un nom, un verbe, un adjectif, un pronom, un groupe nominal, un groupe verbal, une phrase de L).

1	- le (x y) + ART (x y)	ART → le
2	- pilote (x y) + NOM (x y)	NOM → pilote
3	- ferme (x y) + VB (x y)	VB → ferme
4	- ferme (x y) + ADJ (x y)	ADJ → ferme
5	- la (x y) + ART (x y)	ART → la
6	- la (x y) + PR(x y)	PR → la
7	- porte (x y) + NOM (x y)	NOM → porte
8	- porte (x y) + VB (x y)	VB → porte
9	- NOM (x y) + GN (x y)	GN → NOM
10	- ART (x y) - NOM (y z) - ADJ (z u) + GN (x u)	GN → ART NOM ADJ
11	- ART (x y) - NOM (y z) + GN (x z)	GN → ART NOM
12	- VB (x y) + GV (x y)	GV → VB
13	- VB (x y) - GN (y z) + GV (x z)	GV → VB GN
14	- PR (x y) - VB (y z) + GV (x z)	GV → PR VB
15	- GN (x y) - GV(y z) + PH (x z)	PH → GN GV
16	+ le (1 2) 17 + pilote (2 3)	la phrase
18	+ ferme (3 4) 19 + la (4 5)	à lire est :
20	+ porte (5 6)	"le pilote ferme la porte
21	- PH (1 6)	négation du théoreme.

IX.2 - Paramètre de contrôle

Dans le but de limiter la complexité des chaînes construites par le démonstrateur, nous avons été amenés à adjoindre de nouvelles restrictions à celles déjà incluses dans la méthode de déduction. Leur ampleur dépend des valeurs attribuées par l'utilisateur, de façon standard ou interactivement, à des paramètres de contrôle. Pour rendre inopérante l'une de ces restrictions, il suffit d'attribuer au paramètre correspondant, une valeur très grande. Nous décrivons ci-dessous ces restrictions.

1. Niveau maximum d'imbrication fonctionnelle : MAXFN

Ce paramètre permet de contrôler la complexité des termes contenus dans les chaînes lorsque l'on pense que cette complexité ne dépassera pas une certaine limite.

Exemple : Si MAXFN = 3 P [x f(g(h(a))))] sera rejeté ,
Q [f(h(a) g(b))] sera accepté ;

2. Nombre maximum de B-littéraux : MAXBLITS

Ce paramètre interdit la construction de chaînes dont la longueur (i.e. le nombre de B-littéraux) dépasse MAXBLITS ; lorsque ce paramètre est utilisé, il est possible de récupérer systématiquement et simplement de la place dans le tableau TAB ; en effet, supposons que l'on soit en train de remplir la diagonale numéro d du tableau (i.e. un point d'indice (i j) de d est tel que i + j = d) ; si d ≥ MAXBLITS + 2 (cf. figure IX.1), le niveau minimum des chaînes de d est d-MAXBLITS ; les chaînes des lignes 0 à (d-MAXBLITS-2) de TAB ne pourront donc pas être des ancêtres directs de chaînes de d ; elles peuvent donc être supprimées du tableau. La procédure COLLECT effectue ce travail dans

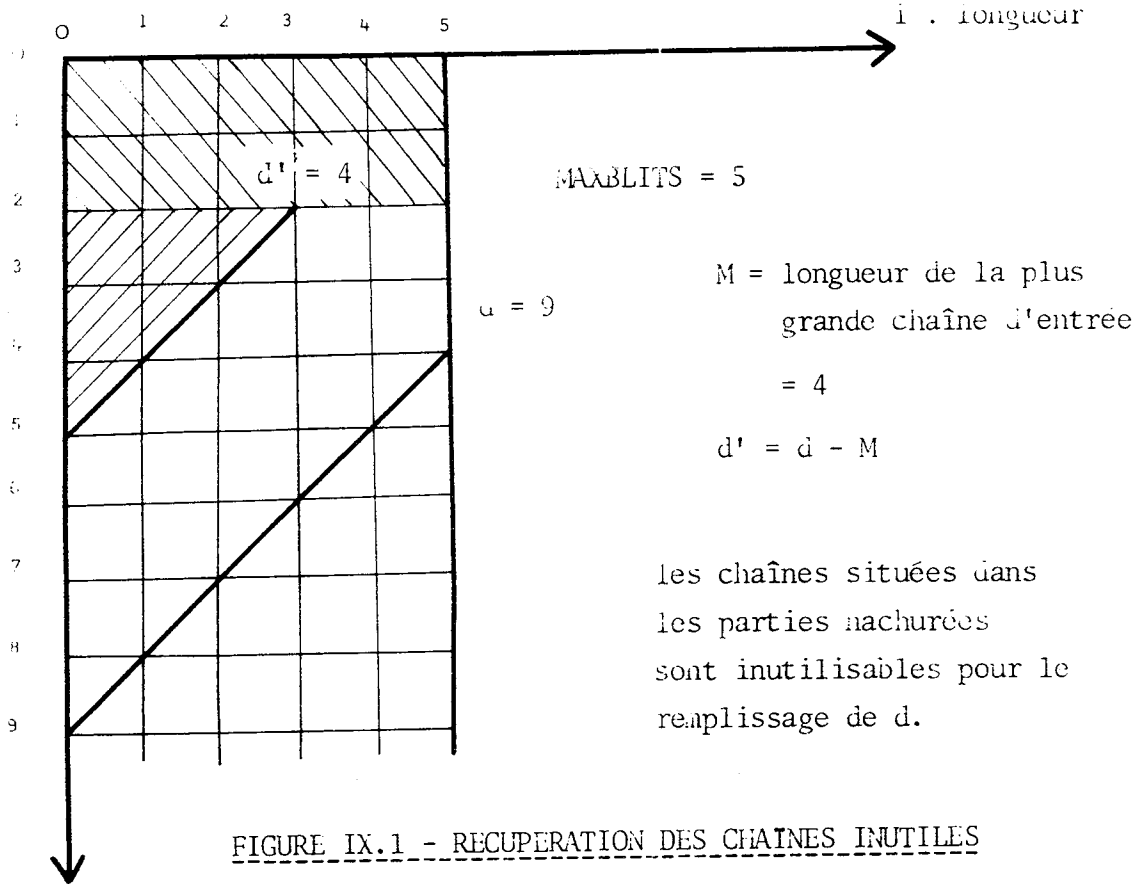


FIGURE IX.1 - RECUPERATION DES CHAINES INUTILES

le démonstrateur : à partir de la diagonale MAXBLITS+2, avant de procéder au remplissage d'une nouvelle diagonale d, COLLECT libère la place occupée dans TAB par les chaînes de la ligne d-MAXBLITS-2.

Remarque : Par des considérations analogues sur les longueurs, on pourrait perfectionner la procédure de récupération des chaînes inutiles de la façon suivante : soient M, la longueur de la plus grande chaîne d'entrée, d le numéro de la diagonale en cours de remplissage ; les chaînes situées sur la diagonale de numéro d' = d - M et à sa gauche, ne peuvent être ancêtres directs de chaînes de d.

3. Nombre maximum de A-littéraux : MAXALITS

Ce paramètre interdit la construction de chaînes contenant plus de MAXALITS A-littéraux ; le nombre m de A-littéraux et le niveau n d'une chaîne C sont liés au nombre p d'opérations d'extension nécessaires pour obtenir C, par les inégalités :

$$n \geq p \geq m$$

4. Subsomption

Il est possible d'effectuer un test de subsomption unitaire ou double en changeant dans le système les fonctions appropriées (cf. VIII.6.2).

5. Factorisation non basique

Dans une précédente version du démonstrateur, il était possible d'effectuer, le cas échéant, des factorisations non basiques selon la valeur booléenne de l'indicateur FACTS ; dans la version actuelle du démonstrateur, seules les factorisations basiques sont effectuées.

6. Liens incompatibles et tautologiques

Dans une précédente version du démonstrateur, il était possible d'effectuer ou non la suppression des liens incompatibles et tautologiques selon les valeurs booléennes respectives des indicateurs SUPTAUTO et SUPINCOMP (cf. VIII.5) ; cette possibilité est supprimée dans la version actuelle.

7. Trace

Outre les possibilités de trace des fonctions propres au système LISP, il est possible d'imprimer, au télétype ou sur une imprimante, toutes les chaînes construites par le démonstrateur en positionnant à VRAI l'indicateur TRACE.

Enfin, l'indicateur SORTIETAB permet d'imprimer éventuellement le tableau TAB des chaînes du S-L graphe simplifié à l'issue d'une démonstration.

IX.3 - Variabilité de l'espace liste

Comme nous l'avons dit précédemment (cf. VIII.8), la structure de donnée élémentaire de LISP est le doublet. L'ensemble des doublets utilisables constitue l'espace liste du système ; il est possible de modifier la taille de cet espace liste selon le problème.

Ce choix est délicat, car quand l'espace liste augmente, le programme de récupération de listes inutilisées (garbage collector) est appelé moins souvent, mais chaque appel dure plus longtemps. Le plus grand espace liste que nous ayons utilisé contenait environ 103 000 doublets (ce qui correspond à une machine virtuelle CP de 1024K OCTETS).

IX.4 - Influence de la formalisation

Les exemples LANG1 et LANG2 constituent deux formalisations d'un même problème. La seconde est plus concise que la première, mais conduit à des temps légèrement supérieurs, car la différenciation qu'il était possible de faire au niveau des noms des prédicats dans LANG1, doit être faite dans LANG2, au niveau des unifications de listes d'arguments.

Les exemples GROUP4 et GROUP4 bis peuvent être considérés comme deux formalisations du même exemple (l'exemple n° 4) ; il suffit de considérer que dans GROUP4 bis, P (x y z) désigne le prédicat y * x = z.

Dans l'exemple PRIM2, la formalisation du théorème repose sur une démonstration constructive de la suite infinie des nombres premiers (on montre que tout nombre premier a, possède un successeur b tel que $a < b \leq a ! + 1$) ; si on remplace la négation du théorème par la clause : - L(a x) - P(x), (ce qui est plus naturel), on obtient une preuve en 0,2 sec. (au lieu de 1,2).

Dans l'exemple NUM2, la grande complexité de l'exemple nous a conduit à modifier la formalisation de la façon suivante :

la clause 6 : - D (x y) + Q (x h (x y) y) est remplacée par :
6': - D (z x y) + Q (h (x y) x QUOTIENT y)

toutes les autres clauses sont modifiées en rajoutant dans chaque littéral de la clause un argument qui est une variable supplémentaire. Exemple :

la clause 9 : - D (x y) - E (z y) + D (x z) est remplacée par
9': - D (u x y) - E (u z y) + D (u x z).

Donc, à tout prédicat P_i^n , on fait correspondre un prédicat P'^{n+1}_i avec la signification suivante :

$$P'^{n+1}_i(x,t_1,\ldots,t_n) = P^n_i(t_1,\ldots,t_n)$$
$$P'^{n+1}_i(t_o,t_1,\ldots,t_n) = P^n_i(\tilde{t}_1,\ldots,\tilde{t}_n)$$

$\tilde{t}_i$ représente le terme obtenu en substituant dans t_i, le symbole QUOTIENT par le terme t_o (si t_o n'est pas une variable).

Le but de cet artifice est de diminuer, de façon sélective, le niveau d'imbrication fonctionnel d'une classe de termes (ceux qui contenaient le symbole fonctionnel h) : par exemple, f (h (x y), a) devient f (QUOTIENT a).

Cet artifice utilisé en conjonction avec MAXFN a permis de trouver une preuve pour cet exemple, alors que l'utilisation de MAXFN seule ne le permettait pas.

IX.5 - Ordre des clauses et des littéraux

L'ordre des clauses a une influence relativement faible sur le temps mis pour trouver une preuve (ceci est dû au choix de la stratégie diagonale montante) ; par contre, l'ordre des littéraux a de l'importance, comme le montrent les résultats suivants des exemples FORM et PRIM1 (2' et 3' représentent les clauses 2 et 3 dans l'ordre inverse) :

Clauses utilisées	CHAINES ESSAYEES	CHAINES ACCEPTEES	CHAINES INUTILES	CHAINES UNITAIRES	
1 , 2 , 3	71	31	18	2	FORM
1 , 2 , 3'	94	39	26	2	
1 , 2' , 3'	94	40	27	2	
3		57	43		PRIM1
3'		127	113		

Figure IX.2 : Influence de l'ordre des littéraux dans les exemples FORM et PRIM1

IX.6 - Utilisation de la subsomption

Dans la plupart des cas, la subsomption unitaire est suffisante ; cependant la subsomption double peut être utile dans certains cas ; elle est plus coûteuse que la 1ère, aussi il n'est pas rare d'obtenir, avec la subsomption double, des preuves plus coûteuses en temps que d'autres preuves qui nécessitent pourtant la construction de plus de chaînes.

SUBSOMPTION UNITAIRE	SUBSOMPTION DOUBLE	CHAINES ESSAYEES	CHAINES ACCEPTEES	CHAINES INUTILES	CHAINES UNITAIRES	TEMPS (sec)
N	N	102	56	43	5	11,7
O	N	96	43	30	2	9,5
N	O	73	36	23	3	10,8
O	O	11	31	18	2	9,6

Figure IX.3 : Utilisation des subsomptions unitaire et double dans l'exemple FORM

IX.7 - Choix de différents supports

Nous avons vérifié l'importance déterminante du choix d'un bon support, pour l'obtention d'une preuve de coût intéressant. En général, les clauses issues de la négation du théorème conviennent parfaitement, dans certains cas (par exemple GROUP1), les hypothèses particulières du problème conviennent mieux. Notons enfin que le choix du support peut refléter le sens choisi pour la recherche d'une preuve : dans le cas de l'analyse syntaxique (par exemple LANG3), selon le choix du support, on peut obtenir une analyse ascendante, descendante ou mixte, de la phrase.

SUPPORT	TEMPS (sec.)	CHAINES ACCEPTEES
{+ P(a b c)}	27,3	89
{+ P(x x e)}	2,7	24
{- P(b a c)}	dépassement (>40)	
{+ P(a b c) , - P(b a c)}	23,6	76
{- P(b a c) , + P(x x e)}	7,0	33
{+ P(a b c) , - P(b a c) , + P(x x e)}	5,4	29
ASSOC1	22,2	67
{+ P(a b c) , + P(x x e)}	2,7	22

Figure IX.4 : Résultats de GROUP1 avec différents supports

Nous présentons maintenant un tableau regroupant les meilleurs résultats. Nous avons donné la préférence aux preuves qui nécessitent la construction d'un minimum de chaînes. La différence entre les nombres de chaînes "ESSAYEES" et "ACCEPTEES" correspond à des chaînes pour lesquelles le test d'admissibilité est positif. NIV est le niveau des preuves obtenues ; MAXB et MAXA correspondent aux valeurs respectives de MAXBLITS et MAXALITS ; UNI et DBL correspondent respectivement à la subsomption unitaire et double.

EXEMPLES	SUPPORT	NIV	MAXFN	MAXB	MAXA	UNI/DBL	CHAINES ESSAYEES/ACCEPTEES/ INUTILES/UNITAIRES	TEMPS (sec.)
GROUP1	5	7		3	2	O / N	20/18/8/4	1,9
GROUP2	4,5,6	3	1	2	1	O / N	6/4/1/2	0,19
GROUP3	1	4	4	3	1	N / N	5/5/1/2	0,4
GROUP4	5	10	3	3	3	N / N	30/29/19/6	2,6
GROUP4 bis	5	10	3	5	2	N / N	61/49/39/1	7,4
GROUP5	5	7	3	5	2	N / N	19/16/9/1	1,9
GROUP6	21	4	3	10	10	O / O	7/7/3/3	0,4
GROUP7	9	7	2	5	2	O / N	24/22/15/6	1,9
ANNEAU	5	8	3	4	2	O / N	218/107/84/8	27,0
CORPS	12	18	3	4	6	O / O	686/263/245/22	100,0
NUM1	7	6	3	4	5	O / N	10/9/3/1	0,5
NUM2	11	11	2	3	5	O / N	942/451/432/123	176,0
PRIM1	3	14	3	4	4	O / N	85/57/43/2	6,6

Figure IX.5 : Tableau récapitulatif des meilleures preuves (1ère partie)

EXEMPLES	SUPPORT	NIV	MAXFN	MAXB	MAXA	UNI/DBL	CHAINES ESSAYEES/ACCEPTEES INUTILES/UNITAIRES	TEMPS (sec.)
PRIM2	12	5	3	4	4	O / O	23/15/5/8	1,1
HAS-PARTS1	7	6	3	4	4	N / N	10/10/0/3	0,4
HAS-PARTS2	8	13	3	3	6	N / N	31/29/5/3	2,7
PROG	19	12	2	3	4	N / O	57/25/13/6	3,5
MINIPROG	19	6	2	3	4	N / N	14/7/1/2	0,6
FORM	1	15	4	4	4	O / O	71/31/18/2	9,5
TRANS1	12 a	9	3	2	4	O / N	26/12/3/4	1,8
TRANS2	12 b	7	3	3	2	O / N	64/22/15/4	3,9
TRANS3	12 c	7	3	3	3	O / N	64/22/15/3	4,4
TRANS4	12 d	13	3	3	2	O / N	54/19/12/4	3,4
LANG1	14	13	10	10	10	O / N	17/15/2/3	0,9
LANG2	10	6	10	10	10	O / N	17/15/2/3	1,2
LANG3	16-20,21	5		3	2	O / N	70/43/29/18	2,7

Figure IX.5 - Tableau récapitulatif des meilleures preuves (2ème partie)

X - EXTENSIONS ET MODIFICATIONS DU SYSTEME

X.1 - Prise en compte des A-littéraux

Une première modification du système nous est suggérée par une remarque de LOVELAND (1969) concernant la procédure de Model-elimination : lorsque la stratégie de recherche doit choisir entre plusieurs chaînes d'égal mérite, le choix conseillé est celui d'une chaîne comportant un maximum de A-littéraux ; en effet, une telle chaîne est sensée offrir un maximum de possibilités d'application ultérieure de l'opération de e-réduction, et donc, de suppression de ses B-littéraux ; la chaîne vide a donc des chances d'être obtenue plus rapidement avec cette chaîne.

On pourrait encore raffiner ce critère en attribuant à tout A-littéral d'une chaîne, un poids **p** égal au nombre de B-littéraux plus récents se trouvant à sa gauche dans la chaîne, le critère devenant alors le suivant : choisir une chaîne parmi celles telles que $\sum_{i=1}^{k} p_i$ soit maximum , $p_1 \ldots p_k$ étant les poids des k A-littéraux de la chaîne. Ces critères peuvent être incorporés dans le calcul de la fonction heuristique, ce qui a pour conséquence de définir un mérite plus fin. On peut inclure cette extension dans le système en ajoutant respectivement aux listes d'identification des littéraux et des chaînes les valeurs p_i et Σp_i ; ces valeurs seraient mises à jour lors d'une opération d'extension, e-réduction, de tronquation ou de post-factorisation.

X.2 - Test de subsomption (cf. V.3)

L'application du principe de subsomption permet de simplifier le S-L graphe ; cependant, elle doit être effectuée avec précaution, car elle peut rendre la méthode de déduction incomplète. KOWALSKI et KUEHNER (1970) proposent une définition plus restrictive de la subsomption pour les chaînes:

Une chaîne C_1 SL-subsume une chaîne C_2 si une instance $C_1\sigma$ de C_1 est une sous-chaîne suffixe d'une chaîne équivalente à C_2 (Cf. V.11).

Exemple -

P(x) [Q(x)] SL-subsume R(a) P(a) [Q(a)] et P(a) R(a) [Q(a)] mais pas R(a) [Q(a)] P(a).

