USTL
FLANDRES ARTOIS

LIFL
U.A 369 du C.N.R.S

LABORATOIRE D'INFORMATIQUE FONDAMENTALE DE LILLE

N° d'ordre : 673

THÈSE

Nouveau Régime

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour obtenir le titre de

DOCTEUR en INFORMATIQUE

par

Philippe MATHIEU

L'UTILISATION DE LA LOGIQUE TRIVALUEE DANS LES SYSTEMES EXPERTS

Thèse soutenue le 14 Janvier 1991 devant la commission d'Examen

Membres du jury

Président	M. MERIAUX
Rapporteurs	L. FARRINAS Del CERRO
	P. SIEGEL
Directeur de thèse	J.P. DELAHAYE
Examinateurs	G. COMYN
	P.Y. GLOESS

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE
FLANDRES ARTOIS

DOYENS HONORAIRES DE L'ANCIENNE FACULTE DES SCIENCES

M.H. LEFEBVRE, M. PARREAU.

PROFESSEURS HONORAIRES DES ANCIENNES FACULTES DE DROIT ET SCIENCES ECONOMIQUES, DES SCIENCES ET DES LETTRES

MM. ARNOULT, BONTE, BROCHARD, CHAPPELON, CHAUDRON, CORDONNIER, DECUYPER, DEHEUVELS, DEHORS, DION, FAUVEL, FLEURY, GERMAIN, GLACET, GONTIER, KOURGANOFF, LAMOTTE, LASSERRE, LELONG, LHOMME, LIEBAERT, MARTINOT-LAGARDE, MAZET, MICHEL, PEREZ, ROIG, ROSEAU, ROUELLE, SCHILTZ, SAVARD, ZAMANSKI, Mes BEAUJEU, LELONG.

PROFESSEUR EMERITE

M. A. LEBRUN

ANCIENS PRESIDENTS DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

MM. M. PAREAU, J. LOMBARD, M. MIGEON, J. CORTOIS.

PRESIDENT DE L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

M. A. DUBRULLE.

PROFESSEURS - CLASSE EXCEPTIONNELLE

M. CONSTANT Eugène	Electronique
M. FOURET René	Physique du solide
M. GABILLARD Robert	Electronique
M. MONTREUIL Jean	Biochimie
M. PARREAU Michel	Analyse
M. TRIDOT Gabriel	Chimie Appliquée

PROFESSEURS - 1ère CLASSE

M. BACCHUS Pierre	Astronomie
M. BIAYS Pierre	Géographie
M. BILLARD Jean	Physique du Solide
M. BOILLY Bénoni	Biologie
M. BONNELLE Jean-Pierre	Chimie-Physique
M. BOSCQ Denis	Probabilitéss
M. BOUGHON Pierre	Algèbre
M. BOURIQUET Robert	Biologie Végétale
M. BREZINSKI Claude	Analyse Numérique

M. BRIDOUX Michel	Chimie-Physique
M. CELET Paul	Géologie Générale
M. CHAMLEY Hervé	Géotechnique
M. COEURE Gérard	Analyse
M. CORDONNIER Vincent	Informatique
M. DAUCHET Max	Informatique
M. DEBOURSE Jean-Pierre	Gestion des Entreprises
M. DHAINAUT André	Biologie Animale
M. DOUKHAN Jean-Claude	Physique du Solide
M. DYMENT Arthur	Mécanique
M. ESCAIG Bertrand	Physique du Solide
M. FAURE Robert	Mécanique
M. FOCT Jacques	Métallurgie
M. FRONTIER Serge	Ecologie Numérique
M. GRANELLE Jean-Jacques	Sciences Economiques
M. GRUSON Laurent	Algèbre
M. GUILLAUME Jean	Microbiologie
M. HECTOR Joseph	Géométrie
M. LABLACHE-COMBIER Alain	Chimie Organique
M. LACOSTE Louis	Biologie Végétale
M. LAVEINE Jean-Pierre	Paléontologie
M. LEHMANN Daniel	Géométrie
Mme LENOBLE Jacqueline	Physique Atomique et Moléculaire
M. LEROY Jean-Marie	Spectrochimie
M. LHOMME Jean	Chimie Organique Biologique
M. LOMBARD Jacques	Sociologie
M. LOUCHEUX Claude	Chimie Physique
M. LUCQUIN Michel	Chimie Physique
M. MACKE Bruno	Physique Moléculaire et Rayonnements Atmosph.
M. MIGEON Michel	E.U.D.I.L.
M. PAQUET Jacques	Géologie Générale
M. PETIT Francis	Chimie Organique
M. POUZET Pierre	Modélisation - calcul Scientifique
M. PROUVOST Jean	Minéralogie
M. RACZY Ladislas	Electronique
M. SALMER Georges	Electronique
M. SCHAMPS Joel	Spectroscopie Moléculaire
M. SEGUIER Guy	Electrotechnique
M. SIMON Michel	Sociologie
Melle SPIK Geneviève	Biochimie
M. STANKIEWICZ François	Sciences Economiques
M. TILLIEU Jacques	Physique Théorique
M. TOULOTTE Jean-Marc	Automatique
M. VIDAL Pierre	Automatique
M. ZEYTOUNIAN Radyadour	Mécanique

PROFESSEURS - 2ème CLASSE

M. ALLAMANDO Etienne	Composants Electroniques
M. ANDRIES Jean-Claude	Biologie des organismes
M. ANTOINE Philippe	Analyse
M. BART André	Biologie animale
M. BASSERY Louis	Génie des Procédés et Réactions Chimiques

Mme BATTIAU Yvonne	Géographie
M. BEGUIN Paul	Mécanique
M. BELLET Jean	Physique Atomique et Moléculaire
M. BERTRAND Hugues	Sciences Economiques et Sociales
M. BERZIN Robert	Analyse
M.BKOUCHE Rudolphe	Algèbre
M. BODARD Marcel	Biologie Végétale
M. BOIS Pierre	Mécanique
M. BOISSIER Daniel	Génie Civil
M. BOIVIN Jean-Claude	Spectroscopie
M. BOUQUELET Stéphane	Biologie Appliquée aux enzymes
M. BOUQUIN Henri	Gestion
M. BRASSELET Jean-Paul	Géométrie et Topologie
M. BRUYELLE Pierre	Géographie
M. CAPURON Alfred	Biologie Animale
M. CATTEAU Jean-Pierre	Chimie Organique
M. CAYATTE Jean-Louis	Sciences Economiques
M. CHAPOTON Alain	Electronique
M. CHARET Pierre	Biochimie Structurale
M. CHIVE Maurice	Composants Electroniques Optiques
M. COMYN Gérard	Informatique Théorique
M. COQUERY Jean-Marie	Psychophysiologie
M. CORIAT Benjamin	Sciences Economiques et Sociales
Mme CORSIN Paule	Paléontologie
M. CORTOIS Jean	Physique Nucléaire et Corpusculaire
M. COUTURIER Daniel	Chimie Organique
M. CRAMPON Norbert	Tectolique Géodynamique
M. CROSNIER Yves	Electronique
M. CURGY Jean-Jacques	Biologie
Melle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	Géologie Appliquée
M. DEGAUQUE Pierre	Electronique
M. DEJAEGER Roger	Electrochimie et Cinétique
M. DELAHAYE Jean-Paul	Informatique
M. DELORME Pierre	Physioliogie Animale
M. DELORME Robert	Sciences Economiques
M. DEMUNTER Paul	Sociologie
M. DENEL Jacques	Informatique
M. DE PARIS Jean Claude	Analyse
M. DEPREZ Gilbert	Physique du Solide - Cristallographie
M. DERIEUX Jean-Claude	Microbiologie
Melle DESSAUX Odile	Spectroscopie de la réactivité Chimique
M. DEVRAINNE Pierre	Chimie Minérale
Mme DHAINAUT Nicole	Biologie Animale
M. DHAMELINCOURT Paul	Chimie Physique
M. DORMARD Serge	Sciences Economiques
M. DUBOIS Henri	Spectroscopie Hertzienne
M. DUBRULLE Alain	Spectroscopie Hertzienne
M. DUBUS Jean-Paul	Spectrométrie des Solides
M. DUPONT Christophe	Vie de la firme (I.A.E.)
Mme EVRARD Micheline	Génie des procédés et réactions chimiques
M. FAKIR Sabah	Algèbre
M. FAUQUAMBERGUE Renaud	Composants électroniques

M. FONTAINE Hubert	Dynamique des cristaux
M. FOUQUART Yves	Optique atmosphérique
M. FOURNET Bernard	Biochimie Sturcturale
M. GAMBLIN André	Géographie urbaine, industrielle et démog.
M. GLORIEUX Pierre	Physique moléculaire et rayonnements Atmos.
M. GOBLOT Rémi	Algèbre
M. GOSSELIN Gabriel	Sociologie
M. GOUDMAND Pierre	Chimie Physique
M. GOURIEROUX Christian	Probabilités et Statistiques
M. GREGORY Pierre	I.A.E.
M. GREMY Jean-Paul	Sociologie
M. GREVET Patrice	Sciences Economiques
M. GRIMBLOT Jean	Chimie Organique
M. GUILBAULT Pierre	Physiologie animale
M. HENRY Jean-Pierre	Génie Mécanique
M. HERMAN Maurice	Physique spatiale
M. HOUDART René	Physique atomique
M. JACOB Gérard	Informatique
M. JACOB Pierre	Probabilités et Statistiques
M. Jean Raymond	Biologie des populations végétales
M. JOFFRE Patrick	Vie de la firme (I.A.E.)
M. JOURNEL Gérard	Spectroscopie hertzienne
M. KREMBEL Jean	Biochimie
M. LANGRAND Claude	Probabilités et statistiques
M. LATTEUX Michel	Informatique
Mme LECLERCQ Ginette	Catalyse
M. LEFEBVRE Jacques	Physique
M. LEFEBVRE Christian	Pétrologie
Melle LEGRAND Denise	Algèbre
Melle LEGRAND Solange	Algèbre
M. LEGRAND Pierre	Chimie
Mme LEHMANN Josiane	Analyse
M. LEMAIRE Jean	Spectroscopie hertzienne
M. LE MAROIS Henri	Vie de la firme (I.A.E.)
M. LEROY Yves	Composants électroniques
M. LESENNE Jacques	Systèmes électroniques
M. LHENAFF René	Géographie
M. LOCQUENEUX Robert	Physique théorique
M. LOSFELD Joseph	Informatique
M. LOUAGE Francis	Electronique
M. MAHIEU Jean-Marie	Optique-Physique atomique
M. MAIZIERES Christian	Automatique
M. MAURISSON Patrick	Sciences Economiques et Sociales
M. MESMACQUE Gérard	Génie Mécanique
M. MESSELYN Jean	Physique atomique et moléculaire
M. MONTEL Marc	Physique du solide
M. MORCELLET Michel	Chimie Organique
M. MORTREUX André	Chimie Organique
Mme MOUNIER Yvonne	Physiologie des structures contractiles
Mme MOUYART-TASSIN Annie Françoise	Informatique
M. NICOLE Jacques	Spectrochimie
M. NOTELET Françis	Systèmes électroniques
M. PARSY Fernand	Mécanique

M. PECQUE Marcel	Chimie organique
M. PERROT Pierre	Chimie appliquée
M. STEEN Jean-Pierre	Informatique

Remerciements.

Je tiens en premier lieu à remercier Jean-Paul Delahaye, mon directeur de thèse, pour la confiance qu'il m'a témoignée, pour ses encouragements, son entière disponibilité et pour toutes les remarques toujours très pertinentes qu'il a pu me faire durant ces années. Il m'a donné le goût de la rigueur scientifique, je lui dois beaucoup.

Je remercie également Luis Fariñas del Cerro, directeur de recherches au CNRS, ainsi que le Professeur Pierre Siegel qui m'ont fait l'honneur d'accepter d'être rapporteurs de cette thèse.

Le Professeur Gérard Comyn, actuel directeur de l'ECRC, qui m'a initié à la recherche durant mon DEA avec le dynamisme et l'enthousiasme qui le caractérisent et qui a bien voulu accepter de se déplacer de Munich pour venir assister à cette thèse.

Paul-Yves Gloess de l'UTC de Compiègne qui m'a beaucoup apporté par les discussions que nous avons eues lors de sa venue à Lille et lors de mon séjour à Compiègne.

Le Professeur Michel Meriaux actuel directeur du LIFL qui a accepté d'être président de ce jury.

Je remercie enfin P. Devienne et A. Rausy pour toutes les discussions fructueuses que nous avons eues.

Sur le plan personnel je tiens à remercier tout particulièrement Isabelle, qui m'a toujours soutenu et encouragé dans les moments difficiles inhérents à tout travail de recherche et qui a eu la lourde tâche de relire tous mes écrits, ainsi que mes parents sans qui rien n'aurait été possible.

Pour terminer je remercie mes collègues et amis, les fidèles de la salle technique (ils se reconnaîtront) pour avoir régulièrement étanché ma soif par diverses boissons pétillantes qui ont sans aucun doute stimulé mes neurones, ainsi que Mr Glanc qui a assuré la reproduction de cette thèse.

SOMMAIRE

INTRODUCTION

A. Situation du problème.

Les systèmes experts sont des outils informatiques qui ont été introduits il y a une vingtaine d'années et qui sont maintenant couramment utilisés. Un très grand nombre de ces systèmes ont été développés avec des outils qui sont basés bien souvent sur des principes heuristiques ou pragmatiques et plus rarement sur des principes logiques [HWL 83][AC 86]. Parmi les problèmes clairement liés à la logique, certains sont très élémentaires à énoncer. Il semble pourtant qu'ils aient été peu abordés, bien qu'étant très importants pour la conception, la compréhension et le développement des systèmes experts. Notre travail a consisté à étudier deux de ces problèmes. Le premier a trait aux propriétés de complétude et d'incomplétude du chaînage avant. Bien que lié aux problèmes de consistances et de contraintes d'intégrité qui ont été largement étudiés [DeK 86][Cor 86][Rou 88], il s'agit là d'un problème très différent. Le second problème a trait à la qualité des questions posées par un système en intéraction avec un utilisateur, problème qui semble n'avoir encore jamais été étudié sous la forme où nous l'énonçons ici.

Cette manière d'aborder l'intelligence artificielle par la logique est proche de celle utilisée en programmation logique, en bases de données déductives ou dans certains travaux comme [LS 88] ou [Sie 87].

B. L'achèvement des bases de connaissances.

Le chaînage avant habituellement utilisé dans les systèmes experts est incomplet car par exemple de $A \rightarrow B$ et $\neg B$ il ne déduit pas $\neg A$ qui est pourtant une conséquence logique. D'autres exemples sont plus complexes: $C \rightarrow A$, $\neg C \rightarrow B$, $A \rightarrow D$, $B \rightarrow D$. Dans ce cas D est une conséquence logique de la base qui n'est pas trouvée par le chaînage avant.

Cette incomplétude peut entraîner un fonctionnement inattendu d'une base de règles par ailleurs parfaitement correcte. Son origine a été clairement identifiée [Del 87b] [Del 87c]: la logique utilisée par le chaînage avant est une logique trivaluée particulière. Ce que calcule le chaînage avant naïf avec négation est un plus petit modèle trivalué et non pas, comme on le croit parfois, un plus petit modèle bivalué (qui n'existe pas quand on utilise des négations).

Pour cette logique trivaluée, le chaînage avant est correct et complet mais comme l'exemple précédent le montre, il s'agit d'une logique plus faible (puisqu'elle donne moins de conséquences) que celle naturellement attendue qui est la logique bivaluée usuelle.

Pour palier cet inconvénient plusieurs méthodes sont envisageables.

- n'écrire que des bases de règles pour lesquelles il y a identité entre les littéraux conséquences bivaluées et les littéraux conséquences trivaluées. C'est la méthode la plus souvent utilisée (inconsciemment) en particulier par ceux qui écrivent des bases de règles avec négations pour systèmes experts.

- tenter de modifier les algorithmes de manière à ce qu'ils calculent non plus seulement les conséquences trivaluées (insuffisantes) mais bien les conséquences bivaluées. Cette approche est bien sûr séduisante, mais puisqu'on sait que le problème de la satisfiabilité est NP-complet, le problème du calcul des conséquences bivaluées l'est lui-aussi, et donc aucun espoir d'algorithmes efficaces n'est permis dans le cas général.

- tenter d'ajouter des règles à la base de règles Br à laquelle on s'intéresse de manière à obtenir une base de règles Br' qui ait pour ensemble de littéraux conséquences trivaluées ce que Br a pour ensemble de littéraux conséquences bivaluées. Nous appellerons 'achèvement' ce type d'opération. L'intérêt d'achever ainsi une base de règles est que l'on peut continuer à utiliser le chaînage avant qui est peu coûteux en temps d'exécution, tout en obtenant toutes les conséquences bivaluées de Br ce qui permet d'avoir une pensée logique conforme à la logique standard.

L'achèvement des bases de règles nous étant apparu comme un problème fondamental, nous y consacrons la première partie de cette thèse. Nous présentons notamment des conditions suffisantes pour qu'une base soit achevée, ce qui permet de définir une méthodologie de construction de bases de connaissances, puis nous présentons une méthode d'achèvement que nous appelons "compilation logique" qui permet d'obtenir pour une base de règles donnée, les littéraux conséquence de n'importe qu'elle base de faits ajoutée par la suite et cela uniquement en utilisant un chaînage avant classique sur la base compilée. Nous transformons donc un problème qui utilise un algorithme exponentiel en temps pour chaque question posée par l'utilisateur en une phase de compilation qui est certes exponentielle mais qui permet de traiter ensuite toute modification de la base de faits avec un chaînage avant.

C. Le calcul d'une question utile dans les systèmes experts.

L'utilisation de la négation dans les systèmes experts basés sur le calcul propositionnel avec négation pose d'autres problèmes. Le chaînage mixte, qui est une association d'un chaînage avant pour effectuer les déductions et d'un algorithme de calcul de questions à poser à l'utilisateur, est maintenant couramment utilisé [IS 86][Del 87a][Guru 87]. Malheureusement bien que de nombreux systèmes experts utilisent un chaînage avant correct vis à vis de la logique trivaluée pour effectuer leurs déductions, il n'en est pas de même pour les autres algorithmes du système et notamment pour le calcul d'une question à poser à l'utilisateur, qui crée de nombreux problèmes pour des bases avec négations. Ces algorithmes, plus difficiles à réaliser que celui du chaînage avant trivalué, ne sont pas cohérents avec le reste du système. Ils sont le plus souvent construits comme une simple extension du calcul d'une question pour bases sans négations, ce qui conduit alors ces systèmes experts à poser des questions à l'utilisateur

qui n'amènent aucune information permettant de se rapprocher du but à obtenir . Ces questions sont donc inutiles voire même stupides comme le montrent les exemples suivants:

non c
si b et c alors a
si d et e alors a

D'après la base précédente, quelle question portant sur les faits de base b, c, d, e est-il utile de poser pour conclure sur a ?.
Puisque c est connu comme faux, la règle 2 ne peut pas amener a, il est donc stupide de demander la valeur de b à l'utilisateur. Par contre les questions "est-ce que d ?" ou "est-ce que e ?" pourraient rendre utilisable la règle 3 et donc amener la conclusion a. Ce sont les seules questions qu'un système un tant soit peu "intelligent" doit poser.

si non b alors a
si c alors b
si d alors non b
si e alors non c
si f alors d

Ici rien n'est initialement connu. Quelle question portant sur les faits de base e et f doit -on poser à l'utilisateur pour conclure sur a ?.
Par un raisonnement élémentaire on voit immédiatement que la question "est-ce que e ?" ne fait pas avancer la déduction de a. La seule question intéressante est ici "est-ce que f ?".

Il est important de noter que les systèmes les plus vendus actuellement, bien qu'ils soient très volumineux et très chers, sont incapables de calculer une question utile. Tous les systèmes que nous avons essayé se sont révélés incapables de résoudre le problème précédent. C'est le cas notamment des générateurs comme GURU [Guru 87] ou IS II [IS 86] qui, utilisés sur ce problème, questionnent tous les deux sur e alors que seul f est intéressant.