La stratégie Σ peut être transfonnée en une stratégie Σ' telle que :

1 - Σ et Σ' engendrent, le cas échéant, la même première réfutation.

2 - Si Σ engendre une dérivation D, Σ' engendre D pourvu que la sous dérivation immédiate de D ait été engendrée par Σ', et n'ait pas été supprimée.

3 - Soit D une dérivation d'une chaîne C par Σ'.

a) Si C est SL-subsumée par une chaîne dont la dérivation a été engendrée par Σ' et non supprimée, D est supprimée.

b) Sinon toute dérivation, antérieurement engendrée par Σ', d'une chaîne subsumée par Σ' est supprimée.

L'incorporation du test de SL-subsomption dans notre système pourrait être facilitée par la création d'un nouveau type de lien : le lien de subsomption ; ce lien serait bi-directionnel, relierait deux occurrences de B-littéraux potentiellement identiques (mod. θ) appartenant à des chaînes différentes et serait étiqueté par θ. Le test de SL-subsomption étant assez coûteux, on peut se limiter à la SL-subsomption unitaire (i.e. cas où la chaîne subsumante ne contient qu'un seul B-littéral) et/ou à la SL-subsomption double (i.e. la chaîne subsumante ne contient que deux B-littéraux) qui constituent des compromis entre l'absence de test et le test complet.

X.3 - Génération de lemmes

Cela correspond, d'une certaine manière, à un processus d'apprentissage du démonstrateur ; en effet, la génération de lemmes procède d'une volonté de tirer parti de l'expérience acquise par le démonstrateur dans la construction de dérivations.

Exemples -

- Supposons que nous ayons dérivé successivement les chaînes suivantes :

D1 : QR [P] N , S [R] Q [P] N , ¬R [S] [R] Q [P] N , Q [P] N

le fait que R puisse être éliminé, quel que soit le "contexte" des littéraux qui le suivent, peut être enregistré par la création du lemme ¬R pouvant être utilisé comme chaîne d'entrée dans la suite, lors d'une opération d'extension.

- Si, au lieu de D1, nous avions dérivé successivement les chaînes suivantes :

D2 : QR [P] N , S [R] Q [P] N , ¬P [S] [R] Q [P] N , Q [P] N

En construisant D2, le démonstrateur a "appris" que R peut être éliminé pourvu que [P] figure dans le contexte des littéraux qui le suivent ; ce fait peut être enregistré par la création du lemme ¬R ¬P. LOVELAND (1969) a défini un mécanisme de construction des lemmes, qui peut s'ajouter aux opérations d'extension, e-réduction, tronquation et post-factorisation ; dans ce mécanisme, on attache à chaque occurrence L de littéral du système un nombre entier : sa portée. A chaque suppression d'un A-littéral L par tronquation, correspond une possibilité de création d'un lemme contenant un B-littéral complémentaire de L, la portée d'un littéral P servant à déterminer si P a été utilisé au cours de l'élimination de L (et si, par conséquent, P ou son complément, doit figurer dans le lemme produit).

La portée peut facilement être rajoutée à la liste d'identification d'un littéral ; le mécanisme de génération de lemmes peut s'incorporer aisément dans le système ; cependant, il semblerait raisonnable de se limiter à la génération des lemmes unitaires.

X.4 - Raffinement de la fonction heuristique

Nous avons vu en X.1 que la prise en compte des A-littéraux des chaînes pouvait donner lieu à un premier raffinement de la fonction heuristique. Nous avons indiqué, en VII.4.4, le principe d'un second raffinement fondé sur l'exploration du S-L graphe et la prévoyance ("look-ahead") des opérations à effectuer ; c'est cette idée (KOWALSKI, 1973) que nous détaillons maintenant.

La connectivité de niveau n d'une occurrence d'un B-littéral L quelconque est une estimation minimum, notée $F_{\sim n}(L)$, du nombre de liens résolvants à supprimer dans le S-L graphe G pour éliminer L, cette estimation étant effectuée en envisageant des chemins du S-L graphes de longueur n, au maximum.

La valeur $F_{\sim 0}(L)$ correspond à une estimation minimum qui ne tient aucun compte du graphe ; on a donc, nécessairement : $F_{\sim 0}(L_i) = 1$, $\forall L_i \in G$. Les valeurs $F_{\sim n}(L_i)$ sont calculées, à partir des valeurs de $F_{\sim n-1}(L_i)$, de la façon suivante :

$$F_{\sim n}(L_i) = 1 + \underset{C \in \text{Conn}(L_i)}{\text{MIN}} \left[S_n (C) \right]$$

Conn (L_i) désigne l'ensemble des chaînes contenant un littéral $\neg L_i$ connectées à L_i par un lien résolvant .

Soit $C \in$ Conn (L) : $C = \neg L\ P_1\ P_2 \ldots P_k$; $S_n(C)$ est un entier défini par :

$$S_n (C) = \sum_{j=1}^{k} F_{\sim n-1} (P_j)$$

On remarque que dans le calcul de $F_{\sim n}(L)$, il n'est pas tenu compte des e-réductions qui pourraient être éventuellement dans la construction des dérivations ; $F_{\sim n}(L)$ estime donc seulement un nombre d'extensions.

Pour la mise en oeuvre de ce raffinement dans le système, il suffirait d'ajouter à la liste d'identification de tout B-littéral L du système, les n valeurs $F_{\sim 1}(L),\ldots,F_{\sim n}(L)$, et à la liste d'identification de la chaîne C contenant L, les n valeurs $S_1(C),\ldots,S_n(C)$ (n > 1 semble à déconseiller). Ces va-

leurs seraient calculées lors de la création d'une nouvelle chaîne et mises à jour lors de l'addition ou de la suppression d'un lien (la mise à jour lors de l'addition d'un lien n'étant pas obligatoire).

X.5 - Traitement de formules du second ordre

Il n'est pas question ici de construire un démonstrateur général qui traite n'importe quel type de formules du second ordre ; nous nous intéressons, au contraire, à certaines formules choisies à l'avance pour l'intérêt qu'elles présentent. Nous pensons, en particulier, aux formules qui expriment le principe d'induction et l'égalité. On peut énoncer ces formules de la façon suivante (DARLINGTON, 1971) :

Principe d'induction :

$$(\forall f)\ \Big[\ f(0) \supset [\text{HERED}\ (f) \supset (\forall x)\ f\ (x)]\ \Big] \wedge$$
$$(\forall f)\ \Big[(\forall x)\ \ [f\ (x) \supset f\ (\text{successeur}\ (x)\ )] \supset\ \text{HERED}\ (f)\ \Big]$$

f désigne une variable de prédicat du 1er ordre à un argument. HERED (f) désigne le prédicat du second ordre : "f est une propriété héréditaire" (i.e. si x possède la propriété f, successeur (x) la possède également).

Egalité :

$$(\forall x)\ (\forall y)\ \Big[\ \text{EGAL}\ (x,y) \supset (\forall f)\ \ [f(x) \supset f(y)]\ \Big] \wedge\ (\forall x)\ \text{EGAL}\ (x,x)$$

Ces formules correspondent aux clauses suivantes :

$\neg$ f (0) $\neg$ HERED (f) f (x)	[INDUC 1]
HERED (f) f (choix (f))	[INDUC 2]
HERED (f) $\neg$ f (successeur (choix (f)))	[INDUC 3]
$\neg$ EGAL (x,y) $\neg$ f (x) f (y)	[EGALITE 1]
EGAL (x,x)	[EGALITE 2]

Choix est un symbole fonctionnel d'ordre 1 qui a la signification suivante : choix (f) a pour valeur un individu α tel que

$$f(\alpha) \text{ est vrai} \iff (\exists x)\ f(x) \text{ est vrai.}$$

Soit $\Pi_1^{(1)}$ l'ensemble des prédicats du 1er ordre à 1 argument. Soit P, un élément quelconque de $\Pi_1^{(1)}$; lorsque f parcourt $\Pi_1^{(1)}$, choix (f) a pour valeur la fonction de Skolem correspondant à l'élément de $\Pi_1^{(1)}$ dont f prend la valeur ; par exemple, si f = P, choix (P) = a, avec P (a) vrai $\iff (\exists x)P(x)$ vrai.

Intuitivement, l'utilisation d'INDUC1, correspond à la démonstration d'une propriété f pour une valeur initiale 0 ; l'utilisation d'INDUC2 correspond à l'assertion de l'hypothèse d'induction : on suppose que f est vrai pour un élément α ; l'utilisation d'INDUC3 correspond à la démonstration de la propriété f pour la valeur successeur (α).

L'utilisation de ce type de formules du second ordre nécessite un algorithme d'unification plus général que celui nécessaire pour des formules du 1er ordre ; il utilise le mécanisme de λ-abstraction dont nous décrivons le fonctionnement sur un exemple.

Exemple -

Soient les deux littéraux $\neg f(x)$ et $\neg P(y,h(b))$; les substitutions les plus générales qui rendent complémentaires ces 2 littéraux, sont les suivantes :

$$\theta_1 = \{x \,.\, y\ ,\ f\ .\ \lambda(u)\ \neg P\ (u\ ,\ h(b))\}$$

$$\theta_2 = \{x\ .\ b\ ,\ f\ .\ \lambda(u)\ \neg P(y\ ,\ h(u))\}$$

$$\theta_3 = \{x\ .\ h(b)\ ,\ f\ .\ \lambda(u)\ \neg P(y\ ,\ u)\}$$

On a les propriétés suivantes : $\lambda(u)\ \neg\ \phi(u,b)\ [a] = \neg\ \phi(a,\mathbf{v})$

$$\neg\ \lambda(u) - \phi(u,v)\ [a] = \phi(a,v)$$

On voit apparaître un problème qui est celui du choix du terme sur lequel porte l'abstraction. On peut supposer que le démonstrateur serait guidé dans ce choix par des considérations sémantiques.

Un autre problème est celui de la représentation des données ; par analogie avec la représentation des termes, nous pensons à la représentation suivante.

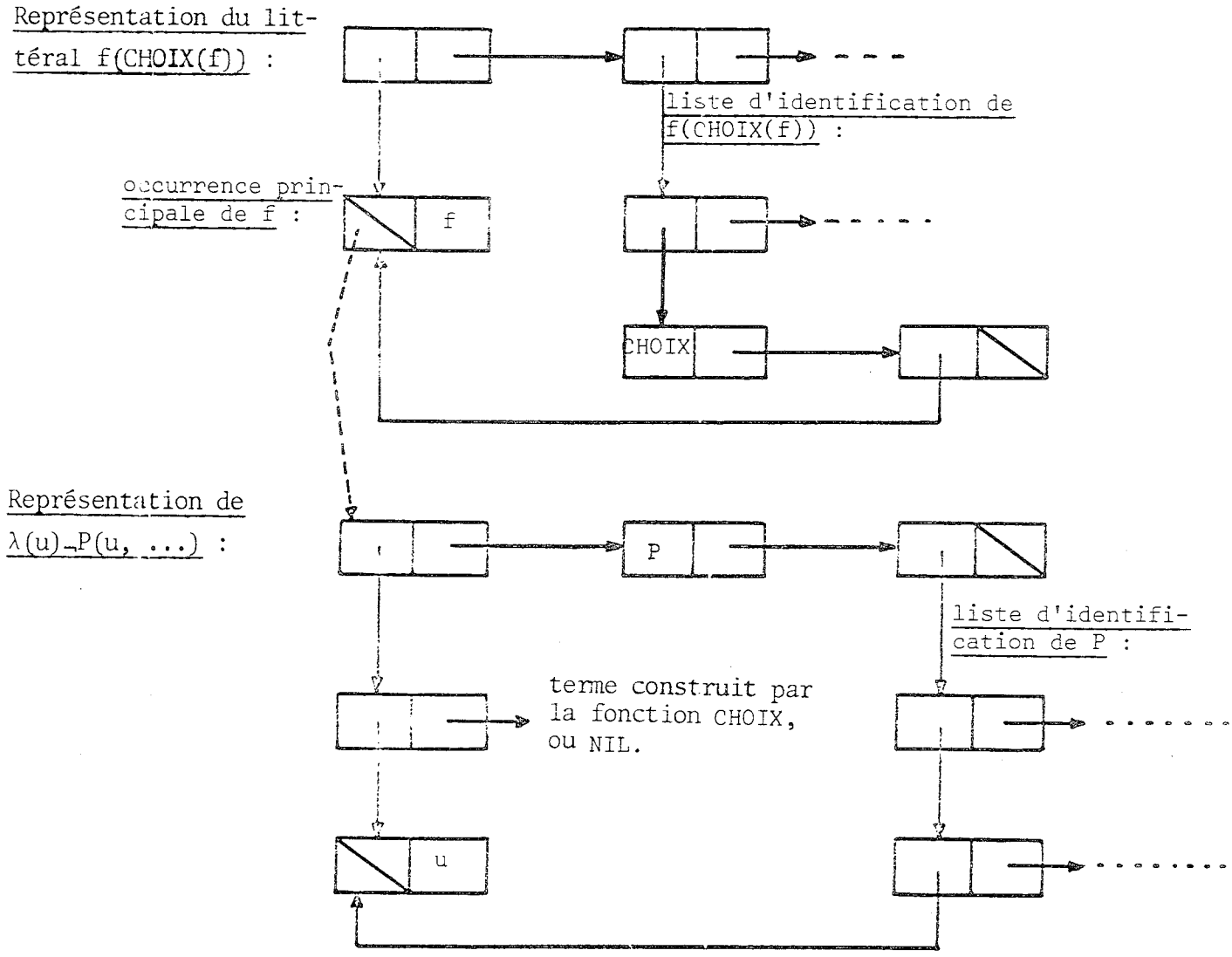


Figure X.1 : Représentation d'une variable de prédicat du 1er ordre et d'un prédicat du 1er ordre obtenu par λ-abstraction.

De même que pour des variables ordinaires (cf. VIII.4), une expression $f\theta$, où $\theta = \{f \,.\, \lambda(u)\ \phi(u)\}$ s'obtiendrait en remplaçant le NIL de l'occurrence principale de f par un pointeur vers la représentation de $\lambda(u)\ \phi(u)$, et en

recopiant f. Lors du recopiage, le symbole fonctionnel CHOIX aurait un traitement spécial : il serait remplacé soit par un terme nouvellement construit à cette occasion, soit par le terme déjà construit lors d'un recopiage antérieur de la fonction CHOIX.

La liste d'identification du littéral f ne comporterait pas de liste de liens. Par contre, celle du littéral P dans $\lambda(u) \neg P(u)$ en comporterait une, construite au moment de l'élaboration de la λ-abstraction.

Remarques -

. A l'occasion de la mise en oeuvre d'un précédent démonstrateur, par ailleurs plus rudimentaire que celui décrit ici (n'utilisant ni S-L graphes, ni stratégie diagonale), nous avons expérimenté le mécanisme de λ-abstraction sur les formules du second ordre, présentées ci-dessus. Ce démonstrateur était utilisé en système de questions-réponses (en se servant du prédicat REPONSE décrit dans GREEN, 1969) pour synthétiser des programmes itératifs élémentaires, selon les principes décrits par MANNA et WALDINGER (1971)

. Le traitement de l'égalité peut être introduit autrement dans le système, comme nous allons le voir au paragraphe suivant.

X.6 - Littéraux et fonctions évaluables

Nous avons trouvé l'idée de littéral évaluable dans COLMERAUER et al. (1972) et ROUSSEL (1972).

Il est possible, dans certains cas, de déterminer la valeur de vérité d'un littéral ou la valeur d'une fonction, selon des critères sémantiques. Ceci peut entraîner une simplification du S-L graphe, lors du recopiage d'une chaîne ; en effet, soit L un littéral évaluable d'une chaîne C ; si L s'évalue à FAUX, on peut supprimer du S-L graphe L et les liens qui lui sont connectés car L est réfuté ; si L s'évalue à VRAI, on peut supprimer C du S-L graphe, car C est irréfutable (analogue d'une tautologie); si L ne s'évalue pas, L est normalement recopié. Dans le cas d'une fonction évaluable g, si

l'évaluation de g fournit une valeur, on recopie cette valeur, sinon on recopie g normalement.

Les actions effectuées consécutivement à l'évaluation d'un littéral à VRAI ou FAUX sont des extensions des règles proposées par ROUSSEL (1972) dans le cas particulier du prédicat égalitaire (i.e. règle de suppression des i-tautologies et règle de contraction).

Cette extension permettrait donc, en particulier, de traiter l'égalité formelle ; elle permettrait également l'incorporation de façon élégante, du prédicat REPONSE de GREEN, et, d'une manière générale, de tout prédicat introduit pour ses effets de bord (prédicats d'entrée-sortie), ou pour garder une trace de certaines actions effectuées par le démonstrateur. Le démonstrateur deviendrait alors une sorte d'interpréteur d'un langage de programmation logique permettant d'écrire des proprammes non déterministes (cf. KOWALSKI, 1973 b, COLMERAUER et al. 1972).

Un exemple de fonction évaluable est celui de la fonction CHOIX du paragraphe précédent ; d'autre part, il paraît naturel que des fonctions comme les fonctions arithmétiques usuelles, les fonctions élémentaires d'opérations sur les listes et, de manière générale, les fonctions qui correspondent à des opérations de base d'un langage de programmation usuel, soient évaluables.

L'incorporation au système de cette extension serait assez facile. Il suffirait, par exemple, de distinguer, par un codage approprié, les noms des littéraux et des fonctions évaluables, de remplacer, le cas échéant, le recopiage d'une expression par un appel à la fonction EVAL de l'interpréteur LISP, et d'effectuer les actions décrites ci-dessus (dans le cas d'un littéral) selon la valeur retournée par EVAL.

Signalons enfin que les littéraux évaluables pourraient être un moyen commode d'établir une communication entre le démonstrateur et un module indépendant de celui-ci.

X.7 - Suppression de certaines redondances

Cette modification du système necessite, au préalable, la mise en oeuvre de l'extension proposée au paragraphe précédent. Elle n'a de sens que si la factorisation non basique est employée dans le système (ce qui n'est pas le cas actuellement avec les S-L graphes simplifiés).

Les liens d'admissibilité évitent certaines redondances ; en effet, considérons, par exemple, la chaîne C = P(x) P(a) Q(x). Lors de la construction du facteur C' = f(a) Q(a), le lien facteur qui relie P(x) à P(a) est transformé en lien d'admissibilité ; ce lien sert à interdire une éventuelle unification entre P(x) et $\neg$ P(a) qui conduirait à une chaîne pouvant aussi être construite avec C' ; cette barrière est cependant insuffisante. En effet, soit C_1 = P(a) Q(y) une chaîne descendant de C (après résolution de P(x) et d'un littéral $\neg$ P(y) d'une autre chaîne). L'éventuelle unification de Q(y) et d'un littéral $\neg$ Q(a) ne peut être empêchée (car il n'y a plus de lien d'admissibilité correspondant), et elle conduit cependant à la construction d'une chaîne pouvant être aussi engendrée par C'.

Pour remédier à cette situation, on pourrait modifier la construction du facteur C' de C de la façon suivante :

- le lien facteur ne serait plus remplacé par un lien d'admissibilité,
- il serait rajouté à la chaîne C un littéral EGAL (x,a).

La même procédure pourrait être accomplie lors d'une résolution ancestrale, le lien ancestral correspondant étant supprimé au lieu d'être remplacé par un lien d'admissibilité.

X.8 - Nouveaux paramètres de contrôle

Ces paramètres sont utilisés par FLEISIG et al. (1974). Le premier spécifie un entier pour chaque clause d'entrée ; cet entier indique le nombre maximum de fois qu'une clause d'entrée peut être utilisée dans une dériva-

tion donnée (lors d'opération d'extensions). Avec le second paramètre, les clauses sont groupées en catégories ; pour chaque catégorie, un entier indique le nombre maximum de fois qu'une clause de la catégorie peut être utilisée dans une dérivation donnée.

Le contrôle introduit par le premier paramètre est équivalent à celui introduit par la V-résolution (CHANG, 1972) ; dans la V-résolution, pour limiter à n le nombre d'utilisations d'une clause C dans une dérivation, on remplace au début la clause C par n copies de C ; dans les copies de C, les variables universelles sont remplacées par des "variables contraintes" qui jouent le rôle d'inconnues dans des équations algébriques : dans une dérivation donnée, contrairement à une variable universelle, une variable contrainte ne peut prendre qu'une seule valeur.

Exemple -

$$C_1 = \neg P(x) \vee P(f(x))$$
$$C_2 = P(a)$$

dans C_1, x est une variable universelle ; on peut engendrer la suite infinie de clauses P(f(a)), P(f(f(a))),....

$$C'_1 = \neg P(1) \quad P(f(1))$$
$$C_2 = P(a)$$

dans C'_1, 1 est une variable contrainte ; on peut engendrer la clause

$C'_3 = P(f(a)) \mid \{1 \leftarrow a\}$

On ne peut pas engendrer P(f(f(a))), car il faudrait attribuer la valeur f(a) à 1 qui possède déjà une valeur.

La V-résolution peut être introduite dans le système au prix d'une modification dans l'algorithme d'unification et de la création du nouveau type de variable.

Notons que tous les changements envisagés dans ce chapitre ne doivent être appliqués au'après une étude expérimentale ; en effet, il est souhaitable que le gain apporté par une modification du démonstrateur soit supérieur à la perturbation supplémentaire qu'elle entraîne dans le système.

XI - CONCLUSION

Résumons la démarche suivie au cours de ce travail.

Après une étude théorique des fondements du problème de la démonstration automatique, nous avons examiné plusieurs procédures parmi les principales procédures de preuve connues. Dans un souci de clarification, nous nous sommes efforcés, à chaque fois, de séparer au maximum, dans une procédure de preuve, les deux notions fondamentales :

(1) La méthode de déduction qui est un processus de construction de clauses, et qui définit donc un espace de recherche de clauses, pour un ensemble de clauses donné au départ.

(2) La stratégie qui est un processus de recherche de la clause vide dans cet espace.

En utilisant les méthodes de SL-résolution, Model-elimination et graphes de classification, nous avons défini successivement la méthode des S-L graphes et celle des S-L graphes simplifiée. Nous avons ensuite défini quelques paramètres de contrôle permettant de limiter la complexité des chaînes construites ; l'utilisation conjointe de la stratégie diagonale montante et de ces paramètres équivaut à la définition implicite d'une nouvelle stratégie de recherche.

Nous avons ensuite implanté sur ordinateur la procédure de preuve ainsi définie, et procédé à sa mise au point en y incluant des possibilités supplémentaires (dialogue, trace, sauvegarde/restauration, mécanisme de récupération de place, impression de statistiques).

Enfin, nous avons expérimenté le système ainsi obtenu sur un grand nombre d'exemples.

Si l'on examine les performances des systèmes décrits dans la littérature spécialisée, notre démonstrateur supporte avantageusement la comparaison, dans la plupart des cas.

Cependant, il faut souligner que ses bons résultats sont souvent dûs à la valeur "bien ajustée" des paramètres de contrôle, et que la détermination de ces valeurs requiert de l'intuition et/ou un certain nombre d'essais au préalable.

Rappelons également deux limitations importantes de notre système :

- Il n'accepte que des problèmes formulés dans le cadre de la logique du 1er ordre.
- Il n'est qu'une procédure de preuve ; ce qui signifie que si on lui soumet un problème qui correspond à une formule $\mathcal{F}$ qui n'est pas un théorème (de la logique du 1er ordre), il est incapable de répondre négativement, c'est-à-dire d'affirmer que $\mathcal{F}$ n'est pas un théorème (cette limitation est intrinsèque ; elle est due à l'indécidabilité de la logique du 1er ordre).

La puissance atteinte par notre démonstrateur n'est pas encore suffisante pour envisager des applications de grande envergure : la validation d'un gros programme (par exemple, un compilateur) dans des limites raisonnables de temps et d'espace ne pourrait être effectuée que par un démonstrateur beaucoup plus puissant. Il est incontestable que d'énormes progrès restent encore à faire dans ce domaine.

Nous pensons que les recherches pourraient être utilement orientées dans les directions suivantes :

(1) Rendre possible l'incorporation, dans un système de démonstration, au niveau des structures de données et des opérations de base qui manipulent ces données (en particulier l'unification), de certaines propriétés élémentaires des relations et des fonctions courantes, telles que l'associativité, la commutativité, la transitivité, la distributivité, etc... Actuellement, on exprime ces propriétés à l'aide de clauses qui comportent beaucoup de littéraux et/ou de variables quantifiées universellement, ce qui est un facteur d'inefficacité. Une telle possibilité simplifierait, en outre, la formalisation des problèmes.

(2) Certaines propriétés, comme l'égalité ou l'induction, peuvent être exprimées soit par un axiome du 2° ordre (cf. X.5), soit par une infinité d'axiomes du 1er ordre. A cette alternative correspondent les deux solutions suivantes :

. Construire des systèmes qui traitent la logique du 2° ordre (ou seulement une classe de formules de cette logique),

. rendre possible l'inclusion dans un démonstrateur, au niveau des structures de données et des opérations de base, du traitement de ces propriétés particulières (on a, dans ce cas, une approche analogue à celle de (1)),ce qui revient en somme à simuler l'inclusion dans le système d'une infinité d'axiomes du 1er ordre.

(3) Parmi les extensions envisagées au chapitre précédent, celle qui concerne l'évaluation sémantique de certains littéraux et symboles fonctionnels, nous paraît capitale pour accroître la puissance d'un démonstrateur : en effet, est-il nécessaire d'utiliser l'unification et la résolution pour déterminer, par exemple, la valeur de vérité du prédicat INFERIEUR (X, PLUS (X,Y)) lorsque X et Y parcourent $\mathbb{N}$?

Intuitivement, il serait souhaitable que, d'une manière générale, le démonstrateur puisse se décharger du travail qui consiste à effectuer des déductions de "bas niveau", i.e. qui peuvent se faire directement en utilisant les opérations primitives de l'ordinateur : il est plus commode d'utiliser les instructions machines d'un ordinateur que les axiomes de PEANO, pour faire des calculs sur les nombres entiers ! Le démonstrateur pourrait alors porter tout son effort sur des déductions de "haut niveau", i.e. faisant intervenir des prédicats plus élaborés tels que, par exemple, X est PREMIER, Z est le PGCD de X et de Y, etc....

Un problème se pose alors : comment déterminer la frontière entre ce qui est de bas niveau et ce qui ne l'est pas ; autrement dit, comment établir exactement le choix entre les propriétés qui doivent être incorporées dans le système, et celles qui doivent être axiomatisées ? La solution de ce problème délicat passe peut-être par une expérimentation systématique.

La liste des voies de recherche en démonstration automatique est loin d'être close. Cependant, celles qui sont esquissées ici nous semblent intéressantes. Il n'est pas invraisemblable de penser que les systèmes de démonstration automatique des prochaines générations possèderont des caractéristiques du type de celles décrites ci-dessus, qui sont de nature à accroître sensiblement leur puissance et leur domaine d'application.

APPENDICE A

Liste et description sommaire des fonctions du système

Les identificateurs utilisés dans cette description ne sont pas toujours les mêmes que ceux utilisés dans notre programme (nous avons effectué cette transcription avec l'espoir de faciliter la compréhension du lecteur).

1) Fonction d'accès

ALIT (LITT)

est vrai si LITT est un A-littéral

ANCETRE1 (CHAINE)

si CHAINE est un axiome, nom de l'axiome, sinon 1er ancêtre immédiat de CHAINE

ANCETRE2 (CHAINE)

2ème ancêtre immédiat de CHAINE (extension), ou MFACT ou ANCEST (e-réduction) ou NIL (axiome)

ARGS (LITT)

a pour résultat la liste des arguments d'un littéral

AXIOM (CHAINE)

est vrai si CHAINE est un axiome

CHAINE (LIEN)

a pour résultat la partie CHAINE de LIEN

CONSTO (TERME)

est vrai si TERME représente un symbole fonctionnel d'ordre 0 (constante)

COUT (CHAINE)

est un entier mesurant le coût de CHAINE (i.e. nombre d'extensions + nombre de e-réductions)

FILS (FONC)

a pour résultat la liste des arguments du symbole fonctionnel FØNC

FRERE (X)

a pour résultat le terme qui suit le terme X dans une liste de termes

GENIDENT (CH1 CH2 H G S)

a pour résultat une liste d'identification de chaîne correspondant aux paramètres ; en outre, le compteur de chaîne est incrémenté d'une unité.

GENLIEN (TYP LITT SUBS)

a pour résultat un lien correspondant aux paramètres

GENLIT (SG NM ALT ARG LK)

a pour résultat un littéral correspondant aux paramètres

HEURIST (CHAPINE)

sa valeur est le nombre de B-littéraux de CHAINE

INMORACT (LITT)

est vrai si LITT est dans le morceau actif d'une chaîne

ISFONCT (TERME)

est vrai si TERME représente un symbole fonctionnel

ISLIEN (X)

est vrai si X représente un lien

ISVAR (TERME)

est vrai si TERME représente une variable

LIENS (LITT)

a pour résultat la liste des liens de LITT

LIT (LIEN)

a pour résultat la partie LITTERAL de LIEN

LITS (CHAINE)

a pour résultat la liste des littéraux de CHAINE

MORT (CHAINE)

vrai si l'indicateur MORT de CHAINE est vrai

NOM (X)

a pour résultat le nom de X, si X représente un LITTERAL ou un symbole fonctionnel

NUM (CHAINE)

est un entier représentant le numéro de la chaîne

NUMCARAC (CHIFFRE)

a pour résultat le caractère correspondant à CHIFFRE

REDUC (X)

est vrai si X est un lien ancestral ou facteur

REPRISE (VAR)

a pour résultat soit l'occurrence principale de la variable VAR, soit un terme si VAR a fait l'objet d'une substitution

RES (X)

est vrai si X est un lien résolvant

REVLITS (LITS)

a pour résultat la suite inverse des littéraux de LITS

SETFRERE (TERME1 TERME2)

TERME2 devient le terme qui suit TERME1, dans la liste d'arguments dans laquelle se trouve TERME1

SETIDENT (CHAINE IDENT)

IDENT devient la liste d'identification de CHAINE

SETLIENS (LITT LIENS)

LIENS devient la liste de liens de LITT

SETLIT (LITT LIEN)

LITT devient le littéral de LIEN

SETLITS (CHAINE LITS)

LITS devient la suite des littéraux de CHAINE

SETYPLN (LIEN TYP)

TYP devient le type du lien LIEN

SIGNE (LITT)

est vrai si le signe de LITT est positif

SUB (LIEN)

a pour résultat la substitution de LIEN

SUIV (LIT)

a pour résultat le littéral qui suit LIT dans une chaîne

SUPPORT (CHAINE)

est vrai si CHAINE possède le support

TERMINAL (X)

est vrai si X ne contient aucune variable

TUER (CHAINE)

positionne à vrai l'indicateur MORT de CHAINE

TYPLIEN (X)

a pour résultat le type de lien X

VAL ( )

fonction auxiliaire de GENARGS

VARP (CARAC)

fonction de lecture ; vrai si CARAC est l'un des caractères X, Y, Z, U, V, W, M qui dénotent une variable

2) Fonctions de génération

COPLITS (CURLIT LITS)

recopie tous les littéraux de LITS, sauf CURLIT

GENARGS (ARGS)

construit la représentation interne correspondant à la liste d'arguments ARGS

GENCHAINS (FICHIER)

construit un S-L graphe simplifié à partir de FICHIER, fonction principale de la phase génération

GENFACTS (CHAINLIST)

inclut dans le S-L graphe tous les facteurs des chaînes d'entrée de CHAINLIST

GENVAR (X)

construit un nouveau nom pour la variable X (standardisation des variables)

RESLNS (CHAINLIST)

construit les liens résolvants reliant les occurrences des littéraux des chaînes d'entrée de CHAINLIST

TESTPUR (CHAINLIST)

supprime du S-L graphe les chaînes de CHAINLIST qui possèdent un B-littéral pur

UPVOCAB (SYMB)

est utilisée pour le codage des symboles d'entrée

3) Fonctions relatives à la méthode de déduction

ADDCHAINE (CHAINE LITT LIEN I J)

est la fonction principale de la méthode de déduction ; construit une nouvelle chaîne de mérite (I J), à partir du lien LIEN appartenant au littéral LITT de CHAINE ; effectue, selon la nature de LIEN, une opération d'extension ou de e-réduction ; le cas échéant, une postfactorisation est effectuée.

COPY (X NIVEAU)

recopie une liste d'arguments X, à condition que le niveau maximum d'imbrication fonctionnel de X soit NIVEAU

COPYLIT (LITT STATLIT)

recopie le littéral LITT avec ses liens (en tenant compte des substitutions éventuelles) ; STATLIT indique le statut du littéral ; la condition d'admissibilité est testée à cette occasion

COPYLITS (LITS)

recopie une suite de littéraux avec leurs liens (en tenant compte des substitutions éventuelles)

NBALIT (LITS)

compte le nombre de A-littéraux présents dans LITS

OCCUR (X TERME)

fonction auxiliaire de UNIFIE ; vrai si la variable X apparaît dans TERME

PHI (FONC CLEX CLEY)

est une fonction d'adressage dispersé utilisée par RECHERCHE

PROFOND (X)

calculé le niveau d'imbrication fonctionnel de X

RECHERCHE (TERME)

si TERME est constant (i.e. sans variable), effectue une recherche de TERME dans le dictionnaire des listes terminales DICTERM et l'y introduit s'il ne l'y trouve pas

SELECT (CHAINE)

sélectionne un B-littéral dans le morceau actif de CHAINE pour l'opération de e-extension (méthode S-L graphe non simplifiée)

SUBREDUC (SUBST)

construit la substitution réduite de SUBST

SUBSTIT (SUBST BOOL)

effectue une substitution sur certaines variables ou les restaure dans leur état initial selon la valeur de BOOL

TRONQUATION (LITS)

effectue l'opération de tronquation sur une suite de littéraux

UNIFIE (LISTE1 LISTE2)

calcule et construit le plus grand unificateur (s'il existe) de 2 listes de termes

UNIFTERM (TERME1 TERME2)

calcule et construit le p.g.u. (s'il existe) de 2 termes

UNITSUB (LIT), DBLESUBS (LITS)

effectuent respectivement les tests de subsomption unitaire et de subsomption double

4) Fonctions relatives à la stratégie

DEMONTRE (FICHIER DIAGMAX)

est la fonction principale du système ; FICHIER contient un ensemble de clauses ; DIAGMAX est le numéro de la diagonale jusqu'à laquelle la stratégie cherchera une preuve

DIAGAMONT cf. VIII.3

DIAGAVAL cf. CIII.3

REDEM (DIAGMAX)

est une fonction analogue à DEMONTRE ; est utilisée pour poursuivre une démonstration commencée lors d'une session précédente, puis interrompue et sauvegardée sur fichier, puis restaurée en mémoire centrale ; DIAGMAX a la même signification que pour démontre

SIGMA cf. VIII.3

5) Fonctions de manipulation du S-L graphe

COLLECT (D MAXBLITS)

est utilisée pour libérer la mémoire occupée par des chaînesdevenues inutiles

INCOMPSUB (VAR TERME1 LIT CURLIT) cf. VIII.5.2

INITTAB ()

initialise à NIL tous les éléments du tableau TAB

PUR (LIENS)

est vrai si LIENS ne contient aucun lien résolvant

SETTAB (I J X)

affecte X à l'élément (I J) du tableau TAB

SUPCHAINE (CHAINE)

supprime CHAINE du S-L graphe

SUPINCOMPLNS cf. VIII.5.2

SUPLIEN (LIEN)

supprime LIEN du S-L graphe

SUPTAUTOLN, SUPTAUTOLNS cf. VIII.5.1

TAB (I J)

a pour valeur l'élément (I J) du tableau TAB

UPDATLNS (LITS LITT)

construit les liens FACTEUR, ANCESTRAUX, d'ADMISSIBILITE, entre LITT et les littéraux de LITS

UPDATTAB (I J X)

rajoute X dans l'élément (I J) du tableau TAB

6) Fonctions d'intérêt général

FIND (X Y F)

Y est une liste de doublets de la forme $((u_1.v_1)....(u_n.v_n))$; le résultat est u_i si v_i est le premier élément de Y égal à x ; s'il n'en existe aucun, le résultat est F( )

FIND2 (X Y)

Y est défini comme ci-dessus; le résultat est v_i si u_i est le premier élément de Y égal à x ; s'il n'en existe aucun, le résultat est x

PROG3 (X Y Z)

a pour résultat Z et sert à évaluer ses paramètres

BIPRINT (X), BIREAD (X)

dirigent les entrées-sorties sur le fichier X, ou sur le télétype si X = NIL

MACRO, DEFINE2, MDEF

sont les fonctions utilisées pour définir des macros en LISP

GET2 (X Y) cf. VIII.5.2

STOP (VALEUR BOOL IDENT)

est utilisée pour la mise au point du programme ; interrompt le déroulement de ce dernier si BOOL a la valeur vrai, IDENTn est alors imprimé ; il est possible de connaître et/ou modifier le contenu des variables du programme, puis de reprendre le cours du programme ; le résultat de la fonction est VALEUR

7) Fonctions d'impression

PRINATOM (X), PRINT2 (X), PRINT3 (X)

sont utilisées pour l'impression des listes circulaires

PRTARGS (ARG SEP)

imprime les arguments d'un littéral ou d'un symbole fonctionnel en les délimitant par SEP

PRTCHAINE (CHAINE BOOL)

imprime une chaîne avec ou sans ses liens selon la valeur de BOOL

PRTCHAINES (CHNS BOOL)

imprime une liste de chaînes avec ou sans leurs liens selon la valeur de BOOL

PRTLIEN (X)

imprime le lien X

PRTLIENS (LITT)

imprime la liste de liens de LITT

PRTLIT (X)

imprime le litteral X

PRTPREUVE (CHAINE)

imprime une déduction de CHAINE, les valeurs des principaux paramètres du programme et certaines statistiques relatives à cette déduction

PRTTAB (BOOL)

imprime toutes les chaînes de TAB avec ou sans leurs liens selon la valeur de BOOL

8) Variables globales

COMPTCHN

compte les chaînes construites par le démonstrateur

CODE

est un compteur utilisé pour le codage des symboles d'entrée

DIALOGUE

est un indicateur booléen permettant, s'il est à VRAI, de faire fonctionner le démonstrateur en mode conversationnel : chaque fois qu'il commence à remplir une nouvelle diagonale du tableau TAB, le démonstrateur s'interrompt ; il est alors possible de faire toutes les modifications que l'on souhaite (supprimer des chaînes, changer la valeur des paramètres de contrôle, etc....)

DICTERM

est un dictionnaire des listes de termes constants

FACTS

est un indicateur booléen permettant d'autoriser, ou d'interdire, la factorisation non basique

MAXALITS

est un paramètre de contrôle du programme permettant d'interdire la construction des chaînes contenant plus de MAXALITS A-littéraux

MAXBLITS

est un paramètre de contrôle du programme permettant d'interdire la construction des chaînes contenant plus de MAXBLITS B-littéraux

MAXFN

est un paramètre de contrôle du programme permettant d'interdire la construction des chaînes contenant des termes dont le niveau d'imbrication fonctionnel dépasse MAXFN

SORTIETAB

indique si les chaînes de TAB sont imprimées ou non, à la fin de la démonstration

SUPINCOMP, SUPTAUTO cf. VIII.5.2

TABIJ

contient toutes les chaînes unitaires construites par le démonstrateur

TRACE

est un indicateur booléen qui, s'il est à VRAI, entraîne l'impression au fur et à mesure des chaînes construites par le programme, ainsi que d'autres indications sur le déroulement du programme

VOCAB

est une table de codage des symboles d'entrée.