Nous nous plaçons dans cette partie dans le cadre des systèmes experts construits autour d'un chaînage mixte et qui utilisent des bases de règles avec négations. Nous dégageons alors les concepts d'un système de déduction ayant un algorithme de calcul d'une question adapté au fonctionnement du chaînage avant. Ce calcul se fait en deux parties. Tout d'abord l'algorithme de calcul d'une question doit s'assurer que la question calculée est 'utile', c'est-à-dire qu'elle apporte un renseignement permettant au chaînage avant de se rapprocher du but recherché. C'est ce que nous appelons le niveau logique. Puis, si plusieurs questions 'utiles' sont possibles, l'algorithme doit poser la question la plus pertinente parmi celles-ci. Ce problème étant de nature combinatoire (il faut évidemment calculer toutes les questions possibles pour pouvoir les comparer), il est alors nécessaire d'utiliser des heuristiques pour orienter plus rapidement le système dans sa recherche. C'est pourquoi nous appelons ce niveau le niveau heuristique. Sur les exemples suivants nous pouvons voir l'intérêt d'une heuristique bien conçue.

si b et x alors a
si c et x alors a

Si le but à obtenir est a il est clair que x apporte plus d'informations que b et c, et qu'à ce titre il doit être demandé en premier.

si d alors a
si b et c alors a
si x et y et z alors d
si t alors b
si u alors c

Le but a peut être obtenu par deux questions (t et u) ou trois questions (x et y et z). Dans ce cas il est préférable d'essayer de prouver la liste de deux questions.

D. Plan adopté.

Nous l'avons dit, cette thèse se divise en deux parties distinctes. La première traite de l'incomplétude du chaînage avant et résout ce problème par l'utilisation d'une méthode de compilation des bases de connaissances que nous appelons compilation logique. La seconde traite du problème du calcul de questions utiles à poser à l'utilisateur dans un système expert.

La première partie se divise en plusieurs chapitres. Après une brève introduction (chapitre I) nous présentons dans le chapitre II les définitions logiques sur lesquelles nous nous baserons par la suite et notamment les notions de base de règles et de chaînage avant par saturation. Puis, nous rappelons un certain nombre de résultats en logique bivaluée usuelle et nous présentons quelques algorithmes travaillant dans cette logique. Nous introduisons ensuite une logique trivaluée particulière permettant de rendre compte formellement de ce que calcule un chaînage avant.

Dans le chapitre III nous établissons le principe d'achèvement et nous donnons des propriétés pour s'assurer qu'une base est achevée relativement aux faits de base, ce qui permet de définir une méthodologie de construction de bases de connaissances.

Dans le chapitre IV nous présentons la compilation logique proprement dite et nous donnons un certain nombre d'algorithmes pour effectuer cette compilation. Après avoir décrit les problèmes de complexité en temps et en taille, nous détaillons d'autres applications de la compilation logique. Notamment le traitement du connecteur 'ou' en conclusion de règles et le calcul de clauses conséquences logiques d'une base de règles. Pour clore cette partie nous donnons la trace de petits problèmes caractéristiques puis nous compilons un certain nombre d'exemples classiques que l'on ne pouvait auparavant pas traiter de manière complète avec un chaînage avant.

Nous terminons enfin cette partie par une conclusion qui expose les résultats et les perspectives futures de ces travaux.

Les résultats principaux de cette partie sont le théorème 1 qui lie la saturation par chaînage avant et le plus petit modèle trivalué, les théorèmes 15 et 16 qui établissent des propriétés assurant qu'une base est achevée relativement à des bases de faits particulières et les théorèmes 22, 24, 26 et 27 qui définissent des méthodes d'achèvement complet.

La seconde partie pose le problème du calcul de questions utiles à poser à l'utilisateur dans un système expert trivalué fonctionnant en chaînage mixte. Après une brève introduction (chapitre I) nous définissons le système sur lequel nous nous basons en présentant en premier lieu le formalisme booléen pur et en insistant sur le fait qu'un usage complet de la négation peut être fait (Chapitre II).

Dans le chapitre III nous donnons deux définitions formelles de questions: les questions 'intelligentes' et les questions 'pertinentes', puis nous proposons une structuration des systèmes de calcul d'une question en deux niveaux: le niveau logique et le niveau heuristique. Nous détaillons dans une première partie les différents niveaux logiques pouvant être obtenus selon les méthodes utilisées et nous présentons deux algorithmes permettant de calculer une question pertinente et une question intelligente. Dans une seconde partie nous présentons les différents niveaux heuristiques qui nous ont amenés à définir une méthode basée sur l'attribution de coefficients de satisfaisabilité aux différentes questions pouvant être posées. Plus le niveau heuristique sera élevé plus les heuristiques utilisées seront fiables.

Après avoir défini ce qu'était une question utile dans le cas propositionnel pur, nous généralisons nos résultats dans le dernier chapitre à un langage propositionnel étendu autorisant les faits symboliques, l'utilisation de variables symboliques puis l'utilisation de méta-valeurs qui permettent de garder un contrôle sur les déductions effectuées. Nous présentons cas par cas les problèmes posés et les manières de les résoudre.

Enfin nous détaillons les grandes lignes du logiciel BIVOUAC que nous avons réalisé pour prendre en compte tous les résultats présentés dans cette thèse.

Les points importants de cette partie sont les définitions formelles de questions "pertinentes" et "intelligentes" dans le chapitre III et les algorithmes présentés en fin de ce chapitre pour calculer ces deux types de questions qui permettent d'établir les théorèmes 32 à 35.

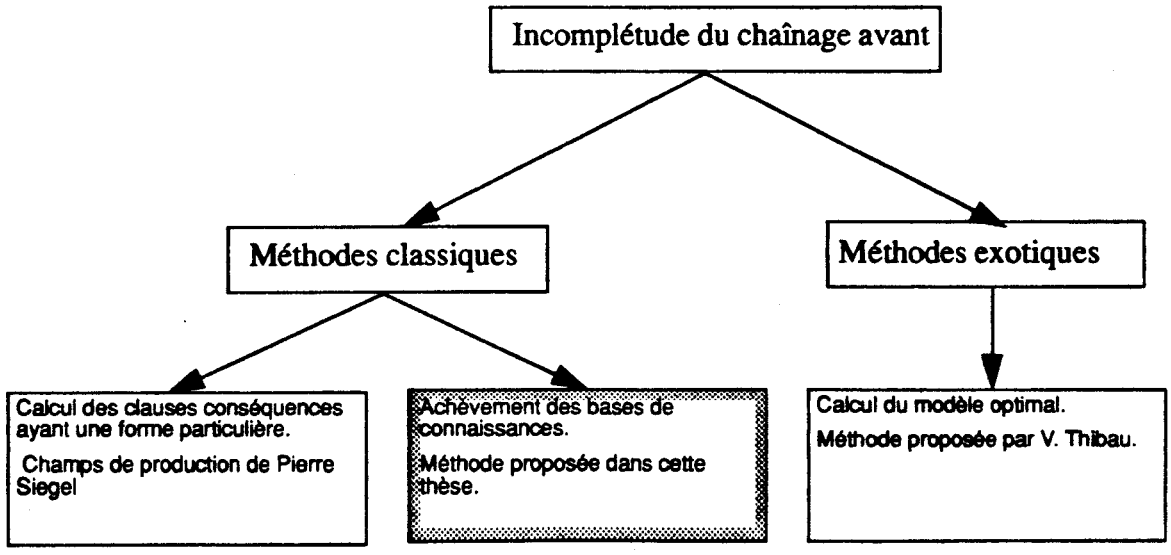
De la logique trivaluée ...

à la logique bivaluée.

I. Introduction.

La négation en programmation logique pose de nombreux problèmes. Parmi ceux-ci nous trouvons le calcul des conséquences bivaluées d'une base de règles avec négations. Le chaînage avant n'étant complet que pour des clauses de Horn il est alors nécessaire , si on fait un usage complet de la négation, d'utiliser d'autres algorithmes que le chaînage avant comme la méthode de Davis et Putnam [DP 60] ou la Sl-résolution [KK 71]. Malheureusement, non seulement ces algorithmes sont de complexité exponentielle (Ce qui est normal puisque le problème est NP-complet) mais de plus ils n'effectuent pas de déductions (point fort des algorithmes de type chaînage avant) mais uniquement des démonstrations. Un pas a donc été franchi par Pierre Siegel [Sie 87] qui, grâce à ces champs de production autorise le calcul de toutes les conséquences d'une base de connaissances qui ont une forme particulière. Cette fois-ci un nouveau type de chaînage avant a été défini, mais cette méthode est exponentielle et elle doit être appliquée à chaque modification de la base de faits. D'autres approches sur la sémantique de la négation ont été tentées et notamment l'approche récente deV. Thibau [Thi 90] qui propose de calculer le modèle optimal de la base de connaissances au lieu du plus petit modèle usuellement pris en compte. Elle propose alors d'autres types de chaînage avant pour calculer ce modèle optimal ce qui lui permet d'avoir une sémantique opérationnelle conforme à sa sémantique logique. Malheureusement, dans cette approche, le sens bivalué usuel n'est retrouvé que pour certaines bases de règles particulières.

Notre approche consiste à ajouter les règles qui manquent à la base pour pouvoir faire une déduction complète à l'aide d'un chaînage avant classique et cela quelle que soit les modifications apportées à la base de faits par la suite. Ceci définit ce que nous appelons une "compilation logique" de la base de connaissances qui est certes exponentielle mais qui n'a à être effectuée qu'une fois pour toutes. Cette approche originale nous permet de plus de résoudre le problème des systèmes à conclusion dubitative (connecteur 'ou' en conclusion) d'une manière sémantiquement propre.



II. Logique trivaluée et résolution.

1. Notations, définitions, rappels.

1.1. Atomes, littéraux, règles, clauses, variantes.

On considère donné une fois pour toutes un ensemble infini dénombrable:

$$Ato = \{ato_0 , ato_1 , ato_2 ,..., ato_n , ... \}$$

dont les éléments sont appelés **atomes.**

On appelle **littéral** toute formule de la forme ato_i (littéraux positifs) ou de la forme $\neg ato_i$ (littéraux négatifs).

L'ensemble des littéraux est noté Lit:

$$Lit = Ato \cup \neg Ato$$

On appelle **règle bivaluée** toute formule de la forme:

$$A \text{ avec } A \in Ato$$

ou de la forme:

$$A_1 \wedge A_2 \wedge ... \wedge A_n \rightarrow A \text{ avec } A, A_1, A_2,..., A_n \in Ato$$

On appelle **règle trivaluée** ou règle toute formule de la forme:

$$L \text{ avec } L \in Lit$$

ou de la forme:

$$L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow L \text{ avec } L, L_1, L_2,..., L_n \in Lit$$

La partie $L_1 \wedge L_2 \wedge ... \wedge L_n$ sera appelée **corps** de la règle et le littéral L sera appelé **tête** de la règle.

On appelle **base de règles bivaluées** tout ensemble de règles bivaluées (on aurait aussi pu appeler ce type de base des bases de Horn puisqu'elles sont composées uniquement de clauses de Horn).

On appelle base de règles trivaluées (ou simplement **base de règles**) tout ensemble de règles trivaluées.

On appelle **base de faits** toute partie finie de Lit.

Exemple:

$\{A, A \rightarrow B, A \rightarrow C, A \wedge C \rightarrow D\}$ est une base de règles bivaluées.

$\{\neg A, B, \neg C, B \wedge \neg C \rightarrow \neg E, \neg A \wedge \neg E \rightarrow D\}$ est une base de règles.

Soit Br une base de règles fixée. On définit:

Ato(Br) = l'ensemble des atomes apparaissant dans les règles de Br.

Lit(Br) = l'ensemble de littéraux apparaissant dans les règles de Br.

Her(Br) = Ato(Br) $\cup$ $\neg$Ato(Br), la base de Herbrand de Br.

Exemple:

Br = $\{\neg A, B, \neg C, B \wedge \neg C \rightarrow \neg E, \neg A \wedge \neg E \rightarrow D\}$

Ato(Br) = $\{A, B, C, D, E\}$

Lit(Br) = $\{\neg A, B, \neg C, D, \neg E\}$

Her(Br) = $\{A, B, C, D, E, \neg A, \neg B, \neg C, \neg D, \neg E\}$

On appelle **clause** toute formule de la forme:

$$ato_{i1} \vee ato_{i2} \vee ... \vee \neg ato_{in} \vee \neg ato_{j2} \vee \neg ato_{j2} \vee ... \vee \neg ato_{jm}$$

On appelle **forme clausale** de la règle $r = L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow L$, la clause $Cl(r) = \neg L_1 \vee \neg L_2 \vee ... \vee \neg L_n \vee L$

On appelle **règles variantes** de la clause $c = L_1 \vee L_2 \vee ... \vee L_n$ l'ensemble noté Var(c) des n règles suivantes :

$L_1 \wedge L_2 \wedge ... \wedge L_{n-1} \rightarrow \neg L_n$

$L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow \neg L_{n-1}$

...

$L_1 \wedge L_3 \wedge ... \wedge L_n \rightarrow \neg L_2$

$L_2 \wedge L_3 \wedge ... \wedge L_n \rightarrow \neg L_1$

On appelle variantes de la règle r et on note **Var(r)** l'ensemble Var(cl(r)).

Pour un ensemble de règles Br ou pour un ensemble de clauses C, on définit de manière évidente les ensembles Cl(Br), Var(Br), Var(C).

1.2. Bases de règles saturées.

On dit qu'une base de règles est **saturée** ssi:

pour toute règle de Br: $L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow L$
Si $L_1, L_2, ..., L_n \in$ Br alors $L \in$ Br.

Proposition.
L'ensemble des bases de règles saturées contenant une base de règles donnée (finie ou infinie) est stable par intersection. Il existe donc une plus petite base Br' saturée contenant Br, on la note **Sat(Br)**, et on l'appelle clôture par saturation de Br.

Démonstration.
Immédiate.♦

Exemple:

Br = $\{\neg A, B, \neg C, B \wedge \neg C \rightarrow \neg E, \neg A \wedge \neg E \rightarrow D\}$

Sat(Br) = Br $\cup \{\neg E, D\}$

1.3. Interprétations, modèles, consistance, conséquences.

On appelle interprétation trivaluée ou interprétation de la base de règles Br, toute partie i de Her(Br) telle que:

pour tout atome ato $\in$ Ato(Br): ato $\in$ i $\Rightarrow \neg$ato $\notin$ i et
$\neg$ato $\in$ i $\Rightarrow$ ato $\notin$ i

Les interprétations trivaluées ne sont pas fidèles au **principe du tiers exclus** car certaines propositions peuvent n'être ni vraies ni fausses.

On appelle partie positive de l'interprétation i et on note Pos(i) l'ensemble des atomes ato tels que ato $\in$ i.

On appelle partie négative de l'interprétation i et on note Neg(i) l'ensemble des atomes ato tels que $\neg$ato $\in$ i.

Bien sûr: $i = Pos(i) \cup \neg Neg(i)$

On appelle interprétation bivaluée de la base de règles Br, toute interprétation trivaluée i telle que:

pour tout ato $\in$ Ato(Br): ato $\in$ i ou $\neg$ato $\in$ i

Ce type d'interprétation est fidèle au **principe de non-contradiction** car aucune proposition ne peut faire partie de deux ensembles à la fois. Il est fidèle par contre au principe du tiers exclus car chaque proposition est soit vraie soit fausse.

L'ensemble des interprétations trivaluées de Br est noté $\mathbf{Int_T(Br)}$ et est muni naturellement de la relation d'ordre d'inclusion: $\subseteq$

L'ensemble des interprétations bivaluées de Br est noté $\mathbf{Int_B(Br)}$ et est muni de la relation d'ordre $\leq$ définie par: $i \leq j \Leftrightarrow Pos(i) \subseteq Pos(j)$

A chaque interprétation i de Br on associe une application notée encore i de l'ensemble des atomes de Br et de l'ensemble de règles qu'on peut construire avec les atomes de Br dans l'ensemble {V, F, I}, définie par:

i(ato) = V si ato $\in$ i
i(ato) = F si $\neg$ato $\in$ i
i(ato) = I sinon

i($\neg$ato) = V si i(ato) = F
i($\neg$ato) = F si i(ato) = V
i($\neg$ato) = I si i(ato) = I

$i(L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow L) = F$ si $i(L_1) = i(L_2) = ... = i(L_n) = V$ et $i(L) \neq V$
$i(L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow L) = V$ sinon

Remarque.
Cette application i, s'étend en fait en une application de l'ensemble de toutes les formules constructibles à partir des atomes de Br et des connecteurs $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\Rightarrow$, $\leftrightarrow$, par utilisation des tables de vérité trivaluées suivantes [Del 87b]:

$\neg$	V	F	I
	F	V	I

$\wedge$	V	F	I
V	V	F	I
F	F	F	F
I	I	F	I

$\vee$	V	F	I
V	V	V	V
F	V	F	I
I	V	I	I

$\rightarrow$	V	F	I
V	V	F	F
F	V	V	V
I	V	V	V

$\Rightarrow$	V	F	I
V	V	F	I
F	V	V	V
I	V	I	I

$\leftrightarrow$	V	F	I
V	V	F	F
F	F	V	I
I	F	F	V

Tous les connecteurs correspondent aux connecteurs trivalués forts de Kleene [Kle 67] excepté l'implication $\rightarrow$. L'implication forte de Kleene est définie par le connecteur '$\Rightarrow$'. Le connecteur '$\rightarrow$' n'est ni celui de Kleene [Kle 67], ni celui de Lukasiewicz [Luk 63], ni même celui présenté récemment par Przymusinski [Prz 89]. Il est défini de la manière suivante:

A$\rightarrow$B est faux si A est vrai et B ne l'est pas. A$\rightarrow$B est vrai dans tous les autres cas.

Kleene

$\Rightarrow$	V	F	I
V	V	F	I
F	V	V	V
I	V	I	I

Przymusinski

$\Rightarrow$	V	F	I
V	V	F	F
F	V	V	V
I	V	F	V

Lukasiewicz

$\Rightarrow$	V	F	I
V	V	F	I
F	V	V	V
I	V	I	V

L'ordre que nous choisissons sur les valeurs de vérité {V,F,I} est le suivant: I$\leq$F, I$\leq$V, I$\leq$I, F$\leq$F, V$\leq$V.

soit F une formule construite dans un langage logique on dira que F est **monotone** ssi pour tout couple d'interprétations i et j on a i$\subset$j $\Rightarrow$ i(F)$\leq$j(F).

Suivant cette définition notre connecteur d'implication n'est pas monotone. En effet pour deux propositions P et Q telles que P a la valeur I dans une interprétation i et la valeur V dans une interprétation j et Q la valeur I dans i et j, P$\rightarrow$Q a la valeur V dans i et F dans j alors que i$\subseteq$j.

Il est important de remarquer de plus que P$\rightarrow$Q n'est pas équivalent à $\neg$P$\vee$Q ni à $\neg$B$\rightarrow\neg$A. C'est d'ailleurs pour cette raison que l'on pourra montrer que le chaînage avant est complet pour cette logique.

On a néanmoins les équivalences suivantes:

$A \vee (B \wedge C) \equiv_T (A \vee B) \wedge (A \vee C)$

$A \wedge (B \vee C) \equiv_T (A \wedge B) \vee (A \wedge C)$

$\neg\neg A \equiv_T A$

$A \vee (B \vee C) \equiv_T (A \vee B) \vee C$

$A \wedge (B \wedge C) \equiv_T (A \wedge B) \wedge C$

$\neg(A \vee B) \equiv_T \neg A \wedge \neg B$

$\neg(A \wedge B) \equiv_T \neg A \vee \neg B$

$(A \vee B) \rightarrow C \equiv_T (A \rightarrow C) \wedge (B \rightarrow C)$

$(A \wedge B) \rightarrow C \equiv_T A \rightarrow (B \rightarrow C)$

$A \rightarrow (B \wedge C) \equiv_T (A \rightarrow B) \wedge (A \rightarrow C)$

$A \rightarrow (B \vee C) \equiv_T (A \rightarrow B) \vee (A \rightarrow C)$

L'étude détaillée de cette logique trivaluée avec en particulier une justification du choix de la table de vérité de $\rightarrow$ peut être trouvée dans [Del 87c][MD 89a][DT 90][Thi 90].