APPENDICE B

Exemple de fonctionnement d'une stratégie diagonale montante Σ

Considérons l'ensemble de clauses envisagé au début de la 2ème partie :

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

$C_1 = P(x)\ Q(x)$

$C_2 = \neg Q(f(z))$

$C_3 = \neg P(f(z))\ R(z)$

$C_4 = \neg R(v)\ \neg R(w)$

Le coût g(D) d'une dérivation D d'une clause C est défini par le niveau de D. L'heuristique h(D) d'une dérivation D d'une clause C est définie par la longueur de C. Soit $K = \{C_1\}$, un ensemble de support.

Nous supposons en outre que la méthode de déduction choisie est la résolution ordinaire avec ensemble de support.

Σ commence par initialiser TAB avec les clauses de S et leurs facteurs ; Σ provoque donc la construction du facteur $C_5 = \neg R(W)$ dans TAB (1,1). Les clauses de S sont rangées dans TAB(1,0) et TAB(2,0).

Σ tente ensuite, sans succès, de remplir TAB(0,1) dans d_1 et TAB(0,2) dans d_2 (voir figure) ; en remontant d_2, Σ provoque la construction du résolvant $C_6 = P(f(z))$ (à partir de C_1 et C_2) qu'elle inclut dans TAB(1,1). Σ essaie aussitôt, et sans succès, de redescendre d_2 pour remplir TAB(0,2) en utilisant C_6 ; en remontant d_3, Σ n'inclut rien dans TAB(0,3), puis insère $C_7 = R(z)$, (obtenu par résolution de C_6 et C_3) dans TAB(1,2) ; enfin, Σ redescend d_3, avec succès, pour insérer $C_8 = \square$ dans TAB(0,3) obtenue par résolution de C_7 et C_5.

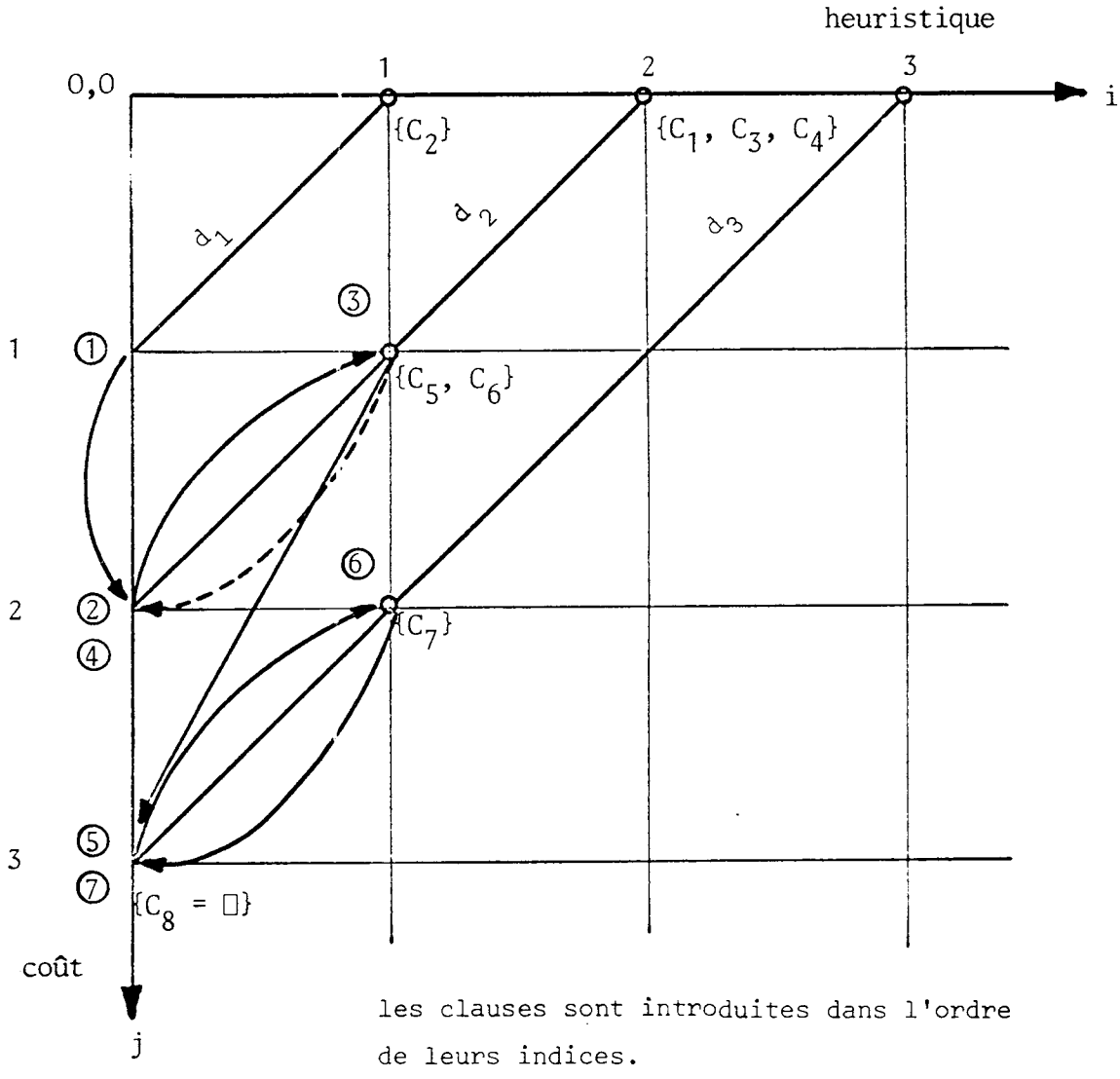


Figure B.1 : Ordre de remplissage de TAB par Σ, lors de la réfutation de S.