Si une formule F (construite avec $\neg$, $\wedge$, $\rightarrow$, $\vee$, $\Rightarrow$, $\leftrightarrow$) est vraie relativement à l'interprétation i (c'est-à-dire si i(F) =V) on note:

$$i \models F$$

On appelle modèle trivalué (resp. modèle bivalué) de l'ensemble de formules Y, toute interprétation trivaluée i (resp. bivaluée) telle que:

$$\text{pour tout } F \in Y : i(F) = V$$

Proposition.
Tout modèle bivalué est un modèle trivalué.

Démonstration.
Cette propriété est immédiate puisque toute interprétation bivaluée est une interprétation trivaluée et que notre logique est une extension de la logique bivaluée usuelle. Il en résulte immédiatement que si M est un modèle bivalué d'un ensemble de formules alors c'est un modèle trivalué.♦

Si F est conséquence au sens trivalué (resp. au sens bivalué) de l'ensemble de formules Y, c'est-à-dire si F est vraie dans tout modèle de Y, on note:

$$Y \models_T F \text{ (resp. } Y \models_B F)$$

On introduit les deux notations suivantes:

$$\mathbf{Cons_T(Y)} = \{Lit \in Her(Y);\ Y \models_T Lit\}$$

$$\mathbf{Cons_B(Y)} = \{Lit \in Her(Y);\ Y \models_B Lit\}$$

Deux ensembles de formules sont dit équivalents au sens bivalué (respectivement au sens trivalué) si, par définition ils ont les mêmes modèles bivalués (respectivement les mêmes modèles trivalués).

Exemple:
Une règle ou une clause est équivalente au sens bivalué à chacune de ses variantes, et donc à leur ensemble. Par contre une règle ou une clause n'est pas équivalente au sens trivalué à ses variantes.

Ces deux dernières définitions sont très importantes car par la suite nous nous intéresserons surtout aux littéraux conséquences d'un ensemble de formules et non aux formules conséquences. D'autres travaux dans un sens similaire ont déjà été réalisés mais cette fois pour calculer les clauses conséquences d'un ensemble de formules, notamment par Pierre Siegel [Sie 87] qui à l'aide de ses champs de production cherche à produire les clauses conséquences d'un ensemble de formules fidèles à une propriété donnée. On pourrait d'ailleurs définir $Cons_B(Y)$ comme étant l'ensemble calculé par la production de Pierre Siegel en prenant comme champ de production toutes les clauses de longueur 1 (constituées d'un seul littéral).

Proposition.
Pour tout ensemble de formules Y (construit avec $\neg, \wedge, \rightarrow, \vee, \Rightarrow, \leftrightarrow$)

$$Cons_T(Y) \subseteq Cons_B(Y)$$

Démonstration.
Résulte du fait que tout modèle bivalué est aussi un modèle trivalué.♦

Remarque.
Ce résultat signifie qu'en général en logique trivaluée on a moins de littéraux conséquences d'une base de règles (et plus généralement d'un ensemble de formules) qu'il n'y en a en logique bivaluée: la logique trivaluée et la négation trivaluée sont plus faibles que la logique bivaluée et la négation bivaluée.

Un ensemble de formules Y qui possède au moins un modèle bivalué (respectivement trivalué) est dit **consistant** au sens bivalué (respectivement consistant au sens trivalué).

Puisque tout modèle bivalué de Y est un modèle trivalué de Y, il en résulte que "consistant au sens bivalué" implique "consistant au sens trivalué". L'inverse est faux comme le montre l'exemple suivant:

$$Br = \{A \rightarrow B, \neg A \rightarrow B, A \rightarrow \neg B, \neg A \rightarrow \neg B\}$$

Proposition.
L'ensemble des modèles trivalués d'une base de règles Br consistante au sens trivalué est stable par intersection (finie ou infinie), et possède donc un plus petit modèle qu'on note:

$$\mathbf{ppm_T(Br)}$$

Démonstation.
Voir [Del 87c].♦

Remarque.
Cette propriété n'est pas vraie pour des ensembles de formules quelconques, et n'est pas vraie non plus pour des ensembles de règles dans lesquels on utiliserait $\Rightarrow$ (le connecteur fort de Kleene) à la place de $\rightarrow$.

Proposition.
Toute base de règles bivaluées Br est consistante au sens bivalué (car Ato(Br) est un modèle de Br) et l'ensemble des modèles bivalués de Br est stable par borne inférieure (finie ou infinie) pour la relation d'ordre $\leq$. Br possède donc un plus petit modèle bivalué que l'on note:

$$\mathbf{ppm_B(Br)}$$

Démonstration.
La consistance provient de ce que Ato(Br) est un modèle bivalué de Br, le reste est immédiat.♦

1.4. Résultat fondamental.

Les notions de saturation, plus petit modèle, ensemble de littéraux conséquences sont liées les unes aux autres par deux résultats fondamentaux, l'un pour les bases de règles bivaluées (sans négations) l'autre pour les bases de règles trivaluées (avec négation).

Théorème 1 [Del 87b] [Del 87c].

(a) Pour toute base de règles consistante au sens trivalué Br on a:

$$Cons_T(Br) = ppm_T(Br) = Sat(Br) \cap Her(Br)$$

(b) Pour toute base de règles bivaluées Br on a:

$$Cons_B(Br) = Pos(ppm_B(Br)) = Sat(Br) \cap Ato(Br)$$

On le voit, calculer l'ensemble saturé d'une base de règles avec négation Br, c'est en fait calculer un plus petit modèle de cette base (en logique trivaluée avec les tables de vérité indiquées plus haut) et c'est aussi déterminer quels sont les littéraux qui sont conséquences (toujours au sens trivalué) de Br. La logique trivaluée que nous utilisons n'est donc pas introduite pour le plaisir mais parce qu'elle s'impose dans le cadre de l'étude des bases de règles avec négation, et qu'elle donne (y compris avec des variables) des résultats aussi simples que ceux classiques obtenus pour les clauses de Horn (règles sans négations).♦

Exemple:

(a) $Br = \{\neg A, B, \neg C, B \wedge \neg C \rightarrow \neg E, \neg A \wedge \neg E \rightarrow D,$
$\neg F \rightarrow \neg D, \neg G \rightarrow H, G \rightarrow H\}$

$Cons_T(Br) = ppm_T(Br) = Sat(Br) \cap Her(Br) = \{\neg A, B, \neg C, \neg E, D\}$

$Cons_B(Br) = \{\neg A, B, \neg C, \neg E, D, F, H\}$

(b) $Br = \{A, B, C, B \wedge C \rightarrow E, A \wedge E \rightarrow D\}$

$Pos(Cons_B(Br)) = Pos(ppm_B(Br)) = Sat(Br) \cap Ato(Br) = \{A, B, C, E, D\}$

2. Algorithmes de calcul en logique bivaluée et trivaluée.

2.1. Chaînage avant

Définition.

Algorithme général de chaînage avant (Chavt).

```
Soit Br une base de règles.
Tant qu'il existe une règle de Br :L1∧L2∧...∧Ln → L, telle que :
        L1,L2,...,Ln ∈ Br et L ∉ Br.
     choisir une telle règle
     si ¬L est déjà dans Br alors
          renvoyer "Base inconsistante au sens trivalué"
          arrêt.
     sinon ajouter L à Br.
fin de tant que
renvoyer Br.
arrêt.
```

Théorème 2
(Correction et complétude du chaînage avant relativement à la logique trivaluée)

Soit Br une base de règles

- Ou bien Br est inconsistante au sens trivalué et alors Chavt renvoie 'Base inconsistante au sens trivalué'

-ou bien Br est consistante au sens trivalué et alors Chavt renvoie Sat(Br) qui vérifie:

$$Cons_T(Br) = ppm_T(Br) = Sat(Br) \cap Her(Br)$$

Démonstration.
Chaque littéral ajouté à Br par Chavt est nécessairement dans $ppm_T(Br)$ (s'il existe) donc si Chavt termine en renvoyant 'Base inconsistante au sens trivalué' c'est que Br ne possède pas de plus petit modèle trivalué, c'est-à-dire est inconsistant.

Si Chavt termine sans renvoyer 'Base inconsistante au sens trivalué', l'ensemble Br_{final} renvoyé est la plus petite base de règles saturée contenant $Br_{initial}$ (car chaque littéral ajouté par Chavt y est nécessairement) et donc Br est consistante au sens trivalué et grâce au théorème 1 on a: $Cons_T(Br) = ppm_T(Br) = Sat(Br) \cap Her(Br)$. ♦

L'algorithme Chavt peut être implémenté de différentes façons que nous ne détaillerons pas ici, l'essentiel dans toutes les implémentations étant que l'arrêt soit obtenu lorsque Br est saturé ou lorsqu'une contradiction a été obtenue.

On peut résumer les résultats donnés jusqu'ici en disant que les algorithmes de chaînage avant sont des démonstrateurs de théorèmes efficaces, corrects et complets parfaitement adaptés à la

logique trivaluée définie par les tables de vérité de $\neg$, $\wedge$, $\rightarrow$ que nous avons choisies. Peut-être d'ailleurs faudrait-il inverser la proposition et dire que: la logique trivaluée que nous avons définie est parfaitement adaptée aux algorithmes de chaînage avant qu'elle permet de considérer comme des démonstrateurs de théorèmes corrects, complets et efficaces (choses qu'ils ne sont pas pour d'autres logiques et en particulier pour la logique bivaluée usuelle).

2.2. Chaînage arrière.

Définition.

Algorithme général de chaînage arrière (Charr).

Soit Br une base de règles fixée et Q une question.

On considère la question comme liste de buts initiale. Le moteur essaie d'effacer chaque but de cette liste. Pour cela il regarde si le but le plus à gauche de la liste appartient à la base de faits. Si oui, il est effacé et l'on continue avec la liste de buts restante. Sinon, il essaie d'effacer de la même manière par appel récursif, les prémisses d'une des règles qui a le but fixé comme conséquence (règles candidates). Dans le cas où la liste de buts initiale est complètement effacée l'algorithme donne un succès, dans le cas contraire il donne un échec.

On ajoute de plus la contrainte suivante: Une règle en cours d'évaluation ne peut être réutilisée. Pour cela on vérifie avant d'utiliser une règle, qu'elle n'appartient pas à la liste de ses ancêtres (règles qui ont conduit à son appel). Cette contrainte n'est évidemment valable qu'avec des bases sans symbole de fonction).

Cette manière de gérer la liste de buts est communément appelée gestion des buts en pile car la dernière règle sélectionnée est la première à être prouvée. Si la règle candidate est choisie dans l'ordre des règles de la base on dira que l'on parcourt l'arbre associé en profondeur d'abord. Cette technique nécessite évidemment un retour aux différents points de choix en cas d'échec (Backtracking).

Attention.

Ce type de chaînage arrière n'est pas le même que celui utilisé par le langage Prolog. D'une part un littéral est considéré comme une entité à part entière (il n'y a donc pas de négation par l'échec). Un littéral n'est développé que si une règle conclut sur lui-même, même s'il est

négatif. D'autre part Prolog ne prend pas en compte la contrainte précédemment citée qui consiste à vérifier qu'une règle en cours d'utilisation ne peut être réutilisée.

Théorème 3
(Correction et complétude du chaînage arrière relativement à la logique trivaluée)

Soit Br une base de règles, soit B un but fixé

- Ou bien $B \in Cons_T(Br) = ppm_T(Br)$ *et alors Charr(B) donne un succès.*

- Ou bien $B \notin Cons_T(Br) = ppm_T(Br)$ *et alors Charr(B) donne un échec.*

Démonstration.
- Si Charr(B) donne un succès, c'est qu'il existe une suite de déductions qui permet d'obtenir B. Donc B peut être obtenu par Chavt. D'après le théorème 2 on a alors $B \in Cons_T(Br)$.

- Si $B \in Cons_T(Br)$ alors B est obtenu par Chavt. Il existe donc une suite de déductions qui permet d'obtenir B. Donc La liste de buts B peut être effacée par Chaînage arrière et Charr(B) donne un succès.♦

Le chaînage arrière tel qu'il est défini ici est donc un démonstrateur correct et complet relativement à notre logique trivaluée.

Remarque.
Il est bien évidemment possible de définir d'autres chaînages arrières. Nous ne les détaillerons pas ici, l'essentiel étant que dans chaque implémentation un succès soit donné pour un littéral si et seulement si ce littéral appartient au plus petit modèle trivalué.

2.3. Résolution.

La résolution est un concept déjà ancien qui grâce à sa puissance a été l'objet de nombreux travaux. En particulier de nombreux raffinements ont été présentés [CL 73][Lov 78][Kow 79] [Nil 80][Llo 84][Del 86]. Nous ne rappelons dans cette partie que les bases de la résolution présentées pour la première fois par Robinson [Rob 65].

Définition.
Soient deux clauses C1 et C2 de la forme $C1 = a \vee C1'$, $C2 = \neg a \vee C2'$, où C1' et C2' sont des clauses. On appelle résolvante de C1 et C2 la clause $C1' \vee C2'$. et on note:

$$C1, C2 \models_{Res} C1' \vee C2'$$

La clause n'ayant aucun littéral est appelée clause vide et est notée ❑.

On dit que l'ensemble de clauses C est clos par résolution si par définition pour tout couple de clauses C1 ∈ C, C2 ∈ C pour lequel il existe C3, tel que: C1, C2 $\models_{Res}$ C3, on a C3 ∈ C.

Théorème 4.
Un ensemble de clauses est insatisfiable si et seulement si il existe une déduction par Resolution de la clause vide. On nomme réfutation cette déduction.

Pour la démonstration voir par exemple [CL 73].♦

Théorème 5.
Soit C un ensemble de clauses, L est conséquence bivaluée de C ssi C∪{¬L} est insatisfiable.

Pour la démonstration voir par exemple [CL 73]♦

2.4. Résolution linéaire.

La résolution linéaire est un raffinement de la résolution de Robinson qui consiste à réutiliser immédiatement la dernière résolvante obtenue pour la prochaine résolution. Cette méthode qui n'est sans doute pas plus efficace que la résolution brute possède un intérêt indéniable pour nous puisqu'elle définit un formalisme plus stricte que la méthode précédente. Nous nous appuierons donc sur ce formalisme dans nos démonstrations. Il semble que cette méthode ait été introduite pour la première fois par Loveland & Luckham. On en trouvera une présentation détaillée dans [CL 73][Lov 78].

Définition.
On appelle déduction linéaire de racine c_0 à partir d'un ensemble de clauses S toute déduction $f_0,...,f_n$ telle que:

f_0 est obtenue par résolution à partir de clauses dont l'une est c_0.

f_i, i>0 est obtenue par résolution à partir de clauses dont l'une est f_{i-1}.

Théorème 6.
Soit C une clause, si C est conséquence d'un ensemble de clauses S alors il existe une déduction linéaire de racine ¬C qui contient la clause vide.

Pour la démonstration voir par exemple [CL 73][Lov 78].♦

Ce qui signifie que lorsqu'on cherche à savoir si C résulte de S, il suffit de parcourir l'arbre des déductions linéaires de racine c_0=¬C et d'y rechercher la clause vide.

La résolution linéaire est donc une méthode qui permet de savoir si un littéral est conséquence bivaluée d'une base de règles.

D'autres méthodes dérivées de la résolution linéaire sont en général plus rapides. Citons par exemple la résolution linéaire ordonnée (OL-résolution) ou encore la SL-résolution. Citons enfin la méthode de Davis et Putnam qui permet de savoir assez rapidement si un ensemble de clauses est insatisfiable [CL 73][Lov 78][Kow 79].

2.5. Résolution sans variables.

Nous présentons ici la méthode la plus simple (mais néanmoins coûteuse) utilisant la résolution pour détecter la clause vide. Elle fut présentée pour la première fois par Robinson [Rob 65] puis améliorée dans [CL 73]. Si elle n'est pas très efficace en général pour détecter la clause vide, nous verrons par la suite qu'elle possède d'autres propriétés qui nous seront fort utiles.

Proposition.
Soit C un ensemble de clauses. L'ensemble des ensembles de clauses clos par résolution sans variables est stable par intersection (finie ou infinie). Il existe donc un plus petit ensemble de clauses clos par résolution sans variables, on le note **Rsv(C)**, et on l'appelle clôture par résolution sans variables de C.

Démonstration.
Immédiate.♦

Algorithme de calcul de Rsv(Br)

```
S0 = Cl(Br)
n=1
Tq Sn-1 ≠ ∅ faire
     S = S0 ∪ ... ∪ Sn-1
     Sn = ∅
     Pour toutes les clauses C1 et C2 telles que C1∈Sn-1 et C2∈S
          avec C1 placée après C2 dans l'ordre de la liste faire
          Pour chaque résolvante C entre C1 et C2
               Si C ∉ S ∪ Sn alors
                    ajouter C à Sn
               Fsi
          Fpour
     Fpour
     n = n+1
Ftq
```

Théorème 7.
(lien entre la saturation et la résolution sans variables)

Soit Br une base de règles: $Sat(Br) \subseteq Rsv(Br)$

Démonstration.
Une conclusion de règle est ajoutée à la mémoire de travail si les prémisses de la règle sont satisfaites. En d'autres termes si à un instant donné on a X et $X \rightarrow Y$ on ajoute Y (modus ponens). Il est évident que Rsv déduit aussi le littéral Y. L'inverse n'est évidemment pas vrai. Par exemple avec $\{\neg Y, X \rightarrow Y\}$ Chavt n'ajoute rien alors que Rsv déduit $\neg X$.♦

Le chaînage avant défini précédemment est donc un cas particulier de résolution sans variables.

En fait, il ne fait que du modus ponens alors que la résolution fait aussi du modus tollens.

Théorème 8.
Rsv(Br) et Br sont équivalents au sens bivalué (c'est-à-dire ont les mêmes modèles bivalués).

Démonstration.
Par la méthode des tables de vérité on vérifie aisément que toute résolvante est conséquence de la base. Donc Rsv ne modifie pas l'ensemble des modèles bivalués.♦

Définition.
Nous appelons $\mathbf{Rsv_S(Br)}$ l'ensemble des clauses obtenu après saturation de Cl(Br) par Rsv puis simplification de Rsv(Br) en enlevant les tautologies et les clauses subsumées.

Théorème 9.
$Rsv_S(Br)$ et Br sont équivalents au sens bivalué (c'est-à-dire ont les mêmes modèles bivalués).

Démonstration.
- Suppression des tautologies.

Les tautologies sont par définition des formules vraies pour toute interprétation. Elles ne modifient pas l'ensemble des modèles bivalués. Soit Cl un ensemble de clauses, l'ensemble de clauses Cl' obtenu après suppression des tautologies possède les mêmes modèles bivalués que Cl.

- Suppression des clauses subsumées.

Une clause f subsume une clause g si l'ensemble des modèles de f est inclus dans l'ensemble des modèles de g. Autrement dit pour le cas propositionnel si g est de la forme $f \vee h$. Il est clair que f amène une contrainte plus forte, et que si f est prouvée $f \vee h$ l'est aussi. Soit Cl un ensemble de clauses, l'ensemble de clauses Cl' obtenu après suppression des clauses

subsumées possède les mêmes modèles bivalués que Cl.

La clôture par Rsv_S génère donc un ensemble de clauses ayant les mêmes modèles bivalués que l'ensemble initial. $Rsv_S(Br)$ et Br ont donc les mêmes conséquences bivaluées.♦

Remarque.
L'introduction de $Rsv_S(BR)$ n'est pas le fruit du hasard. Cette méthode est la seule méthode efficace d'utilisation de la résolution pour effectuer une saturation. En effet les clauses subsumées et les tautologies qui fournissaient auparavant des résolvantes nombreuses et inutiles peuvent maintenant être supprimées dès leur détection. Il est donc naturel de s'assurer que les résultats prouvés pour Rsv peuvent encore l'être pour Rsv_S.

Exemple:

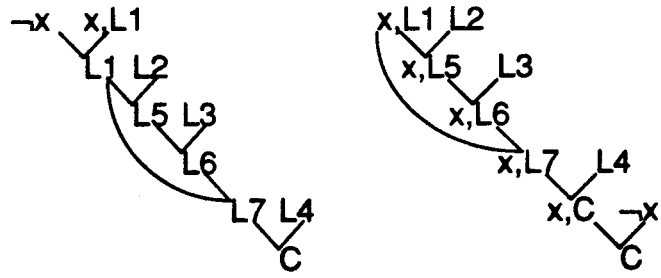
Br: $\neg A \rightarrow B$
$A \rightarrow B$

$Cl(Br)=\{A \vee B, \neg A \vee B\}$ $Rsv(Br)=\{A \vee B, \neg A \vee B, B\}$ $Rsv_S(Br)=\{B\}$

Var(Br) : $\neg A \rightarrow B$
$\neg B \rightarrow A$
$A \rightarrow B$
$\neg B \rightarrow \neg A$

Théorème 10.
Soient S un ensemble de clauses, x un littéral appartenant à S et C une clause quelconque. Si il existe une déduction linéaire de C à partir de S qui utilise x, alors l'utilisation de x peut être repoussée à la dernière résolution qui donne C comme résolvante.

Démonstration.
Après utilisation de x comme pivot, la clause résolvante est utilisée dans la résolution suivante avec un littéral y comme nouveau pivot puisque c'est une résolution linéaire. Or si x n'avait pas été utilisé, les mêmes résolutions pourraient être effectuées avec les mêmes pivots à la seule différence que le littéral x figurerait en plus dans chaque nouvelle résolvante obtenue. Donc quand on déduisait auparavant la clause C on déduira maintenant la clause $\neg x, C$ et donc l'utilisation de x peut être repoussée à la dernière résolution qui fournira C comme résolvante.♦



Théorème 11.
Soit S un ensemble de clauses et L un littéral: $S/=_B L \Leftrightarrow L \in Rsv_S(S)$

Démonstration.
Si L est une conséquence bivaluée de S on sait par le théorème 6 qu'il existe une déduction linéaire de la clause vide à partir de ¬L. Avec le théorème 10 nous pouvons repousser l'utilisation du littéral ¬L en dernier. Donc la dernière résolution portait sur un littéral L. L peut donc être obtenu par résolution sur S donc L ∈ Rsv(C). Comme L ne peut être subsumé (si l'ensemble est consistant) et ne peut être une tautologie on a alors $L \in Rsv_S(C)$.♦

La résolution sans variables est donc une méthode qui permet de calculer les conséquences bivaluées d'une base de règles. Ce résultat n'était pas évident à première vue car s'il est évident que tout littéral calculé par résolution est conséquence de l'ensemble initial, rien jusqu'à présent ne prouvait que tous les littéraux qui étaient conséquences étaient calculés.

Théorème 12.
Si Br est inconsistante alors $Rsv_S(Br)=\varnothing$

Démonstration.
Par définition la clause vide est contenue dans toute clause. Du théorème 4 nous savons que si Br est insatisfiable alors Rsv(Br) contient la clause vide. Comme cette clause subsume toutes les autres, $Rsv_S(Br)=\varnothing$.♦

2.6. Arbres sémantiques.

Les méthodes présentées précédemment étant toutes syntaxiques, nous présentons une autre méthode sémantique cette fois pour prouver la consistance d'un ensemble de clauses.

Cette méthode, communément appelée procédure de Davis&Putnam du nom des deux chercheurs l'ayant présentée pour la première fois [DP 60], consiste à vérifier que l'ensemble testé ne possède pas de modèle par construction d'un arbre binaire pour montrer qu'il est

insatisfiable. Pour cela on utilise la règle de plus petit numéro possible jusqu'à ce qu'aucune ne s'applique. Un ensemble de clauses est satisfiable si et seulement si l'un des ensembles de clauses déduit par cette procédure est satisfiable. Il est insatisfiable si tous les ensembles déduits sont insatisfiables.

Procédure de Davis & Putnam.

Règle 1: Enlever les tautologies.

Règle 2: Si l'une des clauses ne possède qu'un seul littéral L, enlever toutes les clauses contenant ce littéral L et enlever dans les autres clauses toutes les occurrences de Lc.

Règle 3: Si le littéral L apparaît dans certaines clauses et que le littéral Lc n'apparaît pas, enlever toutes les clauses contenant L.

Règle 4: Si une clause C a tous ses littéraux présents dans une clause C', enlever C'.

Règle 5: Si le littéral L ainsi que son complémentaire Lc sont présents dans l'ensemble de clauses, remplacer celui-ci par deux ensembles de clauses:
- le premier obtenu en enlevant toutes les clauses contenant L et toutes les occurrences de Lc.
- le second obtenu en enlevant toutes les clauses contenant Lc et toutes les occurrences de L.

Remarque.
On note Lc le littéral complémentaire (ou opposé) de L. La clause vide est insatisfiable tandis que l'ensemble vide (c'est-à-dire l'ensemble de clauses ne comportant aucune clause) est satisfiable.

3. Exemples d'incomplétude du chaînage avant naïf.

(1) $A \rightarrow B$ $\quad\quad$ $Cons_B=\{\neg B, \neg A\}$ $\quad$ $Cons_T=\{\neg B\}$
$\neg B$

Voici sans doute le cas le plus simple d'incomplétude du chaînage avant. Tenter de faire fonctionner une règle "en arrière" ! Dans ce cas le calcul des variantes suffit.

(2) $A \rightarrow B$ $Cons_B=\{B\}$ $Cons_T=\varnothing$
$\neg A \rightarrow B$

En logique bivaluée un atome est soit vrai soit faux. Si dans les deux cas on conclut sur le même littéral alors ce littéral est conséquence de la base. Bien sûr, ici, le calcul des variantes n'y change rien.

(3) $A \rightarrow B$ $Cons_B=\{\neg A\}$ $Cons_T=\varnothing$
$A \rightarrow \neg B$

Le même cas que le précédent mais après avoir "retourné" les implications. Par le même raisonnement on remarque que $\neg A$ est une conséquence bivaluée n'est pas ajoutée par le chaînage avant.

(4) $A,C \rightarrow B$ $Cons_B=\{C, B\}$ $Cons_T=\{C\}$
$\neg A,C \rightarrow B$
C

Dans cet exemple aucun littéral n'est initialement déductible. C'est après avoir rajouté le littéral C que l'on se retrouve dans le deuxième cas présenté.

(5) $A \rightarrow D$ $Cons_B=\{D\}$ $Cons_T=\varnothing$
$B \rightarrow D$
$C \rightarrow A$
$\neg C \rightarrow B$

Exemple beaucoup plus subtil. Ici un 'ou' est fabriqué artificiellement. Comme C est soit vrai soit faux on a donc A ou B. Dans les deux cas D doit être déduit.

(6) $A,B \rightarrow C$ $Cons_B=\{\neg C, D\}$ $Cons_T=\{\neg C\}$
$\neg A \rightarrow D$
$\neg B \rightarrow D$
$\neg C$

Un autre exemple de 'ou artificiel' cette fois-ci avec des règles fonctionnant "à l'envers". Notons que cette fois-ci 'retourner' les règles engendre un "ou" en conclusion que le chaînage avant n'est pas à même de traiter.

(7) $A \rightarrow B$ $Cons_B$=contradiction $Cons_T=\varnothing$
$A \rightarrow \neg B$
$B \rightarrow A$
$\neg B \rightarrow A$

Ce dernier cas est un exemple d'inconsistance bivaluée qui n'est pas détectée par le chaînage avant. Consistant au sens bivalué implique consistant au sens trivalué mais l'inverse est faux comme le montre cet exemple.

III. L'achèvement relatif.

1. Définitions.

Dans ce paragraphe on recherche des conditions suffisantes garantissant que le sens attendu de la base de règles (que nous identifions à l'ensemble des conséquences bivaluées de Br) est identique à celui que donne le chaînage avant.

Définition.
Soit Br une base de règles, on dit que Br est **achevée**, si, par définition

- ou bien Br est inconsistante au sens trivalué (elle l'est donc aussi au sens bivalué),

- ou bien Br est consistante au sens bivalué et:

$$Cons_T(Br) = Cons_B(Br)$$

Remarque.
Une base de règles achevée est donc une base de règles avec laquelle on peut facilement travailler, car:

- ou bien elle est inconsistante au sens bivalué et alors elle l'est aussi au sens trivalué, et donc on peut le savoir sans peine par exemple grâce au chaînage avant,

- ou bien elle est consistante au sens bivalué et alors elle l'est aussi au sens trivalué et on peut connaître tous les littéraux qui sont conséquences bivaluées de Br, grâce au chaînage avant.

Définition.
Soit Br une base de règles, on dit que Br est **complètement achevée**, si, par définition pour toute base de faits $Bf \subseteq Lit$, la base $Br \cup Bf$ est achevée.

Remarque.
Avoir une base complètement achevée est très intéressant. Cela nous permet de calculer les conséquences bivaluées d'une base de règles avec un chaînage avant et cela quelle que soit la base de faits ajoutée. Cette application est typique des systèmes experts où la base de règles est figée une fois pour toutes et où la base de faits est modifiée à chaque utilisation.

2. Résultats

Proposition.
(a) Toute base complètement achevée est achevée.

(b) Il existe des bases de règles achevées qui ne sont pas complètement achevées.

(c) Il existe des bases complètement achevées

Démonstration.
(a) Evident en prenant $Bf = \emptyset$

(b) $Br = \{A \rightarrow B\}$
$Br \models_B \emptyset$
$Br \models_T \emptyset$ Br est achevée.

Avec $Bf = \{\neg B\}$
$Br \cup Bf \models_B \neg A$
$Br \cup Bf \models_T \emptyset$ Br n'est pas complètement achevée.

(c) Exemple:

La base de règles $Br = \{A \rightarrow B, \neg B \rightarrow \neg A\}$ est une base de règles complètement achevée. En effet:

pour $Bf = \emptyset$, on a : $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) = \emptyset$
pour $Bf = \{A\}$, on a : $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) = \{A, B\}$
pour $Bf = \{B\}$, on a : $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) = \{B\}$
pour $Bf = \{\neg A\}$, on a : $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) = \{\neg A\}$
pour $Bf = \{\neg B\}$, on a : $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) = \{\neg A, \neg B\}$
pour $Bf = \{A, B\}$, on a : $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) = \{A, B\}$
pour $Bf = \{\neg A, \neg B\}$, on a : $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) = \{\neg A, \neg B\}$
pour $Bf = \{\neg A, B\}$, on a : $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) = \{\neg A, B\}$
pour $Bf = \{A, \neg B\}$, $Br \cup Bf$ est inconsistante au sens bi et trivalué. ♦

Proposition.
Si Br est une base de règles bivaluées alors pour toute base de faits $Bf \subseteq Ato$, la base de règles $Br \cup Bf$ est achevée (et donc, en particulier Br est achevée).

Démonstration.
D'après le théorème 1, on a en effet:

$Cons_T(Br \cup Bf) = Sat(Br \cup Bf) \cap Her(Br \cup Bf)$. Comme ici $Her(Br \cup Bf)=Ato(Br \cup Bf)$ on a alors $Cons_T(Br \cup Bf) = Sat(Br \cup Bf) \cap Ato(Br \cup Bf) = Cons_B(Br \cup Bf)$ ♦

Remarque.
La propriété énoncée dans cette proposition n'implique pas que Br soit complètement achevée, comme le montre l'exemple (b) donné dans la démonstration de la proposition précédente.

Définition.
Nous appelons base **semi-Horn** toute base de règles Br telle qu'aucun atome de Br n'apparaît dans Br à la fois sous forme positive et sous forme négative.

Proposition.
Si Br est une base semi-Horn, alors pour tout $Bf \subseteq Lit(Br)$, $Br \cup Bf$ est consistant au sens bivalué et:

$$Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf) \subseteq Lit(Br \cup Bf)$$

et donc, en particulier Br est achevée.

Démonstration
Soit F l'application qui remplace chaque littéral négatif $\neg ato$ de $Lit(Br \cup Bf)$ par ato'. Cette application s'étend sans difficulté à $Br \cup Bf$ et donne $F(Br \cup Bf) = Br' \cup Bf'$ une base de règles bivaluées vérifiant $Ato(Br' \cup Bf') = F(Lit(Br \cup Bf))$.

D'après la proposition précédente on a:

$$Cons_T(Br' \cup Bf') = Cons_B(Br' \cup Bf') \subseteq Ato(Br' \cup Bf')$$

Comme clairement:

$$F(Cons_T(Br \cup Bf)) = Cons_T(Br' \cup Bf')$$
$$F(Cons_B(Br \cup Bf)) = Cons_B(Br' \cup Bf')$$

On en déduit alors le résultat cherché. ♦

Remarque.
Les programmes normaux (sans conclusion négative), hiérarchiques (sans cycles) ou stratifiés tels qu'ils sont définis dans [Llo 84] ou [ABW 88] ne sont pas en général achevés.

Proposition.
Il existe des bases achevées et des bases complètement achevées.

Démonstration.
Pour toute base de règles consistante au sens bivalué Br, la base de règles $Cons_B(Br)$ et la base de règles $Br \cup Cons_B(Br)$ sont évidemment achevées.

Toute base de règles Br ne comportant comme règles que des littéraux ne se contredisant pas est bien sûr complètement achevée puisqu'alors:

$$Br = Cons_B(Br) = Cons_T(B). \blacklozenge$$

Définition.
On dira qu'une base Br est **achevée relativement** à une base de faits Bf ssi pour toute partie P de Bf, $Br \cup P$ est achevée.

Avoir des propriétés suffisantes pour qu'une base soit achevée relativement à une classe de bases de faits est très important. En effet, on remarque que dans les utilisations des systèmes experts seuls certains faits sont utilisés pour questionner l'utilisateur directement ou indirectement. Ce sont en général les capteurs pour les systèmes industriels et les faits dits "demandables" pour les systèmes de type diagnostic. Il est alors très important d'être sûr de connaître toutes les conséquences des faits ajoutés dans le système, même si ce n'est pas possible pour une base de faits quelconque. L'achèvement relatif se situe donc entre l'achèvement, propriété souvent insuffisante et l'achèvement complet, propriété difficile à obtenir.

Définitions.
On rappelle qu'une base de règles bivaluées (ou base de Horn) est une base de règles composée uniquement de littéraux positifs. On appelle base **anti-Horn** toute base composée uniquement de littéraux négatifs.

Propositions.
1) Toute base de règles bivaluées est achevée.

2) Toute base de règles bivaluées est achevée relativement aux atomes de la base.

3) Toute base anti-Horn est achevée.

4) Toute base anti-Horn est achevée relativement aux littéraux de la base.

Démonstrations.
1)et 2) Toute base bivaluée est une base trivaluée, nous pouvons donc utiliser les deux théorèmes suivants:

$Cons_T(Br \cup Bf) = ppm_T = Sat(Br \cup Bf) \cap Her(Br \cup Bf)$
$Cons_B(Br \cup Bf) = pos(ppm_B) = sat(Br \cup Bf) \cap Ato(Br \cup Bf)$
Or ici $Her(Br \cup Bf) = Ato(Br \cup Bf)$ donc $Cons_T(Br \cup Bf) = Cons_B(Br \cup Bf)$.

3) et 4) A une bijection près, une base anti-Horn peut être ramenée à une base de règles bivaluées (remplacer ¬Ato par Ato'). Les résultats précédents sont alors récupérables.♦

Proposition.
La réunion de deux bases achevées n'est pas en général une base achevée.

Démonstration.
Soient Br1={A→B} et Br2={A→¬B}. Ces deux bases sont achevées mais leur réunion ne l'est pas car B est une conséquence bivaluée qui n'est pas calculée par chaînage avant.♦

Définitions.
Un littéral L est dit **conclusif** s'il apparaît en conclusion d'au moins une règle de la base.

Un littéral L1 **dépend** directement d'un littéral L2 et l'on note L1 $\angle$ L2 ssi il existe une règle r telle que L1$\in$ conc(r) et L2=prem(r). On note < la clôture transitive de la relation $\angle$ et $\leq$ la clôture réflexive et transitive de la relation $\angle$.

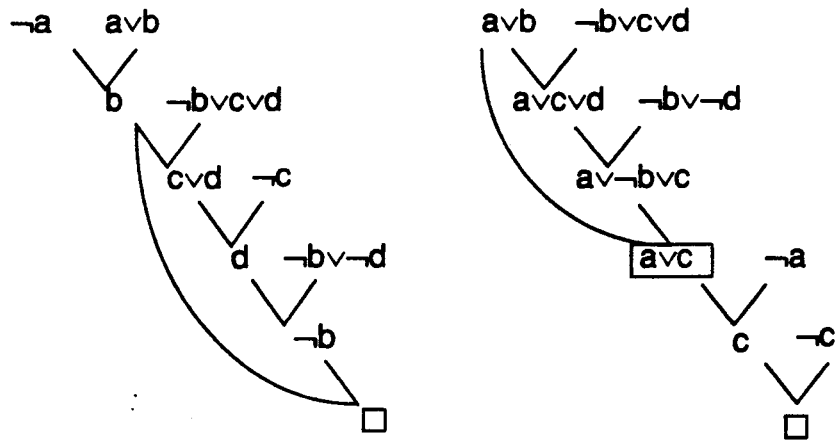
Deux littéraux L1 et L2 sont **liés** s'il existe un littéral L tel que L1$\leq$L et L2$\leq\neg$L.

Théorème 13.
Soit Br une base de règles, Bf un ensemble de littéraux et c un littéral, si $Bf \cup Br \models_B c$ *alors* $\exists$ $Bf' \{b_1,...,b_n\} \subseteq Bf$ *telle que* $\neg b_1 \vee ... \vee \neg b_n \vee c$ *appartient à Rsv(Br).*

Démonstration.
Si $Bf \cup Br \models_B c$ $Bf \cup Br \cup \{\neg c\}$ est inconsistant, donc il existe une déduction linéaire de $Bf \cup Br \cup \{\neg c\}$ qui donne la clause vide. Par une méthode analogue au théorème 10 on peut repousser l'utilisation des littéraux $b_1,...,b_n$ et $\neg c$ en fin de déduction. Donc la clause obtenue juste avant est $\neg b_1 \vee ... \vee \neg b_n \vee c$ puisqu'elle conduit à la clause vide.♦

Exemple.

{a∨b, ¬b∨c∨d, ¬b∨¬d}∪{¬a, ¬c} a pour conséquence la clause vide. Il existe donc une déduction de la clause a∨c.



Un théorème analogue au théorème 13 a été présenté par Lee [Lee 67] mais ses démonstrations étaient très complexes car il n'utilisait pas la résolution linéaire (présentée pour la première fois en 1970).

Définition.
Soit c une clause On appelle **profondeur** de c et l'on note prof(c) la fonction calculée comme suit:

Si c∈ Br alors prof(c)=0.
Si c est une résolvante entre deux clauses c1 et c2 alors
prof(c) = max(prof(c1),prof(c2)) + 1

Définition.
Nous dirons qu'une base Br vérifie la **propriété 1** ssi
∀F ∈ litt(Br), il n'existe pas de littéral L tel que F≤L et F≤¬L.

Définition.
Nous dirons qu'une base Br vérifie la **propriété 2** ssi
Pour tout couple de règles r1 et r2 de Br tel que conc(r1) = ¬conc(r2)
∃L∈ litt(Br) tel que L∈ Prem(r1) et ¬L∈ prem(r2).

Exemples:

Br1	Br2	Br3	Br4
a,b→d	a,c→d	¬a,b→c	a,x→d
a,c→d	a,¬c→d	c→a	a,y→d
			c→x
			¬c→y

Br1 vérifie la propriété 1 tandis que Br2,Br3 et Br4 ne la vérifient pas. Dans Br2 a|=d, dans Br3 b|=a car a dépend de ¬a et de lui-même, dans Br4 a|=d exemple de dépendance large).