APPENDICE C

Preuves des exemples du chapitre IX

```
EXEMPLE GROUP1

TEMPS=1973,70 : ILLISCOMPES
                    PREUVE*DE*NIVEAU*7
MAXEU*MAXBLITS*MAXALITS
3       3         2
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
         20        19              8         4

4 TH1   +P(A B C)

3 CARREUL   +P(X3 X? E)

1 DUEUTUE   +P(X1 E Y1)

6 ASSOC1   +P(U6 Z6 V6)  -P(X6 V6 U6)  -P(Y6 Z6 V6)  -P(X6 Y6 U6)

2 QUEUTUE   +P(E Y2 X2)

5 TH2   -P(B A C)

8 ASSOC1   -P(Y6 Z6 V6)  +P(U6 Z6 V6)  -P(X6 V6 U6)  -P(X6 Y6 U6)

15 FROM 3,8   +P(U6 X3 V6)  -P(Y6 E V6)  -P(X6 Y3 U6)  /+P(Y3 X3 E)/

16 FROM 15,5   -P(X6 E C)  -P(X6 A B)  /+P(A A E)/

17 FROM 16,1   -P(C A B)  /+P(A A E)/

13 ASSOC2   -P(X10 Y10 U10)  +P(X10 V10 U10)  -P(U10 Z10 U10)  -P(V10 Z1
0 V10)

22 FROM 3,13   +P(X3 V10 U10)  -P(E Z10 U10)  -P(X3 Z10 V10)  /+P(Y3 X3
E)/

26 FROM 22,17   -P(E Z10 B)  -P(C Z10 A)  /+P(C C E)/

27 FROM 26,2   -P(C B A)  /+P(C C E)/

28 FROM 27,6   -P(X6 V6 A)  -P(Y6 B V6)  -P(X6 Y6 C)  /-P(C B A)/  /+P(C
C E)/

29 FROM 28,1   -P(Y6 B E)  -P(A Y6 C)  /-P(C B A)/  /+P(C C E)/

30 FROM 29,3   -P(A B C)  /-P(C B A)/  /+P(C C E)/

31 FROM 30,4 **CLAUSE*VIDE**
```

```
EXEMPLE GROUP7

TEMPS=1966,93 MILLISECONDES
                    PREUVE*DE*NIVEAU*7
MAXEN*MAXBLITS*MAXALITS
2       5         2
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
          24        22              15         8

1 AX1   +P(F X1 X1)

5 AX20   +S(A)

3 AX3   +P(X3 G(Y3) E)

9 AX18   +S(Z6)  -S(Y6)  -S(Y6)  -P(X6 G(Y6) Z6)

18 T!   -S(G(A))

19 FROM 18,3   -S(Y6)  -S(Y6)  -P(X6 G(Y6) G(A))  /-S(G(A))/

35 FROM 19,3   -S(Y6)  -S(Y6)  -P(X6 G(Y6) X6)  /-S(Y6)/  -S(Y6)  -P(Y6
G(Y6) G(A))  /-S(G(A))/

36 FROM 35,5   -S(Y6)  -P(A G(Y6) X6)  /-S(Y6)/  -S(Y6)  -P(Y6 G(Y6) G(A
))  /-S(G(A))/

37 FROM 36,5   -P(A G(A) Y6)  /-S(Y6)/  -S(Y6)  -P(X6 G(Y6) G(A))  /-S(G
(A))/

38 FROM 37,3   -S(Y6)  -P(E G(Y6) G(A))  /-S(G(A))/

39 FROM 38,5   -P(E G(A) G(A))  /-S(G(A))/

40 FROM 39,1 **CLAUSE*VIDE**



EXEMPLE PRIM1

TEMPS=6644,65 MILLISECONDES
                    PREUVE*DE*NIVEAU*14
MAXEN*MAXBLITS*MAXALITS
3       4         4
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
          85        57              43         2

1 AX1   +D(X1 Y1)

12 AX7   -P(X12)  -D(Y12 A)

8 AX4   +L(1 G(X7))  +P(X7)
```

```
10 AX5   +L(G(X9) X9)  +P(X9)

17 AX9   +D(F(Y17) Y17)  -L(Y17 A)  -L(1 X17)

14 AX8   +P(F(Y14))  -L(X14 A)  -L(1 X14)

13 AX7   -D(X12 A)  -P(X12)

3 AX2   -P(Y2 Z2)  +P(Y2 Z2)  -P(Y2 Y2)

5 AX3   +D(G(Y5) X5)  +P(X5)

23 FROM 5,3   +D(Y2 X5)  -P(Y2 G(X5))  /+D(G(X5) X5)/  +P(Y5)

26 FROM 23,13   -P(Y2)  /+D(X2 A)/  -D(Y2 G(A))  /+D(G(A) A)/  +P(A)

30 FROM 26,14   -L(Y14 A)  -L(1 X14)  /-P(F(Y14))/  /+D(F(Y14) A)/  -P(F
(X14) G(A))  /+D(G(A) A)/  +P(A)

65 FROM 30,10   +P(A)  /-L(G(A) A)/  -L(1 G(A))  /-P(F(G(A)))/  /+D(F(G(
A)) A)/  -D(F(G(A)) G(A))  /+D(G(A) A)/  +P(A)

66 FROM 65,FACTEUR   -L(1 G(A))  /-P(F(G(A)))/  /+D(F(G(A)) A)/  -D(F(G(
A)) G(A))  /+D(G(A) A)/  +P(A)

67 FROM 66,8   +P(A)  /-L(1 G(A))/  /-P(F(G(A)))/  /+D(F(G(A)) A)/  -D(F
(G(A)) G(A))  /+D(G(A) A)/  +P(A)

68 FROM 67,FACTEUR   -D(F(G(A)) G(A))  /+D(G(A) A)/  +P(A)

70 FROM 68,17   -L(G(A) A)  -L(1 G(A))  /-P(F(G(A)) G(A))/  /+D(G(A) A)/
  +P(A)

71 FROM 70,10   +P(A)  /-L(G(A) A)/  -L(1 G(A))  /-P(F(G(A)) G(A))/  /+D
(G(A) A)/  +P(A)

72 FROM 71,FACTEUR   -L(1 G(A))  /-P(F(G(A)) G(A))/  /+D(G(A) A)/  +P(A)


73 FROM 72,8   +P(A)  /-L(1 G(A))/  /-P(F(G(A)) G(A))/  /+D(G(A) A)/  +P
(A)

74 FROM 73,FACTEUR   +P(A)

75 FROM 74,12   -D(A A)  /+P(A)/

76 FROM 75,1 **CLAUSE*VIDE**
```

```
EXEMPLE NUM1

TEMPS= 525,52 MILLISECONDES
                        PREUVE*DE*NIVEAU*6
MAXFN*MAXBLITS*MAXALITS
5       4         5
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
        10        8                3         1

11 AX5   +M(A S(C) S(B))

3 AX1    +D(X2 Z2)  -M(X2 Y2 Z2)

9 AX3    +M(X9 X9 S(X9))

10 AX4   +P(A)

8 AX2    +D(X4 Z4)  -P(X4)  -M(Y4 Z4 W4)  -D(X4 W4)  +D(X4 Y4)

1 TH   -D(A B)

15 FROM 1,8   -P(A)  -M(Y4 B W4)  -D(A W4)  +D(A Y4)  /-D(A B)/

16 FROM 15,10   -M(Y4 B W4)  -D(A W4)  +D(A Y4)  /-D(A B)/

17 FROM 16,9   -D(A S(B))  +D(A B)  /-D(A B)/

18 FROM 17,3   -M(A Y2 S(B))  /-D(A S(B))/  +D(A B)  /-D(A B)/

19 FROM 18,11   +D(A B)  /-D(A B)/

20 FROM 19,FACTEUR **CLAUSE*VIDE**


EXEMPLE ANNEAU

TEMPS=27005,83 MILLISECONDES
                        PREUVE*DE*NIVEAU*6
MAXFN*MAXBLITS*MAXALITS
5       4         2
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
        218       107              84        6

4 TH   -P(A D D)

2 AX2    +Q(G(X2) X2 D)
```

```
0 AX0   -Q(Y5 Z5 V5)  -Q(X5 Y5 U5)  -Q(X5 V5 W5)  +Q(U5 Z5 W5)

1 AX1   +Q(0 X1 X1)

3 AX3   +P(X3 Y3 F(X3,Y3))

13 AX7   -P(X13 Y13 U13)  -P(X13 Z13 W13)  -P(X13 V13 W13)  -Q(Y13 Z13 V
13)  +Q(U13 W13 W13)

23 FROM 3,13   -P(X3 Z13 W13)  -P(X3 V13 W13)  -Q(Y3 Z13 V13)  +Q(F(X3,Y
3) W13 W13)  /+P(X3 Y3 F(X3,Y3))/

24 FROM 23,3   -P(X3 V13 W13)  -Q(Y3 Z13 V13)  +Q(F(X3,Y3) F(X3,Z13) W13
)  /+P(X3 Y3 F(X3,Y3))/

25 FROM 24,3   -Q(Y3 Z13 V13)  +Q(F(X3,Y3) F(X3,Z13) F(X3,V13))  /+P(X3
Y3 F(X3,Y3))/

26 FROM 25,1   +Q(F(X3,0) F(X3,V13) F(X3,V13))  /+P(X3 0 F(X3,0))/

69 FROM 26,6   -Q(X3 F(X3,0) U3)  -Q(X3 F(X3,V13) W3)  +Q(U3 F(X3,V13) W
3)  /+Q(F(X3,0) F(X3,V13) F(X3,V13))/  /+P(X3 0 F(X3,0))/

1 FROM 69,2   -Q(Q(F(X3,0)) F(X3,V13) W3)  +Q(0 F(X3,V13) W3)  /+Q(F(X3
,0) F(X3,V13) F(X3,V13))/  /+P(X3 0 F(X3,0))/

82 FROM 81,2   +Q(0 F(X3,0) 0)  /+Q(F(X3,0) F(X3,0) F(X3,0))/  /+P(X3 0
F(X3,0))/

9 AX9   -Q(X9 Y9 U9)  -Q(Y9 Z9 V9)  -Q(U9 Z9 W9)  +Q(X9 V9 W9)

27 FROM 25,2   +Q(F(X3,G(Z13)) F(X3,Z13) F(X3,0))  /+P(X3 G(Z13) F(X3,G(
Z13)))/

74 FROM 27,9   -Q(F(X3,Z13) Z9 V9)  -Q(F(X3,0) Z9 W9)  +Q(F(X3,G(Z13)) V
9 W9)  /+Q(F(X3,G(Z13)) F(X3,Z13) F(X3,0))/  /+P(X3 G(Z13) F(X3,G(Z13)))
/

75 FROM 74,26   -Q(F(X3,0) F(X3,V13) W9)  +Q(F(X3,G(0)) F(X3,V13) W9)  /
+Q(F(X3,G(0)) F(X3,0) F(X3,0))/  /+P(X3 G(0) F(X3,G(0)))/

76 FROM 75,26   +Q(F(X3,G(0)) F(X3,V13) F(X3,V13))  /+Q(F(X3,G(0)) F(X3,
0) F(X3,0))/  /+P(X3 G(0) F(X3,G(0)))/

8 AX8   +Q(U5 Z5 W5)  -Q(X5 Y5 U5)  -Q(Y5 Z5 V5)  -Q(X5 V5 W5)

18 AX8   -P(X18 Y18 U18)  -P(X18 Z18 W18)  -Q(Y18 Z18 V18)  -Q(U18 W18 V
18)  +P(X18 V18 W18)

35 FROM 3,18   -P(X3 Z18 W18)  -Q(Y3 Z18 V18)  -Q(F(X3,Y3) W18 W18)  +P(
X3 V18 W18)  /+P(X3 Y3 F(X3,Y3))/

36 FROM 35,3   -Q(Y3 Z18 V18)  -Q(F(X3,Y3) F(X3,Z18) W18)  +P(X3 V18 W18
)  /+P(X3 Y3 F(X3,Y3))/
```

```
57 FROM 56,1   -P(F(X3,D) F(X3,V16) W16)  +P(X3 V16 W16)  /+P(X3 D F(X3,
D))/

56 FROM 57,26   +P(X3 V15 F(X3,V13))  /+P(X3 D F(X3,D))/

10 AXS   -P(X16 Z16 W16)  -P(X16 Y16 G16)  -P(Y16 Z16 W16)  -P(G16 W16 X
16)  +P(X16 V16 W16)

45 FROM 5,15   -P(X3 Y16 G16)  -P(Y16 Y3 V16)  -P(G16 F(X3,Y3) W16)  +P(
X3 V16 W16)  /+P(X3 Y3 F(X3,Y3))/

50 FROM 45,56   -P(V15 Y3 V16)  -P(F(X3,V13) F(X3,Y3) W16)  +P(X3 V16 W1
6)  /+P(X3 Y3 F(X3,Y3))/

52 FROM 50,2   -P(F(X3,G(Y3)) F(X3,Y3) V16)  +P(X3 Y3 W16)  /+P(X3 Y3 F(X
3,Y3))/

125 FROM 52,6   -P(X3 Y3 F(X3,G(Y3)))  -P(Y3 F(X3,Y3) V3)  -P(X3 Y3 W16)
  /-P(F(X3,G(Y3)) F(X3,Y3) W16)/  +P(X3 Y3 W16)  /+P(X3 Y3 F(X3,Y3))/

126 FROM 125,1   -P(F(X3,G(Y3)) F(X3,Y3) V3)  -P(D V3 W16)  /-(F(X3,G(Y
3)) F(X3,Y3) W16)/  +P(X3 D W16)  /+P(X3 Y3 F(X3,Y3))/

127 FROM 126,76   -P(D F(X3,D) W16)  /-P(F(X3,G(D)) F(X3,D) W16)/  +P(X3
 D W16)  /+P(X3 D F(X3,D))/

126 FROM 127,82   +P(X3 D D)  /+P(X3 D F(X3,D))/

125 FROM 123,4 **CLAUSE*VIDE**


EXEMPLE GROUPE

    135,59 MILLISECONDES
                    PREUVE*DE*NIVEAU*3
MAXFIX*MAXBLITS*MAXALITS
1       2       1
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*SUBSTANTIFS
        0       4               1       2

14 AX7   -E(X3)

11 AX4   +P(X11 X11)

1 AX1   -P(X1 Y1 Z1)  -P(X1 Y1 U1)  +E(Z1 U1)

12 AX5   +P(B X12 X12)

17 FROM 12,1   -P(B X12 U1)  +E(X12 U1)  /+P(B X12 X12)/

16 FROM 17,11   +E(X3)  /+P(B X A)/
```

```
19 FROM 18,14 **CLAUSE*VIDE**


EXEMPLE TRANS1

TEMPS=1/17,07 SECONDES
                    PREUVE*DE*NIVEAUX
MAXFN=MAXBLITS=MAXALITS
3       2       4
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
        26      12              5       4

1 AX1   +R(PLUS(A1,Y1) PLUS(Y1,X1))

2 AX2   +R(PLUS(X2,PLUS(Y2,Z2)) PLUS PLUS(X2,Y2),Z2))

3 AX7   +R(A7 Z7)  -R(A7 Y7)  -R(Y7 Z7)

20 THI   -R(PLUS(PLUS(A,B),C) PLUS(A,PLUS(B,C)))

21 FROM 20,3   -R(PLUS(PLUS(A,B),C) Y7)  -R(Y7 PLUS(A,PLUS(B,C)))  /-R(P
LUS(PLUS(A,B),C) PLUS(A,PLUS(B,C)))/

22 FROM 21,1   -R(PLUS(C,PLUS(A,B)) PLUS(A,PLUS(B,C)))  /-R(PLUS(PLUS(A,
B),C) PLUS(A,PLUS(B,C)))/

24 FROM 22,3   -R(PLUS(C,PLUS(A,B)) Y7)  -R(Y7 PLUS(A,PLUS(B,C)))  /-R(P
LUS(C,PLUS(A,B)) PLUS(A,PLUS(B,C)))/  /-R(PLUS(PLUS(A,B),C) PLUS(A,PLUS(
B,C)))/

25 FROM 24,2   -R(PLUS(PLUS(C,A),B) PLUS(A,PLUS(B,C)))  /-R(PLUS(C,PLUS(
A,B)) PLUS(A,PLUS(B,C)))/  /-R(PLUS(PLUS(A,B),C) PLUS(A,PLUS(B,C)))/

27 FROM 25,3   -R(PLUS(PLUS(C,A),B) Y7)  -R(Y7 PLUS(A,PLUS(B,C)))  /-R(P
LUS(PLUS(C,A),B) PLUS(A,PLUS(B,C)))/  /-R(PLUS(C,PLUS(A,B)) PLUS(A,PLUS(
B,C)))/  /-R(PLUS(PLUS(A,B),C) PLUS(A,PLUS(B,C)))/

28 FROM 27,1   -R(PLUS(B,PLUS(C,A)) PLUS(A,PLUS(B,C)))  /-R(PLUS(PLUS(C,
A),B) PLUS(A,PLUS(B,C)))/  /-R(PLUS(C,PLUS(A,B)) PLUS(A,PLUS(B,C)))/  /-
R(PLUS(PLUS(A,B),C) PLUS(A,PLUS(B,C)))/

30 FROM 28,3   -R(PLUS(B,PLUS(C,A)) Y7)  -R(Y7 PLUS(A,PLUS(B,C)))  /-R(P
LUS(B,PLUS(C,A)) PLUS(A,PLUS(B,C)))/  /-R(PLUS(PLUS(C,A),B) PLUS(A,PLUS(
B,C)))/  /-R(PLUS(C,PLUS(A,B)) PLUS(A,PLUS(B,C)))/  /-R(PLUS(PLUS(A,B),C
) PLUS(A,PLUS(B,C)))/

31 FROM 30,2   -R(PLUS(PLUS(B,C),A) PLUS(A,PLUS(B,C)))  /-R(PLUS(B,PLUS(
C,A)) PLUS(A,PLUS(B,C)))/  /-R(PLUS(PLUS(C,A),B) PLUS(A,PLUS(B,C)))/  /-
R(PLUS(C,PLUS(A,B)) PLUS(A,PLUS(B,C)))/  /-R(PLUS(PLUS(A,B),C) PLUS(A,PL
US(B,C)))/

32 FROM 31,1 **CLAUSE*VIDE**
```

```
EXEMPLE TRANS2

TEMPS=3004,41 MILLISECONDES
                    PREUVE*DE*NIVEAU*7
MAXEN*MAXPLITS*MAXALITS
3       3       2
CLAUSES*ESSAYEES*ACCEPTEES*DONT*INUTILES*INITIALES
        64      22              15      4

1 AX1   +P(PLUS(Y1,Y1) PLUS(Y1,Y1))

6 AX6   +P(MINUS(PLUS(X6,X6),Z6) PLUS(MINUS(X6,Z6),X6))

3 AX7   +P(X7 Z7)  -P(X7 X7)  -P(Y7 Z7)

5 AX5   +P(PLUS(MINUS(X5,Y5),Z5) MINUS(PLUS(Y5,Z5),Y5))

10 AX10   +P(U14 MINUS(Y14,V14))  -P(X14 Y14)  -P(U14 MINUS(Y14,V14))

20 TH2   -P(PLUS(MINUS(A,B),C) PLUS(A,MINUS(C,B)))

21 FROM 20,6   -P(PLUS(MINUS(A,B),C) Y7)  -P(Y7 PLUS(A,MINUS(C,B)))  /-P
(PLUS(MINUS(A,B),C) PLUS(A,MINUS(C,B)))/

37 FROM 21,10   -P(Y14 Y14)  -P(PLUS(MINUS(A,B),C) MINUS(Y14,V14))  /-P(
PLUS(MINUS(A,B),C) MINUS(Y14,V14))/  -P(MINUS(Y14,V14) PLUS(A,MINUS(C,B)
))  /-P(PLUS(MINUS(A,B),C) PLUS(A,MINUS(C,B))))/

38 FROM 37,1   -P(PLUS(MINUS(A,B),C) MINUS(PLUS(Y1,Y1),V14))  /-P(PLUS(M
INUS(A,B),C) MINUS(PLUS(Y1,Y1),V14))/  -P(MINUS(PLUS(Y1,Y1),V14) PLUS(A,
MINUS(C,B)))  /-P(PLUS(MINUS(A,B),C) PLUS(A,MINUS(C,B))))/

39 FROM 38,5   -P(MINUS(PLUS(C,A),B) PLUS(A,MINUS(C,B)))  /-P(PLUS(MINUS
(A,B),C) PLUS(A,MINUS(C,B)))/

40 FROM 39,9   -P(MINUS(PLUS(C,A),B) Y7)  -P(Y7 PLUS(A,MINUS(C,B)))  /-P
(MINUS(PLUS(C,A),B) PLUS(A,MINUS(C,B)))/  /-P(PLUS(MINUS(A,B),C) PLUS(A,
MINUS(C,B)))/

41 FROM 40,6   -P(PLUS(MINUS(C,B),A) PLUS(A,MINUS(C,B)))  /-P(MINUS(PLUS
(C,A),B) PLUS(A,MINUS(C,B)))/  /-P(PLUS(MINUS(A,B),C) PLUS(A,MINUS(C,B))
)/

42 FROM 41,1 **CLAUSE*VIDE**
```

```
EXEMPLE TRANS3

TEMPS=4446,76 MILLISECONDES
                    PREUVE*DE*NIVEAU*7
MAXEN*MAXBLITS*MAXALITS
7       3         3
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
          64        22              15        3

6 AX6   +R(MINUS(PLUS(X6,Y6),Z6) PLUS(MINUS(X6,Z6),Y6))

5 AX5   +R(PLUS(MINUS(Y5,Y5),Z5) MINUS(PLUS(Y5,Z5),Y5))

1 AX1   +R(PLUS(X1,Y1) PLUS(Y1,X1))

7 AX7   +R(X7 Z7)  -R(X7 Y7)  -R(Y7 Z7)

16 AX10   +R(U14 MINUS(Y14,V14))  -R(Y14 Y14)  -R(U14 MINUS(Y14,V14))

20 TH3   -R(PLUS(A,MINUS(B,C)) PLUS(MINUS(A,C),B))

21 FROM 20,9   -R(PLUS(A,MINUS(B,C)) Y7)  -R(Y7 PLUS(MINUS(A,C),B))  /-R
(PLUS(A,MINUS(B,C)) PLUS(MINUS(A,C),B))/

35 FROM 21,16   -R(Y14 Y14)  -R(PLUS(A,MINUS(B,C)) MINUS(Y14,V14))  /-R(
PLUS(A,MINUS(B,C)) MINUS(Y14,V14))/  -R(MINUS(Y14,V14) PLUS(MINUS(A,C),B
))  /-R(PLUS(A,MINUS(B,C)) PLUS(MINUS(A,C),B))/

36 FROM 35,1   -R(PLUS(A,MINUS(B,C)) MINUS(PLUS(X1,Y1),V14))  /-R(PLUS(A
,MINUS(B,C)) MINUS(PLUS(Y1,X1),V14))/  -R(MINUS(PLUS(Y1,X1),V14) PLUS(MI
NUS(A,C),B))  /-R(PLUS(A,MINUS(B,C)) PLUS(MINUS(A,C),B))/

39 FROM 36,9   -R(PLUS(A,MINUS(B,C)) Y7)  -R(Y7 MINUS(PLUS(X1,Y1),V14))
/-R(PLUS(A,MINUS(B,C)) MINUS(PLUS(X1,Y1),V14))/  /-R(PLUS(A,MINUS(B,C))
MINUS(PLUS(Y1,X1),V14))/  -R(MINUS(PLUS(Y1,X1),V14) PLUS(MINUS(A,C),B))
/-R(PLUS(A,MINUS(B,C)) PLUS(MINUS(A,C),B))/

40 FROM 39,1   -R(PLUS(MINUS(B,C),A) MINUS(PLUS(X1,Y1),V14))  /-R(PLUS(A
,MINUS(B,C)) MINUS(PLUS(X1,Y1),V14))/  /-R(PLUS(A,MINUS(B,C)) MINUS(PLUS
(Y1,X1),V14))/  -R(MINUS(PLUS(Y1,X1),V14) PLUS(MINUS(A,C),B))  /-R(PLUS(
A,MINUS(B,C)) PLUS(MINUS(A,C),B))/

41 FROM 40,5   -R(MINUS(PLUS(A,B),C) PLUS(MINUS(A,C),B))  /-R(PLUS(A,MIN
US(B,C)) PLUS(MINUS(A,C),B))/

42 FROM 41,6 **CLAUSE*VIDE**
```

```
EXEMPLE TRANS4

TEMPS=3458,05 MILLISECONDES
                    PREUVE*DE*NIVEAU*7
MAXEN*MAXBLITS*MAXALITS
3       3         2
CHAINES*ESSAYEES*ACCEPTEES*NOEUT*INUTILES*UNITAIRES
          54        10              12        4

1 AX1   +R(PLUS(X1,Y1) PLUS(Y1,X1))

6 AX6   +R(MINUS(PLUS(X6,Y6),Z6) PLUS(MINUS(X6,Z6),Y6))

9 AX7   +R(X7 Z7)  -R(X7 Y7)  -R(Y7 Z7)

10 AX8   +R(X10 X10)

16 AX10   +R(U14 MINUS(Y14,V14))  -R(X14 Y14)  -R(U14 MINUS(Y14,V14))

20 TH4   -R(MINUS(PLUS(A,B),C) PLUS(A,MINUS(B,C)))

21 FROM 20,2   -R(MINUS(PLUS(A,B),C) Y7)  -R(Y7 PLUS(A,MINUS(B,C)))  /-R
(MINUS(PLUS(A,B),C) PLUS(A,MINUS(B,C)))/

34 FROM 21,16   -R(X14 Y14)  -R(MINUS(PLUS(A,B),C) MINUS(Y14,V14))  /-R(
MINUS(PLUS(A,B),C) MINUS(Y14,V14))/  -R(MINUS(Y14,V14) PLUS(A,MINUS(B,C)
))  /-R(MINUS(PLUS(A,B),C) PLUS(A,MINUS(B,C)))/

35 FROM 34,1   -R(MINUS(PLUS(A,B),C) MINUS(PLUS(Y1,Y1),V14))  /-R(MINUS(