Br5	Br6	Br7	Br8
a→c	a→c	x→a	¬c→¬d
b→c	b→¬c	a→c	a,b→d
¬a,¬b→¬c	¬b→d	b→¬c	¬a→¬e
		¬x→b	¬b→¬e
		¬b→d	

Br5 vérifie la propriété 2 tandis que Br6,Br7 et Br8 ne la vérifient pas. Dans Br6 a|=d, dans Br7 x|=d et pourtant c et ¬c ont x et ¬x comme antécédents, dans Br8 ¬c|=¬e.

Théorème 14.
Soit Br une base de règles vérifiant les propriétés 1 et 2. Toute règle obtenue par résolution est soit une tautologie, soit une clause subsumée, soit une clause contenant au moins deux littéraux conclusifs liés, soit une règle redondante c'est-à-dire pouvant être obtenue par chaînage avant.

Démonstration.
Raisonnons par récurrence sur la profondeur de résolution.

A la profondeur 1.

Soient deux règles r1 et r2 de Br respectivement de la forme prem1→conc1 et prem2→conc2 vérifiant les propriétés 1 et 2. Trois cas de résolution sont alors possibles:

1) conc1∈ prem2 (et sa réciproque conc2∈ prem1)

a) conc1=conc2, la règle r2 est donc une tautologie et la clause obtenue par résolution est donc subsumée par la règle r1.

b) conc1≠conc2, la résolution donne alors une clause avec un littéral conclusif conc2. Par la propriété 1 conc2 ne peut pas dépendre de ¬conc2. La règle obtenue par résolution a donc pour prémisses prem(r1)∪prem(r2)-{conc1}. Elle est donc simulable par chaînage avant. Elle conserve évidemment les propriétés 1 et 2 et peut donc être assimilée aux règles initiales.

2) ∃L∈ prem1 tel que ¬L∈ prem2.

Par la propriété 1 conc1 doit être différent de conc2, on obtient donc une clause avec deux littéraux conclusifs liés par un littéral L.

3) conc1=¬conc2

par la propriété 2 il existe alors un littéral L tel que L∈ prem1 et ¬L∈ prem2, la résolution donne donc une tautologie.

A la profondeur n.

Les seules clauses pouvant poser des problèmes sont les clauses avec deux littéraux conclusifs

liés. Soit une clause C3 avec deux littéraux liés x1 et y1 obtenue par résolution de deux règles R1 et R2 concluant respectivement sur x1 et y1. Montrons que quelle que soit la résolution effectuée ce nombre de littéraux liés ne peut pas diminuer.

1) résolution avec une règle R de la forme prem→conc

a) $x1 \in prem$

i) conc=x1, la règle utilisée est donc une tautologie, la résolution obtenue est subsumée par C3.

ii) conc=y1, on contredit alors la propriété 1 car x1 dépend d'un littéral l et y1 dépend d'un littéral ¬l puisque x1 et y1 sont liés. Or si conc=y1 on a y1 qui dépend de x1, donc y1 dépend de l et ¬l.

iii) conc≠y1 et conc≠x1, on supprime bien l'un des littéraux conclusifs (x1) mais on en rajoute un nouveau (conc). Or conc dépend de x1 donc conc et y1 sont liés.

b) conc=¬x1

On a donc une règle qui conclut sur x1 (R1) et une sur ¬x1 (R) donc par la proposition 2 il existe alors un littéral l tel que $l \in prem(R)$ et $\neg l \in prem(R1)$.

i) ¬l n'a pas servi comme pivot dans la résolution précédente entre R1 et R2. ¬l est donc présent dans la clause C3. Après résolution avec R on a donc une tautologie.

ii) ¬l a servi comme pivot de la résolution précédente entre R1 et R2. La clause C3 contient donc toutes les prémisses de la règle R2 qui concluait sur y1 sauf l qui a servi comme pivot. Avec la résolution avec R la clause obtenue contient à nouveau l, donc cette clause est subsumée par la règle R2 qui concluait sur y1.

2) résolution avec une autre clause avec deux littéraux conclusifs liés. appelons ces deux clauses C1 et C2. C1 contient au moins deux littéraux conclusifs liés x1 et y1. C2 contient donc au moins deux littéraux conclusifs liés ¬x1 et y2 pour pouvoir diminuer le nombre de littéraux liés.

Il y a donc deux règles qui concluent respectivement sur x1 et ¬x1 donc par la propriété 2 il existe un littéral l tel que x1<l et ¬x1<¬l

a) l n'a jamais servi comme pivot. La nouvelle clause obtenue après résolution de C1 et C2 contient donc l et ¬l, on a donc une tautologie.

b) l a servi comme pivot pour obtenir C1 (resp C2). C1 (resp C2) contient alors tous les littéraux des deux règles qui lui ont donné naissance exceptés l et ¬l puisque l était pivot. La résolution entre C1 et C2 réintroduit ¬l dans la nouvelle clause qui est donc subsumée par l'une des deux règles initiales.

c) l a servi comme pivot pour obtenir c1 ainsi que pour obtenir C2. Donc non seulement x1<l

et $\neg x1 < \neg l$ mais de plus $y1 < \neg l$ et $y2 < l$ pour qu'il y ait eu une résolution avec l comme pivot. x1 est donc supprimé par la nouvelle résolution, il reste au moins deux littéraux conclusifs y1 et y2 liés par l. ♦

Théorème 15.
Soit Br une base de règles, si Br vérifie les propriétés 1 et 2 alors Br est achevée relativement à toute base de faits ne contenant la négation d'aucun littéral conclusif.

Démonstration.
Imaginons que Br ne soit pas achevée relativement à toute base de faits ne contenant la négation d'aucun littéral conclusif. Dans ce cas $\exists$ Bf{b1,...,bn} et c un littéral tels que $Br \cup Bf \models c$ or par le théorème 13, $\neg b1 \vee ... \vee \neg bn \vee c$ appartient à Rsv(Br) et c est le seul littéral conclusif (Théorème 13).

Or nous avons montré (Théorème 14) que toute clause obtenue par résolution et n'ayant qu'un littéral conclusif peut être simulée par le chaînage avant grâce aux règles de la base. Br est donc achevée relativement à toute base de faits ne contenant la négation d'aucun littéral conclusif. ♦

Ce théorème définit une méthodologie de construction de bases de connaissances. On peut par exemple grâce aux propriétés 1 et 2 construire un vérificateur incrémental d'achèvement relatif.

Exemple.

$A, B \rightarrow \neg E$
$\neg A, D \rightarrow F$
$C, E \rightarrow H$
$\neg C, G \rightarrow \neg I$
$\neg F \rightarrow \neg I$

Cette base est achevée relativement à tous les littéraux sauf:
E, $\neg$F, $\neg$H et I

Définition.
On appelle **fermeture** d'une base de règles bivaluées sur un littéral positif L l'opération suivante.

Soient les n règles qui concluent sur L. Elles sont de la forme

$cond_1 \rightarrow L.$
$cond_2 \rightarrow L.$
...
$cond_n \rightarrow L.$

Ce qui est équivalent à $((cond_1 \vee ... \vee cond_n) \rightarrow$ alors L).

Calculer la fermeture sur L consiste à considérer l'implication comme une équivalence. C'est à dire ajouter à la base initiale la formule $((\neg cond_1 \wedge ... \wedge \neg cond_n) \rightarrow \neg L)$. La partie condition de cette règle doit alors être remise sous forme normale disjonctive, ce qui permet alors d'écrire

un certain nombre de règles conforme à notre syntaxe.

Il existe plusieurs méthodes pour calculer cette fermeture (voire par exemple [Del 87a]), notamment en ajoutant de nouveaux faits intermédiaires, mais seule la méthode de calcul présentée ci-dessous possède les propriétés que nous recherchons.

Méthode de calcul.
Comme les règles utilisées ici sont de la forme prémisses→conclusion où prémisses est une conjonction de littéraux et conclusion un littéral, l'opération de fermeture sur un littéral L revient à ajouter à la base initiale toutes les règles dont la conclusion est ¬L et dont les prémisses sont formées de l'opposé d'un littéral de chaque règle concluant sur L.

Ex:	Base initiale	règles ajoutées par fermeture
	si A et B alors C.	si ¬A et ¬D alors ¬C.
	si D et E alors C.	si ¬A et ¬E alors ¬C.
		si ¬B et ¬D alors ¬C.
		si ¬B et ¬E alors ¬C.

Proposition.
Soit Br une base de règles bivaluées, toute règle ajoutée par fermeture sur un littéral L est conforme à la propriété 2.

Démonstration.
Par construction (voir méthode de calcul) chaque règle concluant sur ¬L contient au moins l'opposé d'un littéral de chaque règle concluant sur L et réciproquement chaque règle concluant sur L contient au moins l'opposé d'un littéral de chaque règle concluant sur ¬L.♦

Proposition.
Soit Br une base de règles bivaluées, toute règle ajoutée par fermeture sur un littéral L est conforme à la propriété 1.

Démonstration.
La base initiale étant bivaluée, tout littéral de la base initiale est positif. La fermeture n'ajoute que des règles construites avec des littéraux négatifs. Il n'est donc pas possible qu'un littéral positif (donc utilisé dans la partie positive de la base) dépende d'un littéral négatif. De même il n'est pas possible qu'un littéral négatif (donc utilisé dans la partie négative de la base) dépende d'un littéral positif. Il n'existe donc pas deux littéraux F et L tels que F≤L et F≤¬L.♦

Définition.
Un atome A est dit de base s'il ne figure dans aucune conclusion de règle que ce soit de manière positive ou négative. On appellera **faits de bases**, l'ensemble des atomes de base et leurs négations.

La notion de faits de base est très courante dans les systèmes experts. Ce sont généralement sur ces atomes que des questions sont posées à l'utilisateur (faits demandables). Si la base est bien structurée, il est d'ailleurs évident que si l'atome A n'apparaît dans aucune conclusion alors $\neg$A n'apparaît pas non plus.

Théorème 16.
Toute base de règles bivaluées sur laquelle on a appliqué une ou plusieurs opérations de fermeture est achevée relativement aux faits de base.

Démonstration.
Par les deux dernières propositions nous avons montré que toute base de règles bivaluées sur laquelle on a appliqué une ou plusieurs opérations de fermeture est conforme aux propriétés 1 et 2. Par le théorème 15 nous avons donc la garantie que cette base est achevée relativement à toute base de faits ne contenant la négation d'aucun littéral conclusif. Or les faits de base ne contiennent pas la négation d'un littéral conclusif. En effet si un atome était de base avant la fermeture, il le reste et le littéral opposé ne figure par construction dans aucune conclusion de règle.

La base obtenue après fermeture est donc achevée relativement aux littéraux de base.♦

Ce théorème est fondamental car très proche de la manière d'utiliser les systèmes experts. Ces systèmes étant à trois valeurs, il est souvent intéressant pour le cogniticien de faire l'hypothèse du monde clos. Il écrit alors toutes les règles positives puis il 'ferme' les règles pour obtenir la partie négative. Comme le système pose ensuite des questions à l'utilisateur sur les faits de base, le théorème 16 nous assure de calculer avec un chaînage avant toutes les conséquences des renseignements donnés par l'utilisateur.

Prenons par exemple le cas d'un système comme IS [IS 86] qui permet d'utiliser des méta-valeurs de type connu ou inconnu. Si on veut déduire FERRARI pour une voiture rapide et LADA pour une voiture lente, on écrira les règles:

SI désirez_vous_une_voiture_rapide ALORS type_ferrari
SI non désirez_vous_une_voiture_rapide ALORS type_lada

Si l'on pense avoir indiqué toutes les règles qui peuvent conclure sur type_ferrari et type_lada on peut alors faire l'hypothèse du monde clos (sorte de complété de Clark [Llo 84] pour systèmes propositionnels) pour avoir une gestion complète de l'information négative. La fermeture ajoute alors les règles:

SI non désirez_vous_une_voiture_rapide Alors non type_ferrari
SI désirez_vous_une_voiture_rapide Alors non type_lada

On peut ensuite par exemple gérer la réponse "je ne sais pas" de l'utilisateur par une méta-valeur de type "indéterminé". Pour cela on rajoute alors la règle:

Si désirez_vous_une_voiture_rapide == indéterminé Alors type_ferrari Et type_lada.

Ce qui permet ensuite de déclencher uniquement les règles ayant type_ferrari en prémisses si l'utilisateur désire une voiture rapide, uniquement les règles ayant type_lada en prémisses si l'utilisateur désire une voiture qui n'est pas rapide et les deux sortes de règles si l'utilisateur n'a aucune idée sur la question.

IV. La Compilation logique.

1. Définitions et premiers exemples.

Lorsqu'une base de règles Br n'est pas achevée, (respectivement n'est pas complètement achevée), lui appliquer une méthode de saturation par chaînage avant ne donne pas les littéraux qui sont conséquences au sens bivalué de Br (respectivement de Br $\cup$ Bf). Il est donc naturel d'essayer de remplacer Br par une base de règles *(Br) qui soit achevée (respectivement complètement achevée) et telle que la saturation par chaînage avant appliquée à *(Br) (respectivement à *(Br) $\cup$ Bf) indique les littéraux conséquences bivaluées de Br (respectivement de Br $\cup$ Bf). D'où les définitions suivantes:

Définition.
On appelle **opération d'achèvement** une application *() qui à toute base de règles Br, consistante au sens bivalué, associe une base de règles *(Br) achevée et vérifiant que:

$$Cons_B(Br) = Cons_T(*(Br)) = Cons_B(*(Br)).$$

et qui à toute base inconsistante au sens bivalué associe une base de règles inconsistante au sens trivalué.

Définition.
On appelle **opération d'achèvement complet** une application *() qui à toute base de règles Br associe une base de règles *(Br) complètement achevée et vérifiant que:

- pour tout Bf$\subseteq$Lit(Br) tel que Br $\cup$ Bf est consistant au sens bivalué on a:

$$Cons_B(Br \cup Bf) = Cons_T(*(Br) \cup Bf) = Cons_B(*(Br) \cup Bf)$$

- et pour tout Bf$\subseteq$Lit(Br) tel que Br $\cup$ Bf est inconsistant au sens bivalué on a *(Br)$\cup$Bf inconsistant au sens trivalué.

Proposition.
Toute opération d'achèvement complet est une opération d'achèvement.

Démonstration.
Il suffit de prendre Bf = $\varnothing$ ♦

Proposition.
Il existe des opérations d'achèvement.

Démonstration.
Soit *() l'opération qui à toute base de règles Br inconsistante au sens bivalué associe la base de règles *(Br) = {A, ¬A} (qui est inconsistante au sens trivalué) et qui à toute base de règles consistante au sens bivalué associe la base de règles *(Br)=Br∪$Cons_B$(Br).

On a: $Cons_B(Br \cup Cons_B(Br)) = Cons_B(Br)$,
et: $Cons_T(Br \cup Cons_B(Br)) = Cons_B(Br)$,
donc: $Cons_B(Br) = Cons_T(*(Br)) = Cons_B(*(Br))$,

ce qui signifie que *() est une opération d'achèvement. ♦

Bien sûr une telle opération en pratique n'est pas très utile puisque qu'elle oblige à calculer $Cons_B(Br)$).

Proposition.
Il existe des opérations d'achèvement complet.

Démonstration.
Soit Br une base de règles.

Soient Bf_1, Bf_2,..., Bf_n toutes les bases de faits que l'on peut construire avec les atomes de Br (si n est le nombre d'atomes de Br, il y en a exactement 3^n).

Pour chaque Bf_i soit $Conclu_i$ l'ensemble de littéraux défini de la manière suivante:
si Br ∪ Bf_i est consistant au sens bivalué: $Conclu_i = Cons(Br \cup Bf_i)$,
si Br ∪ Bf_i est inconsistant au sens bivalué: $Conclu_i = \{A, \neg A\}$

Notons ∧ Bf_i la conjonction de tous les littéraux présents dans Bf_i. Pour chaque i on écrit toutes les règles de la forme ∧ $Bf_i \rightarrow$ lit, où lit ∈ $Conclu_i$ (pour chaque i il y a autant de règles que d'éléments de $Conclu_i$).

Soit *(Br) l'ensemble de toutes les règles ainsi obtenues quand on fait varier i. ♦

L'opération *() ainsi définie est une opération d'achèvement complet comme on le vérifie facilement. Une telle opération d'achèvement complet n'est intéressante que d'un point de vue théorique (cela nous montre qu'il en existe), mais est bien sûr trop complexe du point de vue pratique, la base de règle *(Br) ayant plus de 3^n règles ou n est le nombre d'atomes de Br. Notre but par la suite va être de trouver des opérations d'achèvement complet qui soient plus raisonnables.♦

La méthode que nous proposons est complète mais inutilisable en pratique car elle est de complexité trop élevée. Cependant elle montre bien qu'une telle opération est réalisable. On pourrait appeler cet algorithme Force Brute!.

2. Méthodes partielles.

Le problème de l'achèvement d'une base de connaissances est un problème complexe. De nombreuses méthodes peuvent paraître suffisantes à première vue mais être incomplètes. Nous présentons dans ce chapitre de telles méthodes. Pour qu'une méthode soit partielle il faut avant tout qu'elle soit correcte, c'est-à-dire qu'elle ne permette de déduire que des conséquences bivaluées avec un chaînage avant, mais aussi qu'elle permette d'en calculer plus qu'avec la base initiale. Ce qui peut se schématiser de la manière suivante:

$$\models_T \ldots \rightarrow \ldots \models_? \ldots \rightarrow \ldots \models_B$$

Prenons le cas d'une base très simple: Br={A→B}. Cette base n'est pas complètement achevée car de la base de faits Bf={¬B} on ne peut obtenir le littéral ¬A qui est pourtant conséquence bivaluée de Br∪Bf.

2.1. L'opération Var.

La première idée qui vient à l'esprit est que le calcul des variantes va résoudre ce problème. C'est effectivement suffisant dans le cas précédent, mais malheureusement incomplet dans d'autres exemples un peu plus complexes comme celui-ci:

Base	Variantes	
A, B → C	A, B → C	A partir de A et ¬D on n'obtient pas ¬B qui est pourtant conséquence bivaluée, même en utilisant les variantes (La base est pourtant bivaluée!).
B, C → D	A, ¬C → ¬B	
	B, ¬C → ¬A	
	B, C → D	
	B, ¬D → ¬C	
	C, ¬D → ¬B	

Néanmoins comme le calcul des variantes ne modifie pas l'ensemble des conséquences bivaluées et permet de calculer d'autres conséquences par chaînage avant, nous avons le résultat:

$$\models_T \ldots \rightarrow \ldots \models_{VAR} \ldots \rightarrow \ldots \models_B$$

Var est une opération d'achèvement partiel.

En examinant la structure des exemples où les variantes ne suffisent pas on pense alors faire des restrictions sur les structures acceptées considérant qu'il n'est pas très conforme à la réalité de partir de conclusions de règles (comme D) pour aboutir à des prémisses (comme B). Ce genre de bases peut donc être refusé.

On part alors du principe que la base de faits ne peut être qu'un sous-ensemble des faits de base

(faits ne figurant dans aucune conclusion de règle) et qu'une base de connaissances est toujours de structure hiérarchique (sans cycle). Cette contrainte est effectivement assez réaliste et permet d'éliminer les deux bases précédentes. Voyons alors l'exemple suivant:

$A, B \rightarrow C$
$\neg A, B \rightarrow C$
$C \rightarrow D$

Cette base est bien hiérarchique, les seuls faits de base sont A et B et pourtant à partir de la base de faits {B}, on n'obtient pas le littéral D qui est pourtant conséquence bivaluée, ni avec la base initiale ni avec la base + les variantes.

On se dit alors que les contraintes précédemment citées sont insuffisantes et qu'une base réelle est sans doute beaucoup plus structurée. On rajoute alors deux nouvelles contraintes:

. Un littéral et sa négation ne peuvent dépendre d'un même littéral.
(La base {A→B; A→¬B} serait refusée)

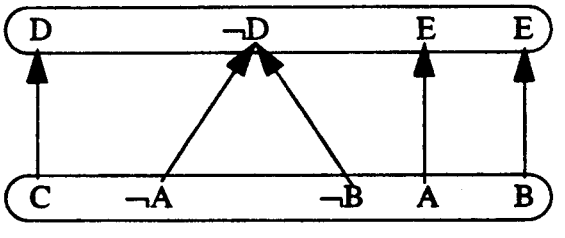
. Un littéral ne peut dépendre d'un littéral et de sa négation.
(La base {A→B; ¬A→B} serait refusée)

Cette fois-ci, on pense avoir cerné le problème, les bases tolérées étant très proches des bases réelles, et pourtant encore une fois un contre-exemple existe.

$A \rightarrow B$	De A qui est un fait de base, on ne déduit pas C qui
$\neg C \rightarrow \neg B$	est pourtant conséquence bivaluée.

Là encore les variantes résolvent cet exemple, mais ce n'est pas général comme le montre l'exemple suivant conçu sur une base hiérarchique stricte. De C on n'obtient pas E.

$C \rightarrow D$
$\neg A, \neg B \rightarrow \neg D$
$A \rightarrow E$
$B \rightarrow E$



Le seul résultat que nous ayons obtenu à propos du calcul des variantes est le suivant:

Proposition.
Pour toute base de règles Br constituée uniquement de règles n'ayant qu'une seule prémisse et telle qu'aucun littéral ne dépend d'un littéral et de sa négation et qu'aucun littéral n'est antécédent à la fois d'un atome et de sa négation, Br→Var(Br) est une opération d'achèvement complet.

Démonstration.
Toute clause obtenue par résolution sur un ensemble de clauses d'exactement deux littéraux possède deux ou un seul littéral. Si elle en possède deux , la clause obtenue correspond alors à l'expression de la transitivité de ses deux parents. Elle peut donc être obtenue par chaînage avant sur les variantes de ses parents. Si elle ne possède qu'un littéral c'est que soit ce littéral dépend d'un littéral et de sa négation, soit ce littéral est antécédent d'un littéral et de sa négation. Ce qui nie l'hypothèse. Donc toutes les clauses obtenues par résolution peuvent être simulées par chaînage avant sur les variantes.♦

Citons une autre méthode un peu plus complexe et qui à première vue peut paraître séduisante: L'achèvement par contradictions trivaluées.

2.2. L'opération AT.

Les contradictions trivaluées sont très facilement calculables par chaînage avant. On peut alors espérer trouver une propriété intéressante de l'ensemble des bases de faits qui apportent une contradiction trivaluée.

Définition.
Appelons AT, l'essai d'achèvement par contradictions trivaluées défini comme suit:

Soit Br une base de règles.

Etape 1:

pour tout sous-ensemble {X} de Her(Br) ayant un élément si Br ∪ {X} est inconsistant au sens trivalué alors ajouter ¬X à Br.

Etape 2:

pour tout sous-ensemble {X, Y} de Her(Br) ayant 2 éléments si Br ∪ {X, Y} est inconsistant au sens trivalué alors ajouter à Br toutes les variantes de ¬X∨¬Y c'est-à-dire: X → ¬Y,Y→ ¬X.

...

Etape n:

pour tout sous-ensemble {X1,..., Xn} de Her(Br) ayant n éléments si Br ∪ {X1,..., Xn} est inconsistant au sens trivalué alors ajouter à Br toutes les variantes de ¬X1∨...∨¬Xn.

Proposition.
AT(Br) ne génère pas en général une base achevée.

Démonstration.

Br: $\neg A \rightarrow B$
$A \rightarrow B$

Il est évident que seule l'étape 1 permet de rajouter des littéraux. Or $\neg B$ ne fournit aucune contradiction trivaluée donc on ne pourra jamais rajouter B qui est pourtant une conséquence bivaluée de Br.♦

On peut alors se demander si, comme précédemment, refuser qu'un littéral et sa négation dépendent d'un même littéral et refuser qu'un littéral dépende d'un littéral et de sa négation ne permettent pas d'achever une base en fonction des faits demandables.

En effet cette technique fonctionne pour un certain nombre de bases comme celle ci:

$A \rightarrow D$ E est conséquence bivaluée de C.
$B \rightarrow D$
$\neg A, \neg B \rightarrow E$
$C \rightarrow \neg D$

On obtient une contradiction avec les bases {A,C} et {B,C} ce qui nous fait rajouter les règles $C \rightarrow \neg A$ et $C \rightarrow \neg B$ ainsi que leurs variantes. Nous obtenons alors bien E à partir de C.

Malheureusement ce n'est pas général comme le montre l'exemple suivant déjà utilisé pour les variantes.

$A, B \rightarrow D$ E est conséquence bivaluée de C.
$\neg A \rightarrow E$
$\neg B \rightarrow E$
$C \rightarrow \neg D$

La seule contradiction obtenue est {A,B,C} ce qui nous fait rajouter les règles

$A, C \rightarrow \neg B$
$A, B \rightarrow \neg C$
$B, C \rightarrow \neg A.$

La base n'est pas achevée relativement au fait C puisqu'on ne peut toujours pas obtenir E par chaînage avant. Le calcul supplémentaire des variantes ne permet évidemment pas non plus d'effectuer cette déduction.

Les règles ajoutées par AT ne modifient pas l'ensemble des conséquences bivaluées de la base initiale mais permettent de déduire plus de littéraux par chaînage avant, donc on a:

$$\models_T \ldots \rightarrow \ldots \models_{AT} \ldots \rightarrow \ldots \models_B$$

AT est une opération d'achèvement partiel.

Nous n'avons pour l'instant pas trouvé de conditions intéressanteset suffisantes sur les bases, nous assurant la complétude de cette méthode.

3. Méthodes complètes.

Après avoir étudié des méthodes partielles séduisantes par leur simplicité mais insatisfaisantes, nous allons étudier maintenant des méthodes complètes, méthodes de transformation permettant d'obtenir des bases achevées dans le cas général.

3.1. L'opération AB

Définition.
Soient les formules F1,F2,...,Fn et la formule G

G est conséquence logique de F1,...,Fn si et seulement si G est vraie pour toute interprétation I dans laquelle $F1 \wedge F2 \wedge ... \wedge Fn$ est vraie.

Définition.
On note **Prem(Br)** l'ensemble des littéraux qui apparaissent dans les prémisses de Br et **Conc(Br)** l'ensemble des littéraux qui apparaissent dans les conclusions de Br.

Définition.
Soient deux clauses C1 et C2, on dit que C1 **subsume** C2 et l'on note $C1 \sqsubset C2$ si à chaque fois que C1 est vraie, C2 est vraie. (tous les littéraux de C1 sont dans C2).

Définition.
Soit Br une base de règles.

On appelle achèvement par contradictions bivaluées (AB) l'opération qui consiste à

- transformer Br en un ensemble de clauses S.
- Pour chaque partie P de $Int_T(Br)$ (donc consistante par définition), P étant de la forme $p_1,...,p_n$ telle que $Br \cup P$ est inconsistante, ajouter la clause $\neg p_1 \vee ... \vee \neg p_n$ à S
- Simplifier S en enlevant les clauses subsumées.
- Calculer les variantes des clauses de S.

On nommera AB(Br) la base ainsi obtenue.

Définition.
Soit S un ensemble de clauses et C une clause, on dira que C est **conséquence minimale** de S si C est conséquence de S et qu'il n'existe pas de clause D avec $D \sqsubset C$ telle que D est

conséquence de S.

Théorème 17.
AB(Br) est strictement équivalent à l'ensemble des clauses conséquences minimales de Br.

Démonstration.
- Soit P une partie de la forme $(p_1,...,p_n)$. Si cette partie apporte une contradiction avec Br c'est que $Br \cup (p_1 \wedge ... \wedge p_n)$ est insatisfiable. D'après le théorème 13 c'est que $\neg p_1 \vee ... \vee \neg p_n$ est une conséquence bivaluée de Br. Donc toutes les clauses ajoutées sont des conséquences bivaluées de Br.

- Toutes les conséquences sont calculées puisque chaque partie de $Int_T(Br)$ est testée et qu'une autre interprétation n'apportera jamais de contradiction.

- Toutes les conséquences qui ne sont pas minimales sont enlevées par suppression des clauses subsumées. ♦

Remarque.
D'un point de vue opérationnel, le calcul de AB(Br) se fait de manière plus simple que l'algorithme exposé.

D'une part on peut restreindre $Int_T(Br)$ à l'ensemble des interprétations trivaluées que l'on peut construire à partir de $prem(Br) \cup \neg conc(Br)$, c'est-à-dire l'ensemble opposé des littéraux de toutes les clauses. En effet si un littéral x n'apparaît dans les clauses que sous une seule forme (positive ou négative), toute partie contenant x n'apportera jamais de contradiction grâce à x. seule une partie contenant $\neg x$ pourra en apporter une.

D'autre part on teste les parties par ordre croissant sur leur nombre d'éléments (parties de 1 élément puis 2 ...etc) en ne traitant pas les parties qui contiennent une partie ayant déjà donné une contradiction. On évite ainsi de tester un grand nombre d'interprétations ainsi que la suppression des clauses subsumées.

Théorème 18.
$Cons_B(Br) = Cons_B(AB(Br))$

autrement dit, l'ensemble des déductions bivaluées de Br est égal à l'ensemble des déductions bivaluées de AB(Br).

Démonstration.
- Toute clause ajoutée est conséquence de Br (voir le théorème 17). Le fait d'ajouter une conséquence d'un ensemble de clauses à ce même ensemble ne modifie évidemment pas l'ensemble des modèles bivalués.

- La suppression des clauses subsumées ne modifie pas non plus l'ensemble des modèles car

une clause f subsume une clause g si l'ensemble des modèles de f est inclus dans l'ensemble des modèles de g. Autrement dit g est de la forme f∨h, h étant une clause, donc si f est vraie f∨h est vraie aussi. En supprimant f∨h on ne modifie pas l'ensemble des modèles bivalués.

- Le calcul des variantes ne modifie pas l'ensemble des conséquences bivaluées, car les variantes d'une clause sont des formules logiquement équivalentes à cette clause.♦

Définition.
Soit Br une base de règles.

On appelle **DChavt(Br)** l'ensemble des déductions opérationnelles directes de Br obtenues par chaînage avant et défini de la manière suivante:

DChavt(Br)={(α,x) | α est une interprétation et x un littéral tels qu'il existe une interprétation β avec β⊆α et qu'il existe une règle β→x appartenant à Br}

Définition.
Soit Br une base de règles.

On appelle **Dbiv(Br)** l'ensemble des déductions logiques bivaluées atomiques de Br défini de la manière suivante:

Dbiv(Br)={(α,x) | α est une interprétation et x un littéral tels que x est conséquence bivaluée de Br∪α}

Proposition.
Soient S un ensemble de clauses, α et β deux interprétations et x un atome. Si S∪α∪{x} et S∪β∪{¬x} donnent une inconsistance alors S∪α∪β donne une inconsistance.

Démonstration.
Evidente.♦

Théorème 19.
Soit Br une base de règles, α une interprétation trivaluée, x un littéral et Chavt l'algorithme général de chaînage avant défini précédemment.

x ∈ Chavt(AB(Br)∪α) si et seulement si (α,x) ∈ DChavt(AB(Br)).

(En d'autres termes, un seul parcours de toutes les règles est suffisant pour déduire tous les littéraux)

Démonstration.
Soient Br une base de règles et I une interprétation. Si deux passes sont nécessaires c'est qu'il existe deux interprétations α, β⊆ I et deux règles de la forme α→x et β,x→y sans qu'il n'existe

une règle $\gamma \rightarrow y$ dans AB(Br) avec $\gamma \subset I$.

Par définition de AB, $Br \cup \alpha \cup \{\neg x\}$ et $Br \cup \beta \cup \{x, \neg y\}$ donnent une inconsistance. D'après la proposition précédente l'interprétation $\alpha \cup \beta \cup \{\neg y\}$ donne aussi une inconsistance. Or cette interprétation est par hypothèse supprimée puisqu'il n'existe pas de règle de la forme $\gamma \rightarrow y$. Cette interprétation a donc été subsumée.

Or si elle est subsumée, c'est soit par une interprétation qui contient $\neg y$ et donc il existe une règle $\gamma \rightarrow y$ ce qui nie l'hypothèse, soit par une interprétation qui ne contient pas $\neg y$, donc incluse dans I et dans ce cas I apporte une contradiction en une seule passe puisqu'il existe alors une règle $\gamma \rightarrow \neg z$ avec $z \in I$. ♦

Remarque.
Le fait que le chaînage avant puisse effectuer toutes ses déductions en une passe ne veut évidemment pas dire que toutes les règles possibles soient présentes comme dans la méthode brute proposée en début de chapitre.

En effet, la base suivante est complètement achevée et pourtant les règles de la forme a,b → c ou a,b → d ne sont pas présentes.

$a \rightarrow c$.
$b \rightarrow d$.
$b \rightarrow e$.
$\neg c \rightarrow \neg a$.
$\neg d \rightarrow \neg b$.
$\neg e \rightarrow \neg b$.

Théorème 20.
Soit Br une base de règles et I une interprétation, $Br \cup I$ est inconsistant si et seulement si Chavt(AB(Br)∪I) donne une contradiction.

Lemme1. Si $Br \cup I$ est inconsistant alors Chavt(AB(Br)∪I) donne une contradiction.

Démonstration.
Soit I une interprétation de la forme $(l_1 ... l_n)$, si $Br \cup I$ est inconsistant alors par définition de AB, on a ajouté une règle $l_1 \wedge ... \wedge l_{n-1} \rightarrow \neg l_n$ donc Chavt(AB(Br)∪I) va déduire $\neg l_n$ qui apportera une contradiction.

Lemme 2. Si Chavt(AB(Br)∪I) donne une contradiction alors $Br \cup I$ est inconsistant.

Démonstration.
On a montré au théorème 18 que AB(Br) avait les mêmes conséquences bivaluées que Br. Comme le chaînage avant est un cas particulier de résolution, Si Chavt(AB(Br)∪I) donne une contradiction alors Chavt($Br \cup I$) donne une contradiction et donc $Br \cup I$ est inconsistant. ♦

Théorème 21.
Soient Br une base de règles, α une interprétation et x un littéral.
$(\alpha,x)\in Dbiv(AB(Br))$ si et seulement si $(\alpha,x)\in DChavt(AB(Br))$.

Lemme 1. Si $(\alpha,x)\in$ Dbiv(AB(Br)) alors $(\alpha,x)\in$ DChavt(AB(Br)).

Démonstration.
Si $(\alpha,x)\in$ Dbiv(AB(Br)) alors d'après la définition de Dbiv, x est conséquence bivaluée de $AB(Br)\cup\alpha$. Or d'après le théorème 5, $AB(Br)\wedge\alpha\wedge\neg x$ est insatisfiable donc l'interprétation $\alpha\cup\{\neg x\}$ donne une inconsistance qui a permis par définition de AB de rajouter la règle $\alpha\rightarrow x$. Donc $(\alpha,x)\in$ DChavt(AB(Br)).

Lemme 2. Si $(\alpha,x)\in$ DChavt(AB(Br)) alors $(\alpha,x)\in$ Dbiv(AB(Br)).

Démonstration.
Si $(\alpha,x)\in$ DChavt(AB(Br)), il existe donc une règle $\beta\rightarrow x$ avec $\beta\subseteq\alpha$ puisque tout peut être déduit en une passe (Théorème 19). Si cette règle existe c'est que l'on a obtenu une inconsistance avec $Br\wedge\beta\wedge\neg x$ par définition de AB. Or d'après le Théorème 5, x est conséquence bivaluée de $Br\cup\beta$ et donc par définition de Dbiv, $(\alpha,x)\in$ Dbiv(AB(Br)). On vérifie bien par cette définition que toute conséquence trivaluée est une conséquence bivaluée.♦

Théorème 22.
AB est une opération d'achèvement complet.

Démonstration.
Provient directement des théorèmes 18, 20 et 21.♦

3.2. L'opération AR.

Définition.
On dit qu'une déduction linéaire **utilise** un littéral x si à un moment donné il y a une résolution effectuée entre x et une autre clause.

Théorème 23.
$Rsv_S(Br)=AB(Br)$

(Autrement dit Rsv_S(Br) calcule les clauses conséquences minimales de Br).

Démonstration.
Soit C une conséquence minimale de Br de la forme $c_1\vee...\vee c_n$. Soit S l'ensemble de clauses équivalentes à Br.

D'après le théorème 5, Si C est conséquence de Br alors $S \cup \{\neg c_1,...,\neg c_n\}$ est insatisfiable, donc il existe une réfutation linéaire de $S \cup \{\neg c_1,...,\neg c_n\}$.
Or si une clause C constituée des littéraux $c_1...c_n$ est conséquence minimale d'un ensemble de clauses S alors toute réfutation linéaire de $S \cup \{C\}$ utilise les littéraux $\neg c_1...\neg c_n$. En effet imaginons que tous ne soient pas utilisés, il y a donc une clause $C' \subset C$ telle qu'il existe une réfutation linéaire de $S \cup \{C'\}$. Donc C' est conséquence de S, ce qui nie l'hypothèse puisque C est une conséquence minimale
D'après le théorème 13 on en conclut qu'il existe une déduction de la clause $c_1 \vee ... \vee c_n$.
Comme cette clause est conséquence minimale, elle n'est pas supprimée par la simplification et appartient donc à $Rsv_S(Br)$. ♦

Une démonstration différente a été effectuée dans [Pof 89] en utilisant les concepts d'interprétation interdite et de résolution duale.

Définition.
Appelons **AR(Br)** l'ensemble défini de la façon suivante: $AR(Br)=Var(Rsv_S(Br))$

Théorème 24.
AR est une opération d'achèvement complet.

Démonstration.
Résulte directement du fait que AR et AB sont équivalentes (théorème précédent), et du fait que AB est une opération d'achèvement complet (théorème 22). ♦

Théorème 25.
Si Br est inconsistante alors AR(Br)=∅

Démonstration.
Provient du théorème 12 qui montre que si Br est inconsistante alors $Rsv_S(Br)=\emptyset$. ♦

Remarques et extensions.
Il est possible d'appliquer cette méthode à la logique du premier ordre sans symbole de fonction. En effet s'il n'y a pas de symbole de fonction, la saturation par résolution est finie. Il n'y a donc aucun problème pour achever une telle base. Dans ce cas, il est alors nécessaire de modifier les tests de subsomptions et de facteurs ce qui ralentit considérablement le traitement. Si une clause C contient deux littéraux (ou plus) tels qu'ils aient un plus grand unifieur σ alors C peut être remplacée par σC (exemple $p(a) \vee p(X)$ peut être simplifiée en p(a)). Une clause C est subsumée par une clause D si et seulement si il existe une substitution telle que $\sigma D \subseteq C$ (exemple: $p(a) \vee q(a)$ est subsumée par p(X)).

Il est à remarquer que la restriction aux bases sans symbole de fonction n'est pas arbitraire. Il est très facile de trouver des exemples de bases avec symbole de fonctions qui engendrent une

saturation infinie:

A(a).	Cette base est positive et stratifiée ce qui ne l'empêche
$A(X) \rightarrow A(f(X))$	pas de générer une infinité de termes $A(f^n(a))$.

Et même sans récursivité dans les règles, des bases hiérarchiques avec symboles de fonctions engendrent des termes infinis:

$\neg R(a)$	Dans cet exemple la saturation par résolution génère des
$\neg R(X) \rightarrow \neg P(X)$	termes infinis de la forme $R(f^n(a))$ et $P(f^n(a))$.
$R(f(X)) \rightarrow P(X)$	

L'extension possible de cette méthode aux bases avec variables sans symbole de fonction est très importante et présente un très gros avantage de AR sur les méthodes que nous allons maintenant exposer.

3.3. La traversée de matrices

W. Bibel dans [Bib81] utilise une représentation par matrices initialement introduite par D. Prawitz [Pra 69] pour tester la satisfiabilité d'une formule du calcul propositionnel. Nous allons utiliser cette représentation pour calculer l'ensemble des clauses conséquences minimales d'un ensemble de clauses.

Définitions.
Une **matrice** est un ensemble d'ensembles de littéraux. Nous représentons chaque ensemble de littéraux sur ligne différente pour obtenir une forme matricielle.

Un **chemin** dans une matrice M est un ensemble de littéraux obtenu en prenant un littéral sur chaque ligne sans répétition.

Un chemin est **valide** s'il ne contient pas un littéral et sa négation.