PLUS(A,B),C) MINUS(PLUS(Y1,Y1),V14))/  -R(MINUS(PLUS(Y1,Y1),V14) PLUS(A,
MINUS(B,C)))  /-R(MINUS(PLUS(A,B),C) PLUS(A,MINUS(B,C)))/

36 FROM 35,10   -R(MINUS(PLUS(B,A),C) PLUS(A,MINUS(B,C)))  /-R(MINUS(PLU
S(A,B),C) PLUS(A,MINUS(B,C)))/

37 FROM 36,2   -R(MINUS(PLUS(B,A),C) Y7)  -R(Y7 PLUS(A,MINUS(B,C)))  /-R
(MINUS(PLUS(B,A),C) PLUS(A,MINUS(B,C)))/  /-R(MINUS(PLUS(A,B),C) PLUS(A,
MINUS(B,C)))/

38 FROM 37,6   -R(PLUS(MINUS(B,C),A) PLUS(A,MINUS(B,C)))  /-R(MINUS(PLUS
(B,A),C) PLUS(A,MINUS(B,C)))/  /-R(MINUS(PLUS(A,B),C) PLUS(A,MINUS(B,C))
)/

39 FROM 38,1 **CLAUSE*VIDE**
```

```
EXEMPLE LANG1

TEMPS=1078,00 MILLISECONDES
                    PREUVE*DE*NIVEAU*13
MAXEN*MAXBLITS*MAXALITS
10      10         10
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
         17        15              2          3

26 AX14   +C(6 7)

12 AX5   +GC(X11 Y11 S(0))  -C(X11 Y11)

25 AX13   +C(5 6)

15 AX6   +GC(X13 Z13 S(U13))  -C(X13 Y13)  -GC(Y13 Z13 U13)

24 AX12   +B(4 5)

7 AX3   +GB(X6 Y6 S(0))  -B(X6 Y6)

23 AX11   +B(3 4)

10 AX4   +GB(X8 Z8 S(U8))  -B(X8 Y8)  -GB(Y8 Z8 U8)

22 AX10   +A(2 3)

2 AX1   +GA(X1 Y1 S(0))  -A(X1 Y1)

21 AX9   +A(1 2)

5 AX2   +GA(X3 Z3 S(U3))  -A(X3 Y3)  -GA(Y3 Z3 U3)

19 AX7   +GS(X19 U19)  -GA(X19 Y19 U19)  -GB(Y19 Z19 U19)  -GC(Z19 U19 U
19)

20 TH   -GS(1 7)

27 FROM 20,19   -GA(1 Y19 U19)  -GB(Y19 Z19 U19)  -GC(Z19 7 U19)  /-GS(
1 7)/

30 FROM 27,5   -A(1 Y3)  -GA(Y3 Y19 U3)  /-GA(1 Y19 S(U3))/  -GB(Y19 Z19
 S(U3))  -GC(Z19 7 S(U3))  /-GS(1 7)/

31 FROM 30,21   -GA(2 Y19 U3)  /-GA(1 Y19 S(U3))/  -GB(Y19 Z19 S(U3))  -
GC(Z19 7 S(U3))  /-GS(1 7)/

32 FROM 31,2   -A(2 Y19)  /-GA(2 Y19 S(0))/  /-GA(1 Y19 S(S(0)))/  -GB(Y
19 Z19 S(S(0)))  -GC(Z19 7 S(S(0)))  /-GS(1 7)/

33 FROM 32,22   -GB(3 Z19 S(S(0)))  -GC(Z19 7 S(S(0)))  /-GC(1
7)/

34 FROM 33,19   -A(3 Y3)  -GB(Y8 Z19 S(0))  /-GB(3 Z19 S(S(0)))/  -GC(Z1
9 7 S(S(0)))  /-GS(1 7)/
```

```
35 FROM 34,23   -CR(4 Z10 S(0))  /-CR(3 Z10 S(S(0)))/  -CC(Z10
7 S(S(0)))  /-CS(1 7)/

36 FROM 35,7   -R(4 Z10)  /-CR(4 Z10 S(0))/  /-CR(3 Z10 S(S(0)))/  -CC(7
10 7 S(S(0)))  /-CS(1 7)/

37 FROM 36,24   -CC(5 7 S(S(0)))  /-CS(1 7)/

38 FROM 37,15   -C(5 Y13)  -CC(Y13 7 S(0))  /-CC(5 7 S(S(0)))/  /-CS(

1 7)/

39 FROM 38,25   -CC(5 7 S(0))  /-CC(5 7 S(S(0)))/  /-CS(1 7)/

40 FROM 39,12   -C(5 7)  /-CC(5 7 S(0))/  /-CC(5 7 S(S(0)))/  /-CS(
1 7)/

41 FROM 40,29 **CLAUSE*VIDE**



EXEMPLE LANG2

TEMPS=1204,04 MILLISECONDES
                    PREUVE*DE**NIVEAU*13
MAXEN*MAXBLITS*MAXALITS
10      10        10
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
        17        15              2         3

16 AX10   +E(5 7 C)

2 AX1   +CE(X1 Y1 V1 S(0))  -E(X1 Y1 V1)

15 AX9   +E(5 5 C)

5 AX2   +CE(X3 Z3 V3 S(U3))  -E(X3 Y3 V3)  -CE(Y3 Z3 V3 U3)

14 AX8   +E(4 5 B)

13 AX7   +E(3 4 B)

12 AX6   +E(2 3 A)

11 AX5   +E(1 2 A)

3 AX3   +CS(Y5 U5)  -CE(Y5 Y5 A U5)  -CE(Y5 Z5 B U5)  -CE(Z5 U5 C U5)

10 TH   -CS(1 7)

17 FROM 10,3   -CE(1 Y5 A U5)  -CE(Y5 Z5 B U5)  -CE(Z5 7 C U5)  /-CS(
1 7)/
```

```
20 FROM 17,5   -P(1 Y3 A)  -GP(Y3 Y6 A U3)  /-GP(1 Y6 A S(U3))/  -GP(Y6
Z6 B S(U3))  -GP(Z6 7 C S(U3))  /-GS(1 7)/

21 FROM 20,11   -GP(2 Y6 A U3)  /-GP(1 Y6 A S(U3))/  -GP(Y6 Z6 B S(U3))
-GP(Z6 7 C S(U3))  /-GS(1 7)/

22 FROM 21,2   -P(2 Y6 A)  /-GP(2 Y6 A S(0))/  /-GP(1 Y6 A S(S(
0)))/  -GP(Y6 Z6 B S(S(0)))  -GP(Z6 7 C S(S(0)))  /-GS(1 7)/

23 FROM 22,12   -GP(3 Z6 B S(S(0)))  -GP(Z6 7 C S(S(0)))  /-GS(
1 7)/

24 FROM 23,5   -P(3 Y3 B)  -GP(Y3 Z6 B S(0))  /-GP(3 Z6 B S(S(
0)))/  -GP(Z6 7 C S(S(0)))  /-GS(1 7)/

25 FROM 24,13   -GP(4 Z6 B S(0))  /-GP(3 Z6 B S(S(0)))/  -GP(Z6
7 C S(S(0)))  /-GS(1 7)/

26 FROM 25,2   -P(4 Z6 B)  /-GP(4 Z6 B S(0))/  /-GP(3 Z6 B S(S(
0)))/  -GP(Z6 7 C S(S(0)))  /-GS(1 7)/

27 FROM 26,14   -GP(5 7 C S(S(0)))  /-GS(1 7)/

28 FROM 27,3   -P(5 Y3 C)  -GP(Y3 7 C S(0))  /-GP(5 7 C S(S(0)))/  /-GS(
1 7)/

29 FROM 28,15   -GP(6 7 C S(0))  /-GP(5 7 C S(S(0)))/  /-GS(1
7)/

30 FROM 29,2   -P(6 7 C)  /-GP(6 7 C S(0))/  /-GP(5 7 C S(S(0)))/  /-GS(
1 7)/

31 FROM 30,16 **CLAUSE*VIDE**



EXEMPLE GROUP6

TEMPS= 406,34 MILLISECONDES
                    PREUVE*DE*NIVEAU*4
MAXFN*MAXPLITS*MAXALITS
3       10        10
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
        7         7              3         3

3 AX3   +P(X3 G(X3) E)

45 AX20   +S(A)

41 AX18   +S(Z39)  -S(X38)  -S(Y38)  -P(X38 G(Y38) Z39)
```

```
46 TH   -S(E)

50 FROM 46,41   -S(Y38)  -S(Y38)  -P(Y38 G(Y38) E)  /-S(E)/

51 FROM 50,45   -S(Y38)  -P(A G(Y38) E)  /-S(E)/

52 FROM 51,45   -P(A G(A) E)  /-S(E)/

53 FROM 52,3 **CLAUSE*VIDE**


EXEMPLE PRIM2

TEMPS=1157,41 MILLISECONDES
                        PREUVE*DE*NIVEAU*E
MAXEN*MAXBLITS*MAXALITS
3       4         4
CLAIMES*ESSAYEES*ACCEPTEES*POINT*UNITILES*UNITAIDES
          23        15              5         8

10 AX11   +P(X10)  +P(H(X10))

12 AX7   +L(Y12 E(Y12))

1 AX1   -L(Y1 X1)

14 TH   +L(E(A) Y14)  -L(A X14)  -P(X14)

23 FROM 14,1   -L(A E(A))  -P(E(A))

24 FROM 23,12   -P(E(A))

26 FROM 24,10   +P(H(E(A)))  /-P(E(A))/

17 AX10   +P(X17)  +P(H(Y17) X17)

25 FROM 24,17   +P(H(E(A)) E(A))  /-P(E(A))/

10 AX6   +L(Y10 X10)  -P(Y10 E(Y10))

21 AX12   +P(Y21)  +L(H(Y21) Y21)

27 FROM 24,21   +L(H(E(A)) E(A))  /-P(E(A))/

2 AX2   -L(Y2 Y2)  -L(Y2 X2)

28 FROM 14,2   -L(X14 E(A))  /+L(E(A) Y14)/  -L(A X14)  -P(X14)

29 FROM 28,27   -L(A H(E(A)))  -P(H(E(A)))

35 FROM 29,10   -P(H(E(A)) E(A))  /-L(A H(E(A)))/  -P(H(E(A)))
```

```
36 FROM 35,25   -P(H(F(A)))

37 FROM 36,20  **CLAUSE*VIDE**


EXEMPLE FORM

TEMPS=0572,00 MILLISECONDES
                        PREUVE*DE*NIVEAU*15
MAXFN*MAXBLITS*MAXALITS
4       4       4
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
        71      31              1?      2

4 AX2   -Q(Y4 G(X4,Y4))  +P(Y4 G(Y4,Y4))  -P(X4 Y4)

15 AX6   +Q(G(X14,Y14) G(X14,Y14))  +Q(X14 Y14)

12 AX5   +P(X12 Y12)  -Q(G(X12,Y12) G(X12,Y12))

5 AX2   +P(X4 G(X4,Y4))  -Q(Y4 G(X4,Y4))  -P(X4 Y4)

7 AX3   +Q(G(X7,Y7) G(X7,Y7))  -P(G(X7,Y7) G(X7,Y7))  -P(X7 Y7)

16 AX7   -Q(X16 Y16)  -Q(G(X16,Y16) G(X16,Y16))

1 AX1   +Q(Y1 G(X1,Y1))  -P(X1 G(X1,Y1))  -P(X1 Y1)

18 FROM 1,15   -Q(G(Y1,G(X1,Y1)) G(Y1,G(X1,Y1)))  /+Q(Y1 G(X1,Y1))/  -P(
X1 G(X1,Y1))  -P(X1 Y1)

21 FROM 18,7   -P(G(Y1,G(X1,Y1)) G(Y1,G(X1,Y1)))  -P(Y1 G(X1,Y1))  /-Q(G
(Y1,G(X1,Y1)) G(Y1,G(X1,Y1)))/  /+Q(Y1 G(X1,Y1))/  -P(X1 G(X1,Y1))  -P(X
1 Y1)

23 FROM 21,12   -Q(G(G(Y1,G(X1,Y1)),G(Y1,G(X1,Y1))) G(G(Y1,G(X1,Y1)),G(Y
1,G(X1,Y1))))  /-P(G(Y1,G(X1,Y1)) G(Y1,G(X1,Y1)))/  -P(Y1 G(X1,Y1))  /-Q
(G(Y1,G(X1,Y1)) G(Y1,G(Y1,Y1)))/  /+Q(Y1 G(X1,Y1))/  -P(X1 G(X1,Y1))  -P
(X1 Y1)

28 FROM 23,15   +Q(G(Y1,G(X1,Y1)) G(Y1,G(X1,Y1)))  /-Q(G(G(Y1,G(X1,Y1)),
G(Y1,G(X1,Y1))) G(G(Y1,G(X1,Y1)),G(Y1,G(X1,Y1))))/  /-P(G(Y1,G(X1,Y1)) G
(Y1,G(X1,Y1)))/  -P(Y1 G(X1,Y1))  /-Q(G(Y1,G(X1,Y1)) G(Y1,G(X1,Y1)))/  /
+Q(Y1 G(X1,Y1))/  -P(X1 G(X1,Y1))  -P(X1 Y1)

29 FROM 28,PRODUC   -P(Y1 G(X1,Y1))  /-Q(G(Y1,G(X1,Y1)) G(Y1,G(X1,Y1)))/

/+Q(Y1 G(X1,Y1))/  -P(X1 G(X1,Y1))  -P(X1 Y1)

31 FROM 29,5   -P(X1 G(X1,X1))  -P(X1 X1)

33 FROM 31,12   -Q(G(X1,G(X1,X1)) G(X1,G(X1,X1)))  /-P(X1 G(X1,X1))/  -P
```

```
(X1 X1)

38 FROM 33,15   +P(X1 G(X1,X1))  /-P(G(X1,G(X1,X1)) G(X1,G(X1,X1)))/  /-
P(X1 G(X1,X1))/  -P(X1 X1)

44 FROM 38,4   +P(X1 G(X1,X1))  -P(X1 X1)  /+P(X1 G(X1,X1))/  /-P(G(X1,G
(X1,X1)) G(X1,G(X1,X1)))/  /-P(X1 G(X1,X1))/  -P(X1 X1)

45 FROM 44,REDUC   -P(X1 X1)  /+P(X1 G(X1,X1))/  /-P(G(X1,G(X1,X1)) G(X1
,G(X1,X1)))/  /-P(X1 G(X1,X1))/  -P(X1 X1)

46 FROM 45,FACTEUR   -P(X1 X1)

10 AX4   +P(X10 Y10)  +P(G(X10,Y10) G(X10,Y10))

47 FROM 46,10   +P(G(X1,X1) G(Y1,X1))  /-P(X1 X1)/

48 FROM 47,46 **CLAUSE*VIDE**


EXEMPLE CORPS

TEMPS=100850,35 MILLISECONDES
                    PREUVE*DE*NIVEAU*1°
MAXEM*MAXBLITS*MAXALITS
3      4        6
CHAINES*ESSAYEES*ACCEPTEES*DONT*UNITILES*UNITAIRES
         686       263             245       22

21 TH1   +S(A)

14 AX7   -P(X14 X14 D)  -S(Y14)

12 AX5   +P(Y12 F(X12) E)  +P(Y12 X12 D)

20 AX3   +P(Y10 X10 U10)  -P(X10 Y10 U10)

22 AX2   -P(E E D)

16 AX11   -P(X16 X16 D)  +P(U16 U16 D)  -P(Y16 Y16 U16)

11 AX4   +P(X10 G(Y10) G(U10))  -P(X10 Y10 U10)

4 AX12   +S(U4)  -P(X4 Y4 U4)  -S(Y4)  -S(X4)

23 AX1   +P(X23 E Y23)

9 AX9   +P(G(X9) G(Y9) U9)  -P(X9 Y9 U9)

24 TH2   -S(F(A))

7 AX13   -S(Y4)  +S(U4)  -P(Y4 Y4 U4)  -S(Y4)

5 AX10   +S(X1)  +S(G(Y1))  +P(Y1 X1 D)
```

```
25 FROM 24,3   +S(G(F(A)))  +P(F(A) F(A) D)  /-S(F(A))/

37 FROM 25,7   +S(U4)  -P(G(F(A)) Y4 U4)  -S(Y4)  /+S(G(F(A)))/  +P(F(A)
 F(A) D)  /-S(F(A))/

39 FROM 37,24   -P(G(F(A)) Y4 F(A))  -S(Y4)  /+S(G(F(A)))/  +P(F(A) F(A)
 D)  /-S(F(A))/

43 FROM 39,9   -P(F(A) Y8 F(A))  /-P(G(F(A)) G(Y8) F(A))/  -S(G(Y8))  /+
S(G(F(A)))/  +P(F(A) F(A) D)  /-S(F(A))/

44 FROM 43,23   -S(G(E))  /+S(G(F(A)))/  +P(F(A) F(A) D)  /-S(F(A))/

112 FROM 44,4   -P(X4 Y4 G(E))  -S(Y4)  -S(X4)  /-S(G(E))/  /+S(G(F(A)))
/  +P(F(A) F(A) D)  /-G(F(A))/

180 FROM 112,11   -P(Y4 Y10 E)  /-P(Y4 G(Y10) G(E))/  -S(G(Y10))  -S(X4)
  /-S(G(E))/  /+S(G(F(A)))/  +P(F(A) F(A) D)  /-S(F(A))/

192 FROM 180,12   +P(X4 X4 D)  /-P(X4 F(Y4) E)/  /-P(Y4 G(F(X4)) G(E))/
-S(G(F(X4)))  -S(X4)  /-S(G(E))/  /+S(G(F(A)))/  +P(F(A) F(A) D)  /-S(F(
A))/

237 FROM 192,14   -S(X4)  /+P(X4 X4 D)/  /-P(X4 F(X4) E)/  /-P(Y4 G(F(X4
)) G(E))/  -S(G(F(X4)))  -S(X4)  /-S(G(E))/  /+S(G(F(A)))/  +P(F(A) F(A)
 D)  /-S(F(A))/

238 FROM 237,FACTEUR   -S(G(F(X4)))  -S(X4)  /-S(G(E))/  /+S(G(F(A)))/  +
P(F(A) F(A) D)  /-S(F(A))/

239 FROM 238,REDUC   -S(A)  /-S(G(E))/  /+S(G(F(A)))/  +P(F(A) F(A) D)  /
-S(F(A))/

240 FROM 239,21   +P(F(A) F(A) D)  /-S(F(A))/

244 FROM 240,16   +P(U16 U16 D)  -P(F(A) Y16 U16)  /+P(F(A) F(A) D)/  /-
S(F(A))/

245 FROM 244,22   -P(F(A) Y16 E)  /+P(F(A) F(A) D)/  /-S(F(A))/

247 FROM 245,20   -P(Y16 F(A) E)  /-P(F(A) Y16 E)/  /+P(F(A) F(A) D)/  /
-S(F(A))/

265 FROM 247,12   +P(A A D)  /-P(A F(A) E)/  /-P(F(A) A E)/  /+P(F(A) F(
A) D)/  /-S(F(A))/

286 FROM 265,14   -S(A)  /+P(A A D)/  /-P(A F(A) E)/  /-P(F(A) A E)/  /+
P(F(A) F(A) D)/  /-S(F(A))/

287 FROM 286,21 **CLAUSE*VIDE**
```

```
EXEMPLE GROUP3

TEMPS= 418,80 MILLISECONDES
                        PREUVE*DE*NIVEAU*4
MAXEN*MAXBLITS*MAXALITS
4       3         1
CHAINES*ESSAYEES*ACCEPTEES*POINT*INUTILES*UNITAIRES
         5         5                1         2

12 TH    -P(K(X12) X12 K(Y12))

10 SOLUTG    +P(G(Y10,Y10) X10 Y10)

9 SOLUTD    +P(X9 H(X9,Y9) Y9)

1 ASSOC1    -P(X1 Y1 U1)  -P(Y1 Z1 V1)  -P(X1 V1 W1)  +P(U1 Z1 W1)

13 FROM 10,1    -P(X10 Z1 V1)  -P(G(Y10,Y10) V1 W1)  +P(Y10 Z1 W1)  /+P(G
(X10,Y10) X10 Y10)/

14 FROM 13,9    -P(G(X10,Y10) V1 W1)  +P(Y10 H(Y10,V1) W1)  /+P(G(Y10,Y10
) X10 Y10)/

16 FROM 14,10    +P(U1 H(V1,V1) W1)  /+P(G(V1,W1) V1 W1)/

17 FROM 16,12 **CLAUSE*VIDE**


EXEMPLE GROUP4

TEMPS=2694,78 MILLISECONDES
                        PREUVE*DE*NIVEAU*10
MAXEN*MAXBLITS*MAXALITS
3       3         3
CHAINES*ESSAYEES*ACCEPTEES*POINT*INUTILES*UNITAIRES
         30        20               10         6

1 AX1    +P(E(X1) X1 E)

2 AX2    +P(E X2 X2)

3 ASSOC2    +P(X8 V8 W8)  -P(U8 Z8 W8)  -P(Y8 Z8 V8)  -P(X8 Y8 U8)

4 ASSOC1    +P(U4 Z4 W4)  -P(Y4 V4 W4)  -P(Y4 Z4 V4)  -P(Y4 Y4 U4)

3 TH    -P(A E A)

14 FROM 3,8    -P(U8 Z8 A)  -P(Y8 Z8 E)  -P(A Y8 U8)  /-P(A E A)/


15 FROM 14,2    -P(Y8 A E)  -P(A Y8 E)  /-P(A E A)/

16 FROM 15,1    -P(A E(A) E)  /-P(A E A)/
```

```
17 FROM 16,4   -P(X4 V4 E)  -P(Y4 F(A) V4)  -P(X4 Y4 A)  /-P(A F(A) E)/
/-P(A E A)/

18 FROM 17,1   -P(Y4 F(A) V4)  -P(E(V4) Y4 A)  /-P(A F(A) E)/  /-P(A E A
)/

20 FROM 18,2   -P(E(F(A)) E A)  /-P(A E(A) E)/  /-P(A E A)/

37 FROM 20,8   -P(U8 Z8 A)  -P(Y8 Z8 E)  -P(E(F(A)) Y8 U8)  /-P(E(F(A))
E A)/  /-P(A F(A) E)/  /-P(A E A)/

38 FROM 37,2   -P(Y8 A E)  -P(E(F(A)) Y8 E)  /-P(E(F(A)) E A)/  /-P(A F(
A) E)/  /-P(A E A)/

39 FROM 38,1   -P(E(F(A)) F(A) E)  /-P(E(F(A)) E A)/  /-P(A F(A) E)/  /-
P(A E A)/

40 FROM 39,1 **CLAUSE*VIDE**


EXEMPLE GROUP4 BIS

TEMPS=7613,07 MILLISECONDES
                    PREUVE*DE*NIVEAU*10
MAXEN*MAXBLITS*MAXALITS
3       5          2
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
          51         49              39         1

2 AX2   +P(X2 E X2)

1 AX1   +P(X1 F(X1) E)

4 ASSOC1   +P(U4 Z4 V4)  -P(X4 V4 U4)  -P(Y4 Z4 V4)  -P(Y4 Y4 U4)

8 ASSOC2   +P(Y8 V8 U8)  -P(U8 Z8 U8)  -P(Y8 Z8 V8)  -P(X8 Y8 U8)

3 TH   -P(E A A)

14 FROM 3,8   -P(U8 Z8 A)  -P(Y8 Z8 A)  -P(E Y8 U8)  /-P(E A A)/

27 FROM 14,4   -P(X4 V4 A)  -P(Y4 Z8 V4)  -P(X4 Y4 U8)  /-P(U8 Z8 A)/  -
P(Y8 Z8 A)  -P(E Y8 U8)  /-P(E A A)/

28 FROM 27,2   -P(Y4 Z8 E)  -P(A Y4 U8)  /-P(U8 Z8 A)/  -P(Y8 Z8 A)  -P(
E Y8 U8)  /-P(E A A)/

29 FROM 28,1   -P(A Y4 U8)  /-P(U8 F(Y4) A)/  -P(Y8 F(Y4) A)  -P(E Y8 U8
)  /-P(E A A)/
```

```
30 FROM 29,1   -P(Y8 F(F(A)) A)  -P(E Y8 E)  /-P(E A A)/

50 FROM 30,4   -P(X4 V4 A)  -P(Y4 F(F(A)) V4)  -P(V4 Y4 Y8)  /-P(Y8 F(F(
A)) A)/  -P(E Y8 E)  /-P(E A A)/

57 FROM 56,2   -P(Y4 F(F(A)) E)  -P(A Y4 Y8)  /-P(Y8 F(F(A)) A)/  -P(E Y
8 E)  /-P(E A A)/

58 FROM 57,1   -P(A F(A) Y8)  /-P(Y8 F(F(A)) A)/  -P(E Y8 E)  /-P(E A A)
/

59 FROM 58,1   -P(E E E)  /-P(E A A)/

60 FROM 59,2 **CLAUSE*VIDE**


EXEMPLE GROUPE

TEMPS=1000,37 MILLISECONDES
                    PREUVE*DE*NIVEAU*7
MAXEN*MAXBLITS*MAXALITS
3       5         2
CLAUSES*ESSAYEES*ACCEPTEES*DOUT*UNITES*UNITAIRES
          10        16              9         1

2 AX2   +P(X2 E X2)

1 AX1   +P(X1 F(X10) E)

4 ASSOC1   +P(U4 Z4 V4)  -P(X4 V4 U4)  -P(Y4 Z4 V4)  -P(Y4 Y4 U4)

8 ASSOC2   +P(X8 Y8 U8)  -P(U8 Z8 U8)  -P(Y8 Z8 V8)  -P(X8 Y8 U8)

3 TH   -P(X3 A E)

15 FROM 3,2   -P(U8 Z8 E)  -P(Y8 Z8 A)  -P(X3 Y8 U8)  /-P(X3 A E)/

16 FROM 15,1   -P(Y8 F(U8) A)  -P(X3 Y8 U8)  /-P(X3 A E)/

23 FROM 16,4   -P(Y4 V4 A)  -P(Y4 F(U8) V4)  -P(Y4 Y4 Y8)  /-P(Y8 F(U8)
A)/  -P(X3 Y8 U8)  /-P(X3 A E)/

24 FROM 23,2   -P(Y4 F(U8) E)  -P(A Y4 Y8)  /-P(Y8 F(U8) A)/  -P(X3 Y8 U