Soit M une matrice, on note **T(M)** l'ensemble de tous les chemins valides dans M.

Transformations.
Nous définissons deux opérations de base et leurs réciproques pour transformer une formule sous forme normale conjonctive (CNF) et une formule sous forme normale disjonctive (DNF) en une matrice.

- Si F est une formule sous CNF (i.e. un ensemble de clauses) on appelle Tcm(F) la matrice obtenue en plaçant les littéraux de chaque clause horizontalement et chaque clause verticalement. Si M est une matrice, on appelle $Tcm^{-1}(M)$ l'opération inverse avec suppression des clauses subsumées.

- Si F est une formule sous DNF (i.e. un ensemble de modèles trivalués) on appelle Tdm(F) la matrice obtenue en plaçant les littéraux de chaque modèle trivalué horizontalement et chaque modèle trivalué verticalement. Si M est une matrice, on appelle $Tdm^{-1}(M)$ l'opération inverse avec suppression des modèles trivalués subsumés.

Exemple.
La formule $((a \vee b \vee c) \wedge (a \vee \neg b \vee d))$ sous CNF est transformée en la matrice $\begin{bmatrix} a & b & c \\ a & \neg b & d \end{bmatrix}$.

les ensembles {a} comme {a,d} sont des chemins valides.

Par contre $\{b,\neg b\}$ n'est pas un chemin valide.

$T(M)=\{\{a\},\{a,\neg b\},\{a,d\},\{b,a\},\{b,d\},\{c,a\},\{c,\neg b\},\{c,d\}\}$

$Tdm^{-1}(M)=(a \vee (b \wedge d) \vee (c \wedge d))$

Propositions.
1) Si S est une formule sous CNF alors $Tdm^{-1}(T(Tcm(S)))$ est une formule équivalente à S sous DNF.

2) Si S est une formule sous DNF alors $Tcm^{-1}(T(Tdm(S)))$ est une formule équivalente à S sous CNF.

Démonstration.
Le calcul de tous les chemins correspond à appliquer les formules de distributivité du et par rapport au ou et du ou par rapport au et. Il est donc évident que T(M) est équivalent à la formule initiale. Le seul problème pourrait provenir des chemins non valides. Or,

- Pour une formule F sous CNF, si une conjonction contient un littéral et sa négation alors l'interprétation obtenue est toujours fausse (car $X \wedge \neg X=F$), donc elle peut être détruite de la CNF (car $X \vee F=X$). De même si une conjonction contient plusieurs fois un même littéral, cette conjonction peut être simplifiée en enlevant les occurrences de ce littéral (car $X \wedge X=X$).

- Pour une formule F sous DNF, si une disjonction contient un littéral et sa négation alors l'interprétation obtenue est toujours vraie (car $X \vee \neg X =T$), donc elle peut être supprimée de la DNF (car $X \wedge T=X$). De même si une disjonction contient plusieurs fois un même littéral, cette disjonction peut être simplifiée en enlevant les occurrences de ce littéral (car $X \vee X =X$). ♦

Nous voyons donc que T(M) nous permet de transformer facilement une formule sous DNF en formule sous CNF et réciproquement.

Proposition.
Si Br est une base de règles alors $Tcm^{-1}(T(T(Tcm(Br)))$ est l'ensemble des clauses conséquences minimales de BR.

Démonstration.

Lemme 1: Si $l_1 \vee ... \vee l_n$ est une clause conséquence de Br alors il y a toujours un l_i sur chaque ligne après la première opération.

Si ce n'était pas le cas nous pourrions compléter le modèle trivalué avec $\neg l_1,...,\neg l_n$. Donc il existe un modèle qui ne contient pas $l_1,...,l_n$, donc $l_1 \vee ... \vee l_n$ est faux dans ce modèle. Donc $l_1 ... l_n$ ne peut pas être une clause conséquence de Br.

Lemme2: Si $l_1 \vee ... \vee l_n$ est une clause conséquence minimale de Br alors les littéraux $l_1 ... l_n$ apparaissent au moins une fois chacun sur une ligne.

Supposons que l_1 n'apparaisse sur aucune ligne, donc l'un des autres littéraux $l_2 ... l_n$ apparaît obligatoirement (lemme1), donc $l_2 \vee ... \vee l_n$ est une clause conséquence et donc $l_1 \vee ... \vee l_n$ ne peut pas être une clause conséquence minimale. Nous obtenons donc une contradiction. ♦

Nous avons donc montré que si $l_1 \vee ... \vee l_n$ est une clause conséquence minimale de Br alors $l_1 \vee ... \vee l_n$ est généré par T*T.

Base	matrice A	matrice B	matrice C	Resultat
¬a	¬a	¬a b c	¬a	¬a
¬a→b	a b	¬a b d	¬a b	b
b,¬c→d	¬b c d		¬a d	c d
			b ¬a	
			b	
			b d	
			c ¬a	
			c b	
			c d	

La matrice A correspond à la formule initiale sous forme normale conjonctive $\neg a \wedge (a \vee b) \wedge (\neg b \vee c \vee d)$
La matrice B est obtenue par T(A) et correspond à la forme normale disjonctive $(\neg a \wedge b \wedge c) \vee (\neg a \wedge b \wedge d)$
La matrice C est obtenue par T(B).
Le resultat correspond à la matrice C sans clauses subsumées (i.e. la forme normale conjonctive $\neg a \wedge b \wedge (c \vee d)$)

Théorème 26.

$Br \rightarrow Var(Tcm^{-1}(T(T(Tcm(Br))))$ est une opération d'achèvement complet.

Démonstration.
Dans la dernière proposition nous avons montré que $Tcm^{-1}(T(T(Tcm(Br)))$ contient l'ensemble des clauses conséquences minimales de Br. Par le théorème 16 nous avons montré que le calcul des variantes d'un ensemble de clauses conséquences bivaluées est une opération d'achèvement complet. Donc $Var(Tcm^{-1}(T(T(Tcm(Br))))$ est une opération d'achèvement complet.

Remarque.
Les clauses ou modèles subsumés doivent être détruits après chaque opération T(M) pour plus d'efficacité.

Optimisations.
La suppression des formules (clauses ou modèles) subsumées est en réalité très coûteuse. C'est pourquoi nous préconisons une optimisation de cette méthode afin de la rendre "utilisable réellement".

Tout d'abord il est préférable de calculer tous les chemins en traitant les formules une par une. Pour une formule de n littéraux on a n débuts de chemins. On croise ensuite ces chemins avec les littéraux de la formule suivante et ainsi de suite jusqu'à ce qu'il n'y ait plus de formule. Cette méthode présente l'avantage de permettre la détection des chemins subsumés et des tautologies très tôt. En effet si un début de chemin est subsumé par un autre début de chemin, il est clair que tous les chemins engendrés par le premier seront subsumés par le second. On peut alors les supprimer le plus rapidement possible et ainsi ne pas les prendre en compte dans les croisements futurs.

De plus, si un chemin en cours de construction possède un littéral en commun avec la formule traitée, tous les croisements obtenus par ce chemin et cette formule seront évidemment subsumés par ce début de chemin. Il est donc inutile de les calculer. Le début de chemin en cours reste donc inchangé.

On remarque alors qu'à chaque étape, un chemin ne peut être subsumé que par un chemin inchangé à la même étape. Imaginons le contraire. On a alors un chemin de la forme C1,x (après ajout du littéral x) qui subsume un chemin de la forme C2,y (après ajout du littéral y). Or x est différent de y sinon C1 aurait subsumé C2 à l'étape précédente. Donc x doit être inclus dans C2. Or comme C2 a été croisé par x, il a été conservé tel quel ce qui nie l'hypothèse.

Enfin un chemin C auquel on vient de rajouter un littéral x, ne peut être subsumé que par une formule inchangée (voir point précédent) contenant obligatoirement x, sinon il aurait été subsumé à l'étape précédente.

En pratique ces optimisations font gagner un temps considérable et sont donc très importantes dans toute implémentation. Nous les résumons dans l'algorithme suivant.

```
R=∅  % R contient les chemins en cours de construction

Pour chaque formule F à croiser
     Si R=∅ Alors
          considèrer chaque littéral de F comme autant de
          débuts de chemins que l'on range dans R
     Sinon
          % Prem contient les chemins inchangés
          Calculer Prem l'ensemble des formules de R qui ont un
          littéral en commun avec F et enlever ces formules de R.
          Suit=∅

          % Suit contient les nouveaux croisements
          Pour chaque formule C restante de R
               Pour chaque littéral L de F
                    Si l'opposé de L n'appartient pas à C Alors
                         Ajouter L à C pour former un nouveau
                         début de chemin C1
                         Si aucune formule de Prem contenant L
                         ne subsume C1 Alors
                                   Suit=Suit+C1
                         Fsi
                    Fsi
               Fpour
          Fpour
          R=Prem+Suit
     Fsi
Fin
```

3.4. L'arbre sémantique.

Comme nous avons montré précédemment qu'il était possible de calculer l'ensemble des clauses conséquences minimales à partir de la forme DNF, il est alors intéressant d'utiliser une méthode plus efficace que la traversée de matrices pour transformer un ensemble de formules en forme DNF. L'une de ces méthodes est dérivée de l'algorithme de Davis&Putnam [DP 67] et utilise un arbre sémantique pour construire les modèles trivalués. Cette méthode est actuellement la source de nombreuses améliorations aussi bien pour des clauses de Horn [DG 84][GU 89] que pour des clauses générales [OR 89][Sie 87].

Définition.
Nous appelons DP(Br) l'ensemble des modèles trivalués calculés par l'algorithme suivant pour une base de règles Br.

Algorithme DP(Br).

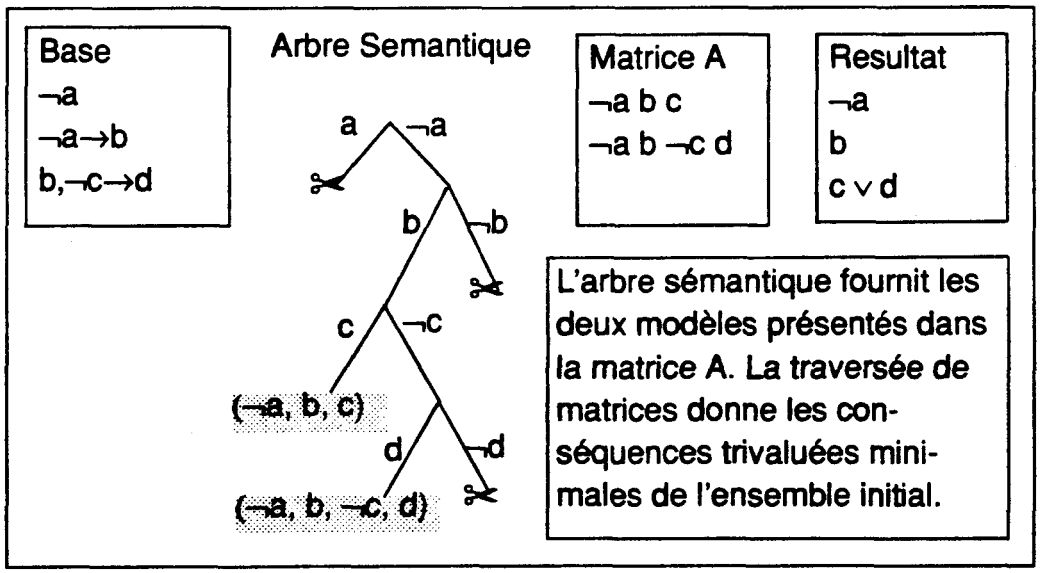
Choisir un littéral X que l'on considère vrai.

1) Si toutes les formules sont satisfaites, nous avons trouvé un modèle trivalué.

2) Si l'une des formules est fausse, l'interprétation en cours n'est pas le début d'un modèle. On affecte alors au littéral X choisi la valeur faux.

3) Dans les autres cas on conserve X à vrai, on choisit un autre littéral contenu dans une des formules non satisfaites et on recommence en 1.

La méthode que nous proposons consiste à utiliser DP pour calculer les modèles puis à réutiliser la traversée de matrices pour obtenir les clauses conséquences minimales comme on le voit dans l'exemple suivant:



Comme pour le précédent algorithme, la matrice obtenue après T(M) doit être simplifiée en supprimant les clauses subsumées. Mais contrairement à la méthode précédente, cette simplification n'a à se faire qu'une seule fois car l'ensemble obtenu par DP(Br) ne contient par construction aucun modèle subsumé.

Théorème 27.
$Br \rightarrow Var(Tcm^{-1}(T(DP(Br))))$ est une opération d'achèvement complet.

Démonstration.
Comme DP(Br) est un ensemble de modèles trivalués équivalent à Br on a DP(Kb) équivalent à T(Tcm(Br)). Comme $Var(Tcm^{-1}(T(T(Tcm(Br)))))$ est une opération d'achèvement complet on a aussitôt $Var(Tcm^{-1}(T(DP(Br))))$ est une opération d'achèvement complet.♦

En termes de complexité le croisement de matrices dépend de la longueur moyenne des clauses tandis que les arbres sémantiques dépendent uniquement du nombre d'atomes de la base, cette dernière méthode est donc plus efficace en général que les autres méthodes présentées. De plus comme il a été dit en remarque, les tests de subsomption n'ont à être effectués qu'une seule fois, ce qui accélère d'autant plus cette méthode.

Heuristiques.
L'un des avantages de cet algorithme est qu'il est très facile à optimiser pour obtenir de bonnes performances. Voir [OR 89] pour avoir une méthode de traitement efficace en utilisant notamment son théorème de partition du modèle et sa structure de données très adaptée. Il est à noter de plus que l'ensemble initial n'a pas à être transformé en forme CNF pour être traité. En effet l'arbre sémantique peut très facilement traiter des formules quelconques à partir du moment où l'on sait calculer leurs valeurs de vérité.

[OR 89] fournit des heuristiques pour favoriser son théorème de partition du modèle, mais pour notre problème les heuristiques les plus utiles sont celles qui ont pour but de minimiser l'ensemble de modèles calculé par DP. Nous en présentons deux qui ont donné de très bons résultats sur de nombreux exemples et qui sont les plus efficaces en général.

a) choix de l'atome à évaluer

Pour chaque formule sous forme CNF il existe bien évidemment plusieurs formules sous forme DNF équivalents. Il est donc très avantageux d'obtenir la plus courte, ce qui allégera d'autant les croisements à effectuer par la suite.

Par exemple l'ensemble de deux règles ($A \rightarrow B$; $B \rightarrow C$) est équivalent à la formule CNF $((\neg A \vee B) \wedge (\neg B \vee C))$ qui est elle-même équivalente aux formules $((\neg A \wedge \neg B) \vee (A \wedge B \wedge C) \vee (\neg A \wedge B \wedge C))$ et $((\neg A \wedge \neg B) \vee (B \wedge C))$. La dernière formule étant bien sûr la plus intéressante.

L'heuristique que nous proposons consiste à choisir comme atome à évaluer, l'atome qui est présent dans le maximum de clauses. Cette heuristique permet d'obtenir en général moins de modèles ce qui accélère considérablement la traversée de matrices. Dans l'exemple précédent la première forme DNF est obtenue en choisissant les atomes A,B et C dans l'ordre tandis que la seconde est obtenue en choisissant d'abord B qui est présent deux fois puis A et C.

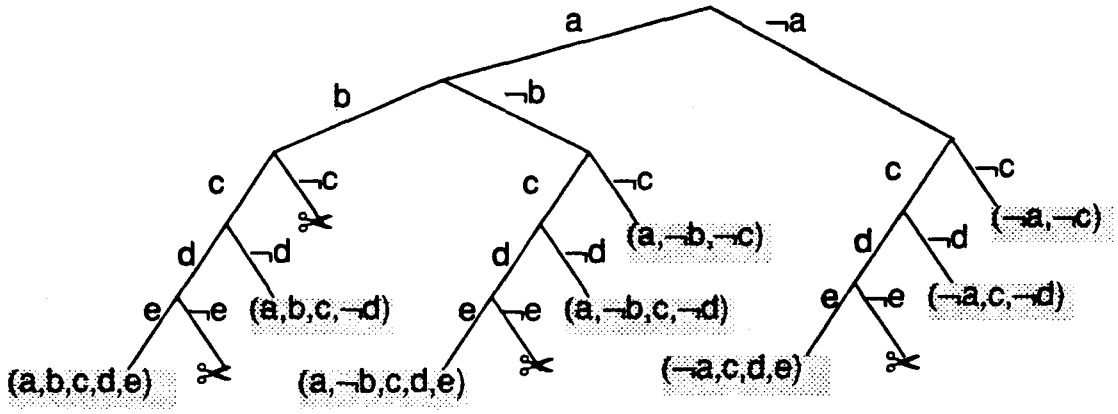
En cas de conflit, c'est-à-dire si plusieurs atomes sont représentés le même nombre de fois, on regarde si l'un de ces atomes est présent dans une clause mono-littérale. Si c'est le cas on

choisira cet atome de préférence aux autres.

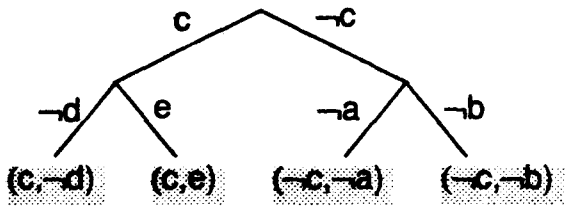
b) fin de dérivation.

L'algorithme DP présenté dans sa forme brute est trop systématique en fin de chaque branche conduisant à un modèle. En effet si l'ensemble de clauses restant à satisfaire se réduit à une clause de la forme $c_1 \vee ... \vee c_n$, le début de modèle construit étant de la forme $(x_1,...,x_m)$ chaque littéral $x_1,...,x_m$ satisfait la dernière clause et donc DP calcule $(x_1,...,x_m,c_1)$ puis $(x_1,...,x_m,\neg c_1,c_2)$, ... etc jusquà $(x_1,...,x_m,\neg c_1,...,\neg c_{n-1},c_n)$. Or il est évident que les $\neg c_i$ figurant dans les modèles sont inutiles. L'heuristique que nous proposons s'applique donc dès que l'ensemble à satisfaire se réduit à une clause unique de la forme $c_1 \vee ... \vee c_n$, le début de modèle construit étant de la forme $(x_1,...,x_m)$ auquel cas l'algorithme produit directement les modèles $(x_1,...,x_m,c_1)$, ... ,(x_1,x_m,c_n). Cette heuristique permet d'obtenir des modèles en général plus petits ce qui accélère d'autant la traversée de matrices.

Nous présentons ici un exemple traité de deux manières différentes. La première sans heuristiques en choisissant les littéraux dans l'ordre de leur apparition et la seconde en utilisant les deux heuristiques précédentes. La base à traiter ne comporte que deux règles Br={A,B→C , C,D→E} qui sous forme de clauses donne C={¬A∨¬B∨C, ¬C∨¬D∨E}.



Nous obtenons 8 modèles qui nécessiteront 28800 croisements (produit des longueurs). Ce n'est pas le pire cas puiqu'en prenant l'ordre A,B,D,E,C on aurait plus de modèles



En utilisant les deux heuristiques précédentes (choix du littéral et fin de dérivation) on ne calcule plus que 4 modèles qui ne nécessiteront que 16 croisements.

Remarque.

Nous avons montré qu'appliquer deux fois la traversée de matrices (T*T) est équivalent à appliquer l'arbre sémantique puis la traversée de matrices (DP*T). Il est alors normal de se demander si deux fois l'arbre sémantique (DP*DP) est équivalent aux deux autres algorithmes.

Malheureusement non, car l'arbre sémantique ne permet pas de calculer toutes les clauses conséquences minimales en général. Par exemple, soit Br={A→B , A→C ,B→D} qui sous forme de clauses donne C={¬A∨B, ¬A∨C, ¬B∨D}. l'arbre sémantique calcule comme modèles {(A,B,C,D),(¬A,D),(¬A,¬B)}. Si on applique une nouvelle fois cette méthode à l'ensemble obtenu on obtient comme conséquences {(A,D,¬B),(¬A,B),(¬A,C),(¬A,D)} et l'on peut voir que (¬B,D) est une conséquence minimale non calculée. Ceci provient du fait que l'arbre sémantique ne remet jamais en cause un atome choisi (seul son signe peut être remis en cause) tandis que la traversée de matrices permet de remettre en cause n'importe quel littéral.