8)  /-P(X3 A E)/

25 FROM 24,1   -P(A U8 Y8)  /-P(Y8 F(U8) A)/  -P(X3 Y8 U8)  /-P(X3 A E)/


26 FROM 25,1   -P(X3 E F(A))  /-P(X3 A E)/

27 FROM 26,2 **CLAUSE*VIDE**
```

```
EXEMPLE HAS-PARTS1

TEMPS= 463,70 MILLISECONDES
                    PREUVE*DE*NIVEAU*6
MAXEN*MAXPLITS*MAXALITS
3       4         4
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
         10        10              0         3

8 AX4   +IN(JOHN BOY)

7 AX3   +IN(X6 HUMAN)  -IN(X5 BOY)

5 AX2   +HP(X4 2 ARM)  -IN(X4 HUMAN)

3 AX1   +HP(X1 T(U1,V1) Z1)  -HP(X1 U1 Y1)  -HP(F(V1,U1,Z1,Y1,X1) V1 Z1)


14 TH   -HP(JOHN T(2,1) HAND)

15 FROM 14,3   -HP(JOHN 2 Y1)  -HP(F(1,2,HAND,Y1,JOHN) 1 HAND)  /-HP(JOH
N T(2,1) HAND)/

19 FROM 15,5   -IN(JOHN HUMAN)  /-HP(JOHN 2 ARM)/  -HP(F(1,2,HAND,ARM,JO
HN) 1 HAND)  /-HP(JOHN T(2,1) HAND)/

19 FROM 18,7   -IN(JOHN BOY)  /-IN(JOHN HUMAN)/  /-HP(JOHN 2 ARM)/  -HP(
F(1,2,HAND,ARM,JOHN) 1 HAND)  /-HP(JOHN T(2,1) HAND)/

20 FROM 19,8   -HP(F(1,2,HAND,ARM,JOHN) 1 HAND)  /-HP(JOHN T(2,
1) HAND)/

9 AX5   -IN(X9 ARM)  +HP(X9 1 HAND)

13 AX6   +HP(Y11 T(U11,V11) Z11)  -HP(X11 U11 Y11)  +IN(F(V11,U11,Z11,Y1
1,Y11) Y11)

18 FROM 14,13   -HP(JOHN 2 Y11)  +IN(F(1,2,HAND,Y11,JOHN) Y11)  /-HP(JOH
N T(2,1) HAND)/

17 FROM 16,5   -IN(JOHN HUMAN)  /-HP(JOHN 2 ARM)/  +IN(F(1,2,HAND,ARM,JO
HN) ARM)  /-HP(JOHN T(2,1) HAND)/

21 FROM 17,7   -IN(JOHN BOY)  /-IN(JOHN HUMAN)/  /-HP(JOHN 2 ARM)/  +IN(
F(1,2,HAND,ARM,JOHN) ARM)  /-HP(JOHN T(2,1) HAND)/

22 FROM 21,8   +IN(F(1,2,HAND,ARM,JOHN) ARM)  /-HP(JOHN T(2,1) HAND)/

23 FROM 22,9   +HP(F(1,2,HAND,ARM,JOHN) 1 HAND)  /+IN(F(1,2,HAND,ARM,JOH
N) ARM)/  /-HP(JOHN T(2,1) HAND)/

24 FROM 23,20 **CLAUSE*VIDE**
```

```
EXEMPLE MAS-PARTS2

TEMPS=2741,70 MILLISECONDES
                    PREUVE*DE*NIVEAU*13
MAXEN*MAXBLITS*MAXALITS
3       3       6
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
        31      28              5       3

8 AX4   +IN(JOHN BOY)

7 AX3   +IN(X6 HUMAN)  -IN(X6 BOY)

5 AX2   +HP(X4 2 ARM)  -IN(X4 HUMAN)

12 AX6  +IN(F(Y11,U11,Z11,Y11,X11) Y11)  -HP(X11 U11 Y11)  +HP(X11 T(U1
1,V11) Z11)

10 AX5  +HP(X9 1 HAND)  -IN(X9 ARM)

7 AX1   +HP(Y1 T(U1,V1) Z1)  -HP(X1 U1 Y1)  -HP(F(V1,U1,Z1,Y1,Y1) V1 Z1)


13 AX6  +HP(X11 T(U11,V11) Z11)  -HP(Y11 U11 Y11)  +IN(F(V11,U11,Z11,Y1
1,X11) Y11)

10 TH   -HP(JOHN T(T(2,1),5) FINGERS)

18 FROM 10,13  -HP(JOHN T(2,1) Y11)  +IN(F(5,T(2,1),FINGERS,Y11,JOHN) Y
11)  /-HP(JOHN T(T(2,1),5) FINGERS)/

19 FROM 18,3  -HP(JOHN 2 Y1)  -HP(F(1,2,Y11,Y1,JOHN) 1 Y11)  /-HP(JOHN
T(2,1) Y11)/  +IN(F(5,T(2,1),FINGERS,Y11,JOHN) Y11)  /-HP(JOHN T(T(
2,1),5) FINGERS)/

22 FROM 19,5  -IN(JOHN HUMAN)  /-HP(JOHN 2 ARM)/  -HP(F(1,2,Y11,ARM,JOH
N) 1 Y11)  /-HP(JOHN T(2,1) Y11)/  +IN(F(5,T(2,1),FINGERS,Y11,JOHN) Y11)
  /-HP(JOHN T(T(2,1),5) FINGERS)/

23 FROM 22,7  -IN(JOHN BOY)  /-IN(JOHN HUMAN)/  /-HP(JOHN 2 ARM)/  -HP(
F(1,2,Y11,ARM,JOHN) 1 Y11)  /-HP(JOHN T(2,1) Y11)/  +IN(F(5,T(
2,1),FINGERS,Y11,JOHN) Y11)  /-HP(JOHN T(T(2,1),5) FINGERS)/

24 FROM 23,8  -HP(F(1,2,Y11,ARM,JOHN) 1 Y11)  /-HP(JOHN T(2,1) Y11)/  +
IN(F(5,T(2,1),FINGERS,Y11,JOHN) Y11)  /-HP(JOHN T(T(2,1),5) FINGERS)/

28 FROM 24,19  -IN(F(1,2,HAND,ARM,JOHN) ARM)  /-HP(F(1,2,HAND,ARM,JOHN)
 1 HAND)/  /-HP(JOHN T(2,1) HAND)/  +IN(F(5,T(2,1),FINGERS,HAND,JOHN) HA
ND)  /-HP(JOHN T(T(2,1),5) FINGERS)/

29 FROM 28,12  -HP(JOHN 2 ARM)  +HP(JOHN T(2,1) HAND)  /-IN(F(
1,2,HAND,ARM,JOHN) ARM)/  /-HP(F(1,2,HAND,ARM,JOHN) 1 HAND)/  /-HP(JOHN
T(2,1) HAND)/  +IN(F(5,T(2,1),FINGERS,HAND,JOHN) HAND)  /-HP(JOHN T(T(
2,1),5) FINGERS)/
```

```
37 FROM 29,5   -IN(JOHN HUMAN)  /-HP(JOHN 2 ARM)/  +HP(JOHN T(
2,1) HAND)  /-IN(F(1,2,HAND,ARM,JOHN) ARM)/  /-HP(F(1,2,HAND,ARM,JOHN)
1 HAND)/  /-HP(JOHN T(2,1) HAND)/  +IN(F(5,T(2,1),FINGERS,HAND,JOHN) HAN
D)  /-HP(JOHN T(T(2,1),5) FINGERS)/

38 FROM 37,7   -IN(JOHN BOY)  /-IN(JOHN HUMAN)/  /-HP(JOHN 2 ARM)/  +HP(
JOHN T(2,1) HAND)  /-IN(F(1,2,HAND,ARM,JOHN) ARM)/  /-HP(F(1,2,HAND,ARM,
JOHN) 1 HAND)/  /-HP(JOHN T(2,1) HAND)/  +IN(F(5,T(2,1),FINGERS,HAND,JOH
N) HAND)  /-HP(JOHN T(T(2,1),5) FINGERS)/

39 FROM 38,2   +HP(JOHN T(2,1) HAND)  /-IN(F(1,2,HAND,ARM,JOHN) ARM)/  /
-HP(F(1,2,HAND,ARM,JOHN) 1 HAND)/  /-HP(JOHN T(2,1) HAND)/  +IN(F(
5,T(2,1),FINGERS,HAND,JOHN) HAND)  /-HP(JOHN T(T(2,1),5) FINGERS)/

40 FROM 39,REDUC   +IN(F(5,T(2,1),FINGERS,HAND,JOHN) HAND)  /-HP(JOHN T(
T(2,1),5) FINGERS)/

15 AX7   +HP(X14 5 FINGERS)  -IN(X14 HAND)

2 AX1   -HP(F(V1,U1,Z1,Y1,X1) V1 Z1)  -HP(Y1 U1 Y1)  +HP(Y1 T(U1,V1) Z1)


9 AX5   -IN(X9 ARM)  +HP(X9 1 HAND)

17 FROM 16,3   -HP(JOHN T(2,1) Y1)  -HP(F(5,T(2,1),FINGERS,Y1,JOHN)
5 FINGERS)  /-HP(JOHN T(T(2,1),5) FINGERS)/

20 FROM 17,13   -HP(JOHN 2 Y11)  +IN(F(1,2,Y1,Y11,JOHN) Y11)  /-HP(JOHN
T(2,1) Y1)/  -HP(F(5,T(2,1),FINGERS,Y1,JOHN) 5 FINGERS)  /-HP(JOHN T(T(
2,1),5) FINGERS)/

21 FROM 20,5   -IN(JOHN HUMAN)  /-HP(JOHN 2 ARM)/  +IN(F(1,2,Y1,ARM,JOHN
) ARM)  /-HP(JOHN T(2,1) Y1)/  -HP(F(5,T(2,1),FINGERS,Y1,JOHN)
5 FINGERS)  /-HP(JOHN T(T(2,1),5) FINGERS)/

25 FROM 21,7   -IN(JOHN BOY)  /-IN(JOHN HUMAN)/  /-HP(JOHN 2 ARM)/  +IN(
F(1,2,Y1,ARM,JOHN) ARM)  /-HP(JOHN T(2,1) Y1)/  -HP(F(5,T(2,1),FINGERS,Y
1,JOHN) 5 FINGERS)  /-HP(JOHN T(T(2,1),5) FINGERS)/

26 FROM 25,8   +IN(F(1,2,Y1,ARM,JOHN) ARM)  /-HP(JOHN T(2,1) Y1)/  -HP(F
(5,T(2,1),FINGERS,Y1,JOHN) 5 FINGERS)  /-HP(JOHN T(T(2,1),5) FINGERS)/

27 FROM 26,2   +HP(F(1,2,Y1,ARM,JOHN) 1 HAND)  /+IN(F(1,2,Y1,ARM,JOHN) A
RM)/  /-HP(JOHN T(2,1) Y1)/  -HP(F(5,T(2,1),FINGERS,Y1,JOHN) 5 FINGERS)
/-HP(JOHN T(T(2,1),5) FINGERS)/

31 FROM 27,2   -HP(JOHN 2 ARM)  +HP(JOHN T(2,1) HAND)  /+HP(F(
1,2,HAND,ARM,JOHN) 1 HAND)/  /+IN(F(1,2,HAND,ARM,JOHN) ARM)/  /-HP(JOHN
T(2,1) HAND)/  -HP(F(5,T(2,1),FINGERS,HAND,JOHN) 5 FINGERS)  /-HP(JOHN T
(T(2,1),5) FINGERS)/

34 FROM 31,5   -IN(JOHN HUMAN)  /-HP(JOHN 2 ARM)/  +HP(JOHN T(
2,1) HAND)  /+HP(F(1,2,HAND,ARM,JOHN) 1 HAND)/  /+IN(F(1,2,HAND,ARM,JOHN
) ARM)/  /-HP(JOHN T(2,1) HAND)/  -HP(F(5,T(2,1),FINGERS,HAND,JOHN)
5 FINGERS)  /-HP(JOHN T(T(2,1),5) FINGERS)/
```

```
41 FROM 34,7   -IN(JOHN BOY)  /-IN(JOHN HUMAN)/  /-HP(JOHN 2 ARM)/  +HP(
JOHN T(2,1) HAND)  /+HP(F(1,2,HAND,ARM,JOHN) 1 HAND)/  /+IN(F(
1,2,HAND,ARM,JOHN) ARM)/  /-HP(JOHN T(2,1) HAND)/  -HP(F(5,T(2,
1),FINGERS,HAND,JOHN) 5 FINGERS)  /-HP(JOHN T(T(2,1),5) FINGERS)/

42 FROM 41,8   +HP(JOHN T(2,1) HAND)  /+HP(F(1,2,HAND,ARM,JOHN)
1 HAND)/  /+IN(F(1,2,HAND,ARM,JOHN) ARM)/  /-HP(JOHN T(2,1) HAND)/  -HP(
F(5,T(2,1),FINGERS,HAND,JOHN) 5 FINGERS)  /-HP(JOHN T(T(2,1),5) FINGERS)
/

43 FROM 42,REDUC   -HP(F(5,T(2,1),FINGERS,HAND,JOHN) 5 FINGERS)  /-HP(JO
HN T(T(2,1),5) FINGERS)/

44 FROM 43,15   -IN(F(5,T(2,1),FINGERS,HAND,JOHN) HAND)  /-HP(F(
5,T(2,1),FINGERS,HAND,JOHN) 5 FINGERS)/  /-HP(JOHN T(T(2,1),5) FINGERS)/


45 FROM 44,40 **CLAUSE*VIDE**

EXEMPLE MINIPROG

TEMPS= 623,22 MILLISECONDES
                    PREUVE*DE*NIVEAU*6
MAXEN*MAXBLITS*MAXALITS
2       3       4
CHAINES*ESSAYEES*ACCEPTEES*POINT*INUTILES*UNITAIRES
        14      7               1       2

5 AX9   +FOLLOWS(P8 P3)

16 AX18   +SUCCEEDS(X15 Y15)  -FOLLOWS(X15 Y15)

1 AX4   +LABELS(LOOP P3)

6 AX14   +HAS(P8 GOTO(LOOP))

9 AX15   +SUCCEEDS(Y7 X7)  -HAS(X7 GOTO(Z7))  -LABELS(Z7 Y7)

14 AX17   +SUCCEEDS(X12 Y12)  -SUCCEEDS(X12 Z12)  -SUCCEEDS(Z12 Y12)

17 TH   -SUCCEEDS(P3 P3)

18 FROM 17,14   -SUCCEEDS(P3 Z12)  -SUCCEEDS(Z12 P3)  /-SUCCEEDS(P3 P3)/


19 FROM 18,9   -HAS(Z12 GOTO(Z7))  -LABELS(Z7 P3)  /-SUCCEEDS(P3 Z12)/  -
SUCCEEDS(Z12 P3)  /-SUCCEEDS(P3 P3)/

20 FROM 19,6   -LABELS(LOOP P3)  /-SUCCEEDS(P3 P8)/  -SUCCEEDS(P8 P3)  /
-SUCCEEDS(P3 P3)/

21 FROM 20,1   -SUCCEEDS(P8 P3)  /-SUCCEEDS(P3 P3)/

23 FROM 21,15   -FOLLOWS(P8 P3)  /-SUCCEEDS(P8 P3)/  /-SUCCEEDS(P3 P3)/

24 FROM 23,5 **CLAUSE*VIDE**
```

```
EXEMPLE PROG

TEMPS=3540,30 MILLISECONDES
                    PRELIVE*DE*NIVEAU*12
MAXEM*MAXBLITS*MAXALITS
2       3         4
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
          57        25              13        6

0 AX0    +FOLLOWS(P6 P3)

24 AX18    +SUCCEEDS(Y23 Y23)   -FOLLOWS(Y23 Y23)

11 AX11    +FOLLOWS(P7 P6)

22 AX17    +SUCCEEDS(X20 Y20)   -SUCCEEDS(X20 Z20)   -SUCCEEDS(Z20 Y20)

13 AX13    +FOLLOWS(P8 P7)

4 AX4    +LABELS(LOOP P3)

14 AX14    +HAS(P8 GOTO(LOOP))

17 AX15    +SUCCEEDS(Y15 X15)   -HAS(Y15 GOTO(Z15))   -LABELS(Z15 Y15)

25 TH    -SUCCEEDS(P3 P3)

26 FROM 25,22    -SUCCEEDS(P3 Z20)   -SUCCEEDS(Z20 P3)   /-SUCCEEDS(P3 P3)/


20 FROM 26,17    -HAS(Z20 GOTO(Z15))   -LABELS(Z15 P3)   /-SUCCEEDS(P3 Z20)
/   -SUCCEEDS(Z20 P3)   /-SUCCEEDS(P3 P3)/

30 FROM 20,14    -LABELS(LOOP P3)   /-SUCCEEDS(P3 P3)/   -SUCCEEDS(P8 P3)   /
-SUCCEEDS(P3 P3)/

31 FROM 30,4    -SUCCEEDS(P8 P3)   /-SUCCEEDS(P3 P3)/

35 FROM 31,22    -SUCCEEDS(P8 Z20)   -SUCCEEDS(Z20 P3)   /-SUCCEEDS(P8 P3)/
  /-SUCCEEDS(P3 P3)/

38 FROM 35,24    -FOLLOWS(P8 Z20)   /-SUCCEEDS(P8 Z20)/   -SUCCEEDS(Z20 P3)
  /-SUCCEEDS(P8 P3)/   /-SUCCEEDS(P3 P3)/

39 FROM 38,13    -SUCCEEDS(P7 P3)   /-SUCCEEDS(P8 P3)/   /-SUCCEEDS(P3 P3)/


44 FROM 39,22    -SUCCEEDS(P7 Z20)   -SUCCEEDS(Z20 P3)   /-SUCCEEDS(P7 P3)/
  /-SUCCEEDS(P8 P3)/   /-SUCCEEDS(P3 P3)/
```

```
46 FROM 44,24   -FOLLOWS(P7 Z20)  /-SUCCEEDS(P7 Z20)/  -SUCCEEDS(Z20 P3)
  /-SUCCEEDS(P7 P3)/  /-SUCCEEDS(P8 P3)/  /-SUCCEEDS(P3 P3)/

47 FROM 46,11   -SUCCEEDS(P6 P3)  /-SUCCEEDS(P7 P3)/  /-SUCCEEDS(P8 P3)/
  /-SUCCEEDS(P3 P3)/

49 FROM 47,24   -FOLLOWS(P6 P3)  /-SUCCEEDS(P6 P3)/  /-SUCCEEDS(P7 P3)/
/-SUCCEEDS(P8 P3)/  /-SUCCEEDS(P3 P3)/

50 FROM 49,9 **CLAUSE*VIDE**


EXEMPLE NUM12

TEMPS=176006,64 MILLISECONDES
                    PREUVE*DE*NIVEAU*11
MAXEN*MAXBLITS*MAXALITS
2       3       5
CHAINES*ESSAYEES*ACCEPTEES*DONT*INUTILES*UNITAIRES
        942     451             432     123

3 AX4   +Q(M3 A S(C) S(B))

23 AX14   -Q(M23 X23 Z23 U23)  -Q(M23 X23 Y23 U23)  +E(M23 Y23 Z23)

2 AX2   +Q(M2 B B S(B))

4 AX5   +Q(M4 X4 Y4 E(X4,Y4))

13 AX8   +Q(M12 Y12 X12 Z12)  -Q(M12 X12 Y12 Z12)

19 AX13   -Q(M19 X19 Y19 U19)  -Q(M19 Y19 Z19 V19)  -Q(M19 U19 Z19 W19)
+Q(M19 X19 V19 W19)

17 AX12   -D(M17 X17 Y17)  +Q(M(Y17,Y17) X17 QUOTIENT Y17)

6 AX6   +D(M5 X5 Z5)  -Q(M5 X5 Y5 Z5)

7 AX7   -P(X7)  -D(M7 X7 S(Y7))  +D(M7 X7 Y7)

1 AX1   +P(A)

26 FROM 1,7   -D(M7 A S(Y7))  +D(M7 A Y7)  /+P(*)/

27 FROM 26,6   -Q(M7 A Y5 S(Y7))  /-D(M7 A S(Y7))/  +D(M7 A Y7)  /+P(*)/


28 FROM 27,3   +D(M7 A B)  /+P(A)/

30 FROM 28,17   +Q(M(A,B) A QUOTIENT B)  /+D(M7 A B)/  /+P(*)/

50 FROM 30,19   -Q(M(A,B) QUOTIENT Z19 V19)  -Q(M(A,B) B Z19 W19)  +Q(M(
A,B) A V19 W19)  /+Q(M(A,B) A QUOTIENT B)/  /+D(M7 A B)/  /+P(*)/
```

```
161 FROM 50,13   -Q(H(A,B) Z19 QUOTIENT V19)  /-Q(H(A,B) QUOTIENT Z19 V1
9)/  -Q(H(A,B) B Z19 W19)  +Q(H(A,B) A V19 W19)  /+Q(H(A,B) A QUOTIENT B
)/  /+D(M7 A B)/  /+P(A)/

163 FROM 161,4   -Q(H(A,B) B Z19 W19)  +Q(H(A,B) A F(Z19,QUOTIENT) W19)
/+Q(H(A,B) A QUOTIENT B)/  /+D(M7 A B)/  /+P(A)/

164 FROM 163,2   +Q(H(A,B) A F(B,QUOTIENT) S(B))  /+Q(H(A,B) A QUOTIENT
B)/  /+D(M7 A B)/  /+P(A)/

446 FROM 164,23   -Q(H(A,B) A Y23 S(B))  +E(H(A,B) Y23 F(B,QUOTIENT))  /
+Q(H(A,B) A F(B,QUOTIENT) S(B))/  /+Q(H(A,B) A QUOTIENT B)/  /+D(M7 A B)
/  /+P(A)/

447 FROM 446,3   +E(H(A,B) S(C) F(B,QUOTIENT))  /+Q(H(A,B) A F(B,QUOTIEN
T) S(B))/  /+Q(H(A,B) A QUOTIENT B)/  /+D(M7 A B)/  /+P(A)/

9 AX7   +D(M7 X7 Y7)  -P(X7)  -D(M7 X7 S(Y7))

10 AX8   -D(M10 X10 B)  -D(M10 X10 C)

29 FROM 28,10   -D(M7 A C)  /+D(M7 A B)/  /+P(A)/

42 FROM 29,9   -P(A)  -D(M7 A S(C))  /-D(M7 A C)/  /+D(M7 A B)/  /+P(A)/


43 FROM 42,REDUC   -D(M7 A S(C))  /-D(M7 A C)/  /+D(M7 A B)/  /+P(A)/

16 AX11   -D(M14 X14 Y14)  +D(M14 X14 Z14)  -E(M14 Z14 Y14)

5 AX6   -Q(M5 X5 Y5 Z5)  +D(M5 X5 Z5)

60 FROM 50,4   -Q(H(A,B) B Z19 W19)  +Q(H(A,B) A F(QUOTIENT,Z19) W19)  /
+Q(H(A,B) A QUOTIENT B)/  /+D(M7 A B)/  /+P(A)/

62 FROM 60,4   +Q(H(A,B) A F(QUOTIENT,Z19) F(B,Z19))  /+Q(H(A,B) A QUOTI
ENT B)/  /+D(M7 A B)/  /+P(A)/

79 FROM 62,5   +D(H(A,B) A F(B,Z19))  /+Q(H(A,B) A F(QUOTIENT,Z19) F(B,Z
19))/  /+Q(H(A,B) A QUOTIENT B)/  /+D(M7 A B)/  /+P(A)/

474 FROM 79,16   +D(H(A,B) A Z14)  -E(H(A,B) Z14 F(B,Z19))  /+D(H(A,B) A
 F(B,Z19))/  /+Q(H(A,B) A F(QUOTIENT,Z19) F(B,Z19))/  /+Q(H(A,B) A QUOTI
ENT B)/  /+D(M7 A B)/  /+P(A)/

475 FROM 474,43   -E(H(A,B) S(C) F(B,Z19))  /+D(H(A,B) A F(B,Z19))/  /+Q
(H(A,B) A F(QUOTIENT,Z19) F(B,Z19))/  /+Q(H(A,B) A QUOTIENT B)/  /+D(M7
A B)/  /+P(A)/

476 FROM 475,447 **CLAUSE*VIDE**
```

```
EXEMPLE LANG3

TEMPS=2703,80 MILLISECONDES
                    PREUVE*DE*NIVEAU*5
MAXEN*MAXBLITS*MAXALITS
3       3         2
CHAINES*ESSAYEES*ACCEPTEES*PONT*INUTILES*UNITAIRES
          70        43              29        18

8 AX7   -LA(X8 Y8)  +ART(X8 Y8)

4 AX4   +LA(4 5)

47 FROM 4,8   +ART(4 5)  /+LA(4 5)/

29 AX16   -NOM(Y28 Z28)  -ART(X28 Y28)  +GN(Y28 Z28)

12 AX9   -PORTE(Y12 Y12)  +NOM(X12 Y12)

5 AX5   +PORTE(5 6)

43 FROM 5,12   +NOM(5 6)  /+PORTE(5 6)/

59 FROM 49,29   -ART(X28 5)  +GN(Y28 6)  /+NOM(5 6)/  /+PORTE(
5 6)/

60 FROM 59,47   +GN(4 6)  /+NOM(5 6)/  /+PORTE(5 6)/

33 AX18   -VB(X33 Y33)  -GN(Y33 Z33)  +GV(Y33 Z33)

18 AX12   -FERME(X18 Y18)  +VB(X18 Y18)

3 AX3   +FERME(3 4)

46 FROM 3,18   +VB(3 4)  /+FERME(3 4)/

63 FROM 46,33   -GN(4 Z33)  +GV(3 Z33)  /+VB(3 4)/  /+FERME(3
4)/

64 FROM 63,60   +GV(3 6)  /+VB(3 4)/  /+FERME(3 4)/

14 AX10   -PILOTE(X14 Y14)  +NOM(X14 Y14)

2 AX2   +PILOTE(2 3)

44 FROM 2,14   +NOM(2 3)  /+PILOTE(2 3)/

6 AX6   -LE(Y6 Y6)  +ART(Y6 Y6)

1 AX1   +LE(1 2)
```

```
43 FROM 1,6    +ART(1 2)  /+LE(1 2)/

30 AX16   +GN(X28 Z28)  -ART(X28 Y28)  -NOM(Y28 Z28)

41 AX20   +PH(Y39 Z39)  -GN(X39 Y39)  -GV(Y39 Z39)

42 TH   -PH(1 6)

55 FROM 42,41   -GN(1 Y39)  -GV(Y39 6)  /-PH(1 6)/

82 FROM 55,30   -ART(1 Y28)  -NOM(Y28 Y39)  /-GN(1 Y39)/  -GV(Y39
6)  /-PH(1 6)/

83 FROM 82,43   -NOM(2 Y39)  /-GN(1 Y39)/  -GV(Y39 6)  /-PH(1
6)/

84 FROM 83,44   -GV(3 6)  /-PH(1 6)/

85 FROM 84,64 **CLAUSE*VIDE**
```

BIBLIOGRAPHIE

ACKERMANN, W. (1954)

"Solvable cases of the decision problem".
North-Holland publishing Co., Amsterdam, 1954.

BOYER, R.S., MOORE, J.S. (1973)

"Proving theorems about LISP functions".
I.J.C.A.I., Stanford, pp. 486-493 (Août 1973).

CHANG, C.L. (1970)

"The unit proof and the input proof in theorem-proving".
J.A.C.M., Vol. 17, N° 4, pp. 698-707 (Oct. 1970).

CHANG, C.L. (1972)

"Theorem-proving with variable-constrained resolution".
Information sciences 4, pp. 217-231 (1972).

CHANG, C.L., LEE, R.C.T. (1973)

"Symbolic logic and mechanical theorem-proving".
Academic Press. Computer science and applied mathematics series.
New-York, 1973.

CHURCH, A. (1936)

"An insolvable problem of number theory".
Amer. J. Math., Vol. 58, pp. 345-363 (1936).

COLMERAUER, A. (1970)

"Les systèmes-Q, ou un formalisme pour analyser et synthetiser des phrases sur ordinateur". Publication interne, n° 43, Département d'informatique - Université de Montréal (Sept. 1970).

COLMERAUER, A. et al. (1972)

"Un système de communication homme-machine en français".
Rapport CRI - Groupe de recherche en Intelligence - Université d'Aix-Marseille, Lumigny (1972).

COURTIN, J. (1973)
"Un analyseur syntaxique interactif pour la communication homme-machine" Compte-rendu d'une conférence internationale sur le traitement automatique des langues. Pise (Août, 1973).

DARLINGTON, J.L. (1969)
"Theorem-proving and information retrieval".
Machine Intelligence 4. Edinburgh University Press (1969).

DARLINGTON, J.L. (1971)
"A partial mechanization of second order logic".
Machine Intelligence 6 - B. Meltzer & D. Michie, eds, pp. 91-100, American Elsevier, New-York (1971).

DAVIS, M., PUTNAM, H. (1960)
"A computing procedure for quantification theory".
J.A.C.M. Vol. 7, N° 3, pp. 201-215 (1960).

DESCARTES, R. (1637)
"Discours de la méthode".
Gallimard - La Pléïade - Paris (1953).

DUPRAZ, M. (1966)
"Utilisation de l algèbre de boole en logique mathématique".
Thèse de doctorat de spécialité, 3° cycle. Grenoble (1966).

FEIGENBAUM & FELDMANN, eds (1963)
"Computers and thought".
Mc Graw Hill - New-York (1963).

FLEISIG, S., LOVELAND, D.W., SMILEY III, A.K., YARMUSH, D.L. (1974)
"An implementation of the model-elimination proof procedure".
J.A.C.M. Vol. 21, N° 1, pp. 124-140 (Janv. 1974).

GELERNTER, H. (1959)
"Realization of a geometry theorem-proving machine", dans FEIGENBAUM & FELDMANN (1963), pp. 153-163.

GILMORE, P.C. (1960)
"A proof method for quantification theory : its justification and realization". I.B.M. J. of Res. and Dev., pp. 28-35 (Janv. 1960).

GÖDEL, K. (1931)
" On formally undecidable proposition of Principia Mathematica and related systems", dans VAN HEIJENOORT (1967).

GREEN, C.C. (1969)
"The application of theorem-proving to question-answering systems". Ph. D. Thesis - Stanford University (Juin, 1969).

GUARD, J. et al. (1969)
"Semi-automated mathematics".
J.A.C.M. Vol. 16, N° 1, pp. 49-62 (Janv. 1969).

GUIZOL, J. (1972)
Rapport de DEA. Groupe de recherche en Intelligence artificielle, Université d'Aix-Marseille Lumigny (1972).

HART, P.E., NILSSON, N.J., RAPHAEL, B. (1968)
"A formal basis for the heuristic determination of minimum cost paths". IEEE trans. on sys. Sc. and Cyb., (July, 1968).

HENKIN, L. (1950)
"Completeness in the theory of types".
J. of Symb. logic, Vol. 15, N° 2, pp. 81-91 (Juin, 1950).

HERBRAND, J. (1930)
"Recherches sur la théorie de la démonstration". Travaux de la société des sciences et lettres de Varsovie. Classe 3
Sciences physiques et mathématiques, N° 33 (1930).

HEWITT, C. (1972)
"Description and theoretical analysis of PLANNER : a language for proving theorems and manipulating models in a robot".
M.I.T., Artificial Intelligence laboratory, report AI TR-28 (Av. 1972).

HILBERT, D., ACKERMANN, W. (1928)
"Grundzüge der theoretischen logic". Traduction anglaise :
"Principles of Mathematical logic", Chelsea Pub, Co., New-York (1950).

HUET, G.P. (1973)
"Resolution in type theory".
Int. Joint Conf. on Art. Int., Stanford (Août, 1973).

KLEENE, S.C. (1971)
"Logique mathématique".
Collection U, Armand Colin, Paris (1971)

KNUTH, D.E. (1968)
"The art of computer programming".
Addison-Wesley, Reading, Pa. (1968).

KOWALSKI, R. (1969)
"Search strategies for theorem-proving".
Machine Intelligence 5, pp. 181-200. B. Meltzer & D. Michie, Eds,
Edinburgh University press (1970).

KOWALSKI, R. (1970)
"Studies in the completeness and efficiency of theorem-proving by resolution". Ph. D. Thesis, Edinburgh University - Scotland (Av. 1970).

KOWALSKI, R. (1972)
"And or graphs, theorem-proving graphs and bi-directional search".
Machine Intelligence 7, pp. 167-194. B. Meltzer & D. Michie, eds,
Edinburgh University press (1972).

KOWALSKI, R. (1973)
"An improved theorem-proving system for first-order logic".
Memo. N° 65. Department of Computational logic, School of A.I.,
Edinburgh University (1973).

KOWALSKI, R. (1973 b)
"Predicate logic as programming language".
Memo. N° 70 - Depart. of Comp. logic, School of A.I., Edinburgh
University (Nov. 1973).

KOWALSKI, R., KUEHNER, D. (1970)
"Linear resolution with selection function".
Artificial Intelligence N° 2, pp. 227-260 (1971), (publié en 1970
dans un rapport de l'Université d'Edinburgh).

LABORDE, J.M. (1974)
"Généralisation de l'algorithme d'exclusion".
Zentralblatt für mathematik und Grenzgebiet. Académie des Sciences
de R.D.A. (Janv. 1974).

LEIBNIZ, G.W. (1703)

"Nouveaux essais sur l'entendement humain".
Garnier-Flammarion, Paris (1966).

LOVELAND, D.W. (1969)

"A simplified format for the model elimination theorem-proving procedure". J.A.C.M. Vol. 16, N° 3, pp. 349-363 (July 1969).

LOVELAND, D.W. (1970 a)

"A linear format for resolution".
Proc. of IRIA Symp. on Automatic demonstration, Versailles, France, 1968. Lecture Notes in Mathematics, N° 125 - Springer - Verlag, New-york, pp. 147-162 (1970).

LOVELAND, D.W. (1970 b)

"Some linear Herbrand proof procedures : an analysis".
Depart. of Comp. Science - Carnegie Mellon University (Dec. 1970).
Paru sous le titre : "A unifying view of some linear Hebrand procedures" dans J.A.C.M., J.A.C.M. Vol. 19, N° 2, pp. 366-384 (AV. 1972).

LOVELAND, D.W., STICKEL, M.E. (1973)

"A hole in goal trees : some guidance from resolution theory".
Proceedings of the I.J.C.A.I., pp. 153-161, Stanford (Août 1973).

LUCKHAM, D. (1968)

"Refinement theorems in resolution theory" dans Proc. of IRIA Symp. on Automatic demonstration. Versailles, France (1968). Lecture Notes on Mathematics N° 125 - Springer Verlag,New-York (1970).

LUCKHAM, D. (1968 b)

"Some tree paring strategies for theorem-proving".
Machine Intelligence 3. E. Dale & B. Meltzer, eds, Oliver and Boyd, Edinburgh, pp. 95-112 (1968).

LUX, A. (1974)

"Utilisation de LISP 1.5 sous CP-CMS". Rapport interne du Laboratoire de Mathématiques Appliquées-Informatique de l'U.S.M.G. Grenoble (1974).

MAC CARTHY, J. (1961)

"LISP 1.5 programmer's manual".
M.I.T. Press, Cambridge, Mass. (1961).

MANNA, Z. (1969)
"The correctness of programs".
Journal of computer and system sciences 3 (1969).

MANNA, Z., WALDINGER, R. (1971)
"Towards automatic program synthesis".
C.A.C.M. Vol. 14, N° 3, pp. 151-165 (Mar. 1971).

MELTZER, B. (1966)
"Theorem proving for computers : some results on resolution and renaming". Computer Journal, Vol. 8, pp. 341-343 (1966).

MELTZER, R. (1970)
"Prolegomena to a theory of efficiency of proof procedures".
Advanced study institute on A.I. and heuristic programming.
Menaggio - Italy. pp. 15-33 (Août 1970).

MENDELSON, E. (1964)
"Introduction to mathematical logic".
D. Van Nostrand Company Inc., Princeton - New-Jersey (1964).

NEWELL, A., SHAW, J., SIMON, H. (1957)
"Empirical explorations of the logic theory machine"
dans FEIGENBAUM & FELDMANN (1963), pp. 109-133.

NGOA, M.H. (1967)
"Algorithmes de démonstration automatique pour le calcul des prédicats classiques". Thèse de doctorat de spécialité - Lille (1967).

NILSSON, N.J. (1971)
"Problem-solving methods in artificial intelligence".
Mc Graw Hill Book Company - New-York (1971).

PICHAT, E. (1970)
"Contribution à l'algorithmique non numérique dans les ensembles ordonnés". Thèse de doctorat ès-sciences. Université de Grenoble (Oct. 1970).

PIROTTE, A. (1972)
"Automatic theorem-proving based on resolution". Report R 169, M.B.L.E., Bruxelles (July 1971). Revised in Dec. 1972).

PITRAT, J. (1966)
"Réalisation de programmes de démonstration de théorèmes utilisant des méthodes heuristiques". Thèse de doctorat ès-sciences - Paris (1966).

PITRAT, J. (1970)
"Un programme de démonstration de théorèmes".
DUNOD - Paris (1970).

POHL, I. (1969)
"Bi-directional and heuristic search in path problems".
Ph. D. Thesis - Stanford University (1969).

PRAWITZ, D., PRAWITZ, H., VOGHERA, N. (1960)
"A mechanical proof procedure and its realization in an electronic computer".
J.A.C.M. Vol. 7, N° 1-2, pp. 102-128 (Janv. - Av. 1960).

REBOH, R., RAPHAEL, B., YATES, R.A., KLING, R.E., VELARDE, C. (1972)
"Study of automatic theorem proving programs".
A.I. center, T.N. 75, S.R.I., Stanford - U.S.A. (Nov. 1972).

ROBINSON, J. A. (1965)
"A machine oriented logic based on the resolution principle".
J.A.C.M., Vol. 12, N° 1, pp. 23-41 (Janv. 1965).

ROUSSEL, P. (1972)
"Définition et traitement de l'égalité formelle en démonstration automatique". Thèse de doctorat de spécialité - Université d'Aix-Marseille Luminy (Mars 1972).

SIRET, Y. (1967)
"Utilisation de LISP 1.5 sur IBM 360". Rapport IMAG - Grenoble (1967).

STICKEL, M.E. (1974)
"The programmable strategy tneorem-prover : an implementation of the linear MESON procedure". Rep. of Dept of Comp. Sc. Carnegie-Mellon University - Pittsburgh, Pa (June 1974).

TURING, A.M. (1936)
"On computable numbers with an application to the entscheidungs-problem".
Proceedings of the London Mathematical Society, 2, 42, pp. 230-265 (1936-1937).

VAN HEIJENOORT, J. Editeur (1967)
"From Frege to Gödel, a source book in mathematical logic, 1879-1931".
Cambridge, Mass., Harvard University press (1967).

WANG, H. (1960)
"Proving theorems by pattern recognition".
PART. I : C.A.C.M., Vol. 3, N° 4, pp. 220-234 (1960),
PART II : Bell System Tech. Jour., Vol. 11, N° 1, pp. 1-41 (1961).

WOS, L., CARSON, D., ROBINSON, G. (1964)
"The unit preference strategy in theorem-proving".
Proceedings of AFIPS, F.J.C.C., Vol. 26, pp. 616-621 (1964).

WOS, L., ROBINSON, G., CARSON, D. (1965)
"Efficiency and completeness of the set of support strategy in theorem-proving".
J.A.C.M., Vol. 12, N° 4, pp. 536-541 (Oct. 1965).

WOS, L., ROBINSON, G., CARSON, D., SHALLA, L. (1967)
"The concept of demodulation in theorem-proving".
J.A.C.M., Vol. 14, N° 4, pp. 698-709 (Oct. 1967).