Pour terminer précisons qu'il existe évidemment d'autres méthodes pour calculer l'ensemble des conséquences minimales d'une base de règles basées notamment sur la SL-résolution [KK 71] utilisée dans [Sie 87] qui sert de support au traitement des contraintes booléennes de Prolog III [Col 87], Les diagrammes de Karnaugh utilisés en électronique ou encore la représentation de Bryant [Bry 86] utilisée dans [BS 87] pour résoudre le problème de l'unification booléenne du langage Chip [Dinc 88][VEnt 89][RT 90] et que d'autres sont encore à imaginer.

4. Réduction des bases achevées.

Le théorème 19 montre qu'une étape de chaînage avant est suffisante pour déduire tous les littéraux. Ceci n'est évidemment pas très satisfaisant car cela implique que de nombreuses règles ont été ajoutées à la base. Parmi celles-ci, certaines sont inutiles si l'on utilise un chaînage avant véritable (plusieurs passes) pour les exploiter.

Ex: br: A→B
B→C

AR(Br):

A	→B
¬B	→¬A
B	→C
¬C	→¬B
A	→C
¬C	→¬A

Dans cet exemple les deux dernières règles expriment la transitivité et sont par conséquent inutiles.

C'est pourquoi nous proposons à partir des clauses et juste avant le calcul des variantes, une méthode de réduction de la base achevée.

Définition.
Soit Br une base de règles, on dit que Br est une base redondante si il existe une règle R de la forme Prem → Conc telle que Conc ∈ Chavt((Br-R)∪Prem)

En d'autres termes s'il est possible d'obtenir Conc par chaînage avant à partir de Prem sans utiliser la règle R.

Définition.
Soit Br une base de règles, on dit que Br est une base non redondante si Br n'est pas redondante, c'est-à-dire si pour toute règle R de Br, R étant de la forme Prem → Conc, Conc ∉ Chavt((Br-R)∪Prem).

Définition.
On appelle opération de réduction toute opération sur une base de règles ou une base de clauses qui permet d'obtenir une base non redondante.

4.1. Réduction sur les variantes.

Définition.
On appelle Réduc l'opération suivante:

Algorithme de calcul de Réduc(Br)

```
Soit Br une base de règles.
Pour toute règle R de Br de la forme Prem→Conc prise dans l'ordre
de la base
     Si Conc∈ Chavt((Br-R)∪Prem)
          enlever R de Br
     Fsi
Fpour
```

Proposition.
Soit Br une base de règles, si R est une règle redondante dans Br alors Chavt(Bf∪Br)=Chavt(Bf∪(Br-R))

Démonstration.
Soit R de la forme Prem→Conc, par définition si R est supprimée c'est que Conc peut être obtenue à partir de Prem dans Br-R. Donc toute déduction possible dans Br est toujours possible dans Br-R.♦

Proposition.
Soit Br une base de règles, Bf une base de faits et Br'=Réduc(Br),
Chavt(Bf∪Br) = Chavt(Bf∪Br')

Démonstration.
Par définition de Réduc, si une règle est supprimée c'est que les déductions qu'elle permet de faire peuvent être faites à l'aide des autres règles. Par récurrence on montre que les déductions

possibles avec la base initiale peuvent être faites avec la base réduite.♦

Proposition.
Réduc est une opération de réduction.

Démonstration.
Par définition de l'algorithme toute règle de la forme Prem→Conc telle que Conc ∈ Chavt((Br-R)∪Prem) est supprimée. Réduc(Br) ne contient alors plus de règles redondantes. Réduc(Br) est donc une base non redondante.♦

Remarque.
La définition de Réduc garde un sens même si la base de règles n'est pas achevée.

Théorème 28.
Réduc(AR(Br)) est une base achevée non redondante.

Démonstration.
Dérive directement du fait que AR est une opération d'achèvement complet et des deux propositions précédentes qui montrent que Réduc conserve les déductions initiales et génère une base non redondante.♦

Remarque.
Selon l'ordre dans lequel les règles sont testées, la base réduite pourra être différente voire même avec un nombre différent de règles.

Il est évident que cet ordre dépend aussi de l'ordre des règles au départ, de la manière dont la saturation est effectuée ainsi que les simplifications.

Les bases obtenues ne sont donc pas forcément minimales en nombre de règles.

Exemple.
(Un exemple avec une base achevée serait complexe, aussi cette base n'est pas complètement achevée mais en contrepartie elle est courte et facilement compréhensible!)

1	a → b	5+4	Si les règles sont testées de haut en bas il
2	b → a		reste 3 règles (2, 4,5).
3	c → a	4+2	Si les règles sont testées de bas en haut il
4	c → b	3+1	reste 4 règles (5,3,2,1).
5	a → c		

4.2. Réduction sur les clauses.

4.2.1. Réduction par chaînage avant.

Avec le calcul des variantes un nouveau problème apparaît: Le très grand nombre de règles générées. On peut alors restreindre le nombre de règles de la base achevée en effectuant une modification du moteur d'inférences utilisé de manière à ce qu'il ne traite non plus des règles mais des clauses. D'une part le calcul des variantes n'est plus nécessaire mais de plus la base voit sa taille réduire de manière considérable.

Dans l'algorithme de chaînage avant précédemment décrit une règle de la forme $L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow L$ était utilisée si $L_1, L_2, ..., L_n \in Br$ et $L \notin Br$. Dans ce cas L était rajouté à la base.

Cette fois-ci nous dirons qu'une clause de la forme $L_1 \vee ... \vee L_n$ sera sélectionnée si sur les n littéraux de la clause, les opposés de n-1 sont présents dans la base. Dans ce cas le nième sera rajouté à la base.

On obtient alors l'algorithme suivant:

Algorithme de chaînage avant sur les clauses: Chavtcl

```
Soit S un ensemble de clauses.
Tant qu'il existe une clause de S: L1∨...∨Ln, telle que:
     il existe n-1 littéraux de cette clause présents sous forme
     négative dans S et que le nième (noté L) n'est pas présent
          choisir une telle clause
     si ¬L est déjà dans S alors
          renvoyer 'Base inconsistante au sens trivalué'
          arrêt.
     sinon ajouter L à S.
fin de tant que
renvoyer S.
arrêt.
```

Nous avons montré qu'un chaînage avant sur les clauses tenant compte du calcul des variantes pouvait être réalisé. Il est alors très intéressant de récupérer l'algorithme de Réduc et le faire travailler sur des clauses.

Le gain en place mémoire est alors conséquent malheureusement la complexité temporelle reste la même. En effet, si une règle est redondante ses variantes ne sont pas forcément redondantes.

Une opération de réduction sur les clauses oblige donc à tester chaque littéral de chaque clause et devient donc aussi coûteuse en temps que l'opération de réduction sur les règles.

Exemple.

Base	Achèvement	variantes
$a,b \rightarrow c$	$\neg a, \neg b, c$	$a, b \rightarrow c$
$\neg a \rightarrow d$	a, d	$a, \neg c \rightarrow \neg b$
$\neg b \rightarrow d$	b, d	$b, \neg c \rightarrow \neg a$
	c, d	$\neg a \rightarrow d$
		$\neg d \rightarrow a$
		$\neg b \rightarrow d$
		$\neg d \rightarrow b$
		$\neg c \rightarrow d$
		$\neg d \rightarrow c$

La règle $\neg d \rightarrow c$ est redondante mais pas la règle $\neg c \rightarrow d$.

Définition.
On appelle Réduc-cl l'opération suivante:

Algorithme de réduction sur les clauses: Réduc-cl

Soit Cl une base de clauses.
Pour toute clause C $(x_1 \vee \ldots \vee x_n)$ de Cl prise dans l'ordre de la base
    Si $\forall x_i$, $x_i \in$ Chavt-cl(Br$\cup\{x_1,\ldots,x_n\}-\{x_i\}$) alors
        (toutes les variantes sont redondantes)
        enlever C de Cl
    Fsi
Fpour

Théorème 29.
Réduc-cl(AR(Br)) est une base achevée non redondante.

Démonstration.
Démonstration analogue au théorème 28.♦

Remarque.
Comme précédemment, l'ordre dans lequel les clauses sont testées est capital. La base obtenue n'est pas forcément minimale en nombre de règles.

4.2.2. Réduction par test de résolution directe.

Il est encore intéressant de remarquer que puisque l'ensemble de clauses obtenu après nos deux algorithmes d'achèvement sont saturés par résolution, si une clause peut être obtenue par résolution de deux autres clauses, ce ne peut être que par résolution directe.

On peut alors espérer que si une clause C peut être obtenue par résolution de deux autres clauses alors les variantes de C sont des règles redondantes.

Malheureusement ce n'est pas le cas comme le montre l'exemple suivant.

Base	Clauses achevées
$a,b \rightarrow c$	$\neg a, \neg b, c$
$b,c \rightarrow d$	$\neg b, \neg c, d$
	$\neg a, \neg b, d$

la clause ajoutée est obtenue par résolution directe sur les deux autres mais toutes ses variantes ne sont pas redondantes (voir $a,\neg d \rightarrow \neg b$)

On remarque cependant que ce problème provient du fait que les deux clauses ont des littéraux en commun. Il est alors possible d'en déduire le théorème suivant:

Théorème 30.
Soit S un ensemble de clauses saturées par Rsv_S et C,C1 et C2 des clauses de S, si C est une résolvante directe de C1 et C2 et que $C1 \cap C2 = \varnothing$ alors les variantes de C sont redondantes.

Démonstration.
Si C1 est de la forme $(x \vee a_1 \vee ... \vee a_n)$ et C2 est de la forme $(\neg x \vee b_1 \vee ... \vee b_m)$ alors C est de la forme $(a_1 \vee ... \vee a_n \vee b_1 \vee ... \vee b_m)$ puisque $C1 \cap C2 = \varnothing$.

Soit la règle $\neg a_2,...,\neg a_n,\neg b_1,...,\neg b_m \rightarrow a_1$ tirée de C, on a la règle $\neg b_1,...,\neg b_m \rightarrow \neg x$ tirée de C2 ainsi que la règle $\neg x,\neg a_2,...,\neg a_n \rightarrow a_1$ tirée de C1 puisque toutes les variantes sont ajoutées. Donc avec $\neg a_2,...,\neg a_n,\neg b_1,...,\neg b_m$ on obtient a_1 par chaînage avant sans utiliser les règles tirées de C.

Un raisonnement analogue peut être effectué pour toutes les variantes de C ce qui nous permet de montrer que toute déduction obtenue grâce à une variante de C peut être obtenue par deux règles tirées de C1 et C2.♦

Plusieurs méthodes sont utilisables pour réduire les clauses de cette manière. La méthode la plus efficace en général consiste à tester les clauses dans l'ordre opposé à leur apparition. Comme précédemment la base obtenue n'est pas forcément minimale en nombre de règles.

Définition.
On appelle Réduc-res l'opération suivante:

Algorithme de réduction par résolution: Réduc-res

```
Soit Cl une base de clauses.
Pour toute clause C de Cl prise dans l'ordre inverse de la créa-
tion des clauses
     Si C peut être obtenue par résolution directe de deux
     clauses C1 et C2 de Cl telles que C1 et C2 n'ont pas de
     littéraux communs alors
          enlever C de Cl
     Fsi
Fpour
```

Exemple.

Base	Clauses	Achèvement		Réduction
a→¬d	¬a,¬d	¬a,¬d	1	¬a,¬d
¬d→c	d,c	d,c	2	d,c
c→e	¬c,e	¬c,e	3	¬c,e
e→¬b	¬e,¬b	¬e,¬b	4	¬e,¬b
		¬a,c	5=1+2	
		d,e	6=2+3	
		¬c,¬b	7=3+4	
		¬a,e	8=1+6	On teste les clauses de la 10 à la 1.
		d,¬b	9=4+6	(Si les clauses étaient testées de
		¬a,¬b	10=4+8	haut en bas, il en resterait 6).

Remarque.
Cette méthode de réduction est très facilement réalisable après utilisation de l'algorithme AR (achèvement par résolution). Il suffit en effet à chaque résolution de garder les ∀noms∀ des parents de la résolvante, puis en fin d'algorithme, supprimer au fur et à mesure les clauses dont les parents sont encore présents et sans littéraux communs, en testant les clauses dans l'ordre inverse de leur apparition.

Proposition.
Réduc-res(AR(Br)) n'est pas en général une base non redondante.

Exemple.

Base	Achèvement		Réduc-cl	
$a \rightarrow b$	$b, \neg a$	1	$b, \neg a$	
$b \rightarrow e$	$e, \neg b$	2	$e, \neg b$	
$a \rightarrow f$	$f, \neg a$	3=1+6;4+5	$f, \neg a$	Cette clause est encore redondante
$e \rightarrow f$	$f, \neg e$	4	$f, \neg e$	
	$e, \neg a$	5=1+2		
	$f, \neg b$	6=2+4		

Certaines clauses qui répondaient au critère de suppression à un moment donné, n'y répondent plus ensuite. Ceci montre l'importance de l'ordre dans lequel les clauses sont testées ainsi que l'importance de l'ordre d'origine et la manière dont la saturation est effectuée.

On remarque d'ailleurs que si la réduction était faite de haut en bas ou si les règles étaient disposées dans un autre ordre au départ il ne resterait plus que 3 clauses.

Proposition.
Deux bases achevées et équivalentes entre elles au sens bi et trivalué peuvent être différentes (même sous forme clausale).

Voir ci-dessous les bases constituées par les clauses 1,2,3 et 4,5,6.

Exemple.

Base	Clauses	Achèvement		Résultat 1	Résultat 2
$a \rightarrow \neg b$	$\neg a, \neg b$	$\neg a, \neg b$	1=4+5	$a \rightarrow \neg b$	$a \rightarrow \neg c$
$\neg b \rightarrow \neg c$	$b, \neg c$	$b, \neg c$	2=4+6	$b \rightarrow \neg a$	$c \rightarrow \neg a$
$\neg a \rightarrow c$	a, c	a, c	3=5+6	$c \rightarrow b$	$b \rightarrow c$
		$\neg a, \neg c$	4=1+2	$\neg b \rightarrow \neg c$	$\neg c \rightarrow \neg b$
		$\neg b, c$	5=1+3	$\neg a \rightarrow c$	$\neg b \rightarrow a$
		a, b	6=2+3	$\neg c \rightarrow a$	$\neg a \rightarrow b$

Résultat 1 est obtenu par réduction de haut en bas et résultat 2 est obtenu par réduction de bas en haut.

Toute déduction effectuée sur l'une des deux bases peut être effectuée sur l'autre.

5. Complexité de la base obtenue.

Pourquoi parler de complexité de la base obtenue sans parler de complexité des algorithmes utilisés? Il y a en fait deux raisons à cela. La première est que la vérification de l'inconsistance d'un ensemble de clauses du calcul propositionnel appartient à la classe des problèmes NP-complets. Il en résulte donc que tous les algorithmes présentés ici (ainsi que d'autres que l'on pourrait imaginer) sont de complexité exponentielle. Il n'y a donc aucun espoir d'obtenir un

algorithme efficace dans le cas général, bien qu'en pratique la majorité des cas se résolvent en moins d'une minute. La seconde raison que l'on pourrait invoquer est que nous présentons une méthode de compilation. Ce n'est donc plus le temps de compilation qui est important (tâche qui peut s'exécuter en tâche de fond, la nuit, etc...) mais bel et bien la taille de la base obtenue puisque c'est elle qui déterminera le temps que prendra les exécutions futures.

Nous traiterons donc ici de la complexité de la base obtenue, ou en d'autres termes de sa taille maximale. Soit N le nombre d'atomes différents dans la base initiale, il est clair que toutes les clauses constructibles (qui ne sont pas des tautologies) sont au nombre de 3^N car chaque atome peut être soit positif soit négatif soit absent. Mais dans ce cas de nombreuses clauses se subsument. Par ailleurs plus une clause est petite, plus elle subsume de clauses. Donc pour obtenir le plus grand ensemble de clauses non subsumées il faut prendre toutes les clauses de N atomes. Il y en a 2^N. D'autres se subsument encore car il ne peut y avoir deux clauses de même taille, qui diffèrent d'un seul littéral figurant sous forme positive dans la première et sous forme négative dans la seconde (car sinon la résolvante les subsumerait). Notre ensemble est alors divisé par deux et l'on obtient au maximum 2^{N-1} clauses.

Un tel ensemble peut se construire très facilement de la manière suivante: Enlever un atome X aux N atomes initiaux. Il en reste donc N-1. Calculer toutes les clauses de N-1 littéraux que l'on peut construire avec ces N-1 atomes (il y en a 2^{N-1}). Considérer chacune de ces clauses comme les prémisses d'une règle qui conclut soit sur X si il y a un nombre pair de littéraux négatifs en prémisses, soit sur $\neg X$ si il y a un nombre impair de littéraux négatifs en prémisses.

Exemple:

Pour N=4 (A,B,C,D par exemple).

$$
\begin{array}{ll}
B, C, D & \rightarrow A \\
B, C, \neg D & \rightarrow \neg A \\
B, \neg C, D & \rightarrow \neg A \\
B, \neg C, \neg D & \rightarrow A \\
\neg B, C, D & \rightarrow \neg A \\
\neg B, C, \neg D & \rightarrow A \\
\neg B, \neg C, D & \rightarrow A \\
\neg B, \neg C, \neg D & \rightarrow \neg A
\end{array}
$$

Sur cet exemple, l'achèvement semble être linéaire par rapport au nombre de règles. Si on ne prend que la première règle on obtient une base de 3*1 règles achevées (les trois variantes). Pour les n premières règles on obtiendra 3*n règles achevées (ou n clauses).

On notera de plus que pour un nombre d'atomes constant, plus il y a de règles initiales moins il y a de modèles et donc moins il y a de conséquences en général. Notre méthode ne dépend donc pas du nombre de règles. Elle ne dépend pas non plus directement du nombre d'atomes. Il est en effet très facile de construire un exemple avec un nombre de règles constant et un nombre d'atomes croissant qui fournisse un nombre de règles achevées linéaire. Il suffit pour

cela de construire une base d'une seule règle ayant N-1 atomes en prémisses et le $N^{ième}$ en conclusion. La base achevée obtenue contiendra uniquement les N variantes de la règle initiale (ou 1 clause). Malheureusement cela ne prouve pas que notre compilation n'est pas exponentielle.

Comment interpréter alors ces deux résultats? Doit-on mesurer l'importance d'une base à son nombre de règles ou à son nombre d'atomes? Sans aucun doute les deux. C'est pourquoi l'efficacité théorique de cet algorithme est difficilement calculable. Seule la pratique permettra de juger son efficacité. Nous avons testé notre méthode sur un nombre très important de cas de toutes tailles et tous se sont révélés linéaires. En guise d'exemples et pour convaincre le lecteur nous présentons ici quelques structures non achevées et nous calculons la complexité de la base obtenue.

Base de règles	
$\begin{cases} A_1 \rightarrow A_2 \\ \neg A_3 \rightarrow \neg A_2 \\ A_3 \rightarrow A_4 \\ \ldots\ldots \\ \neg A_{n-1} \rightarrow \neg A_{n-2} \\ A_{n-1} \rightarrow A_n \end{cases}$	la base initiale contient n-1 règles et n atomes. AR(Br) contient $\frac{n(n-1)}{2}$ clauses. La base compilée contient n-1 clauses (ou 2(n-1) clauses car seules les variantes suffisent). Ex: n=5 donne 4 clauses ou 8 règles. n=10 donne 9 clauses ou 18 règles.

Base de règles	
$\begin{cases} A_1, \ldots, A_n \rightarrow B \\ \neg A_1 \rightarrow C \\ \neg A_2 \rightarrow C \\ \ldots\ldots \\ \neg A_n \rightarrow C \end{cases}$	la base initiale contient n+1 règles et n+2 atomes. AR(Br) contient n+2 clauses. La base compilée contient n+2 clauses (les mêmes + b∨c) ou 3n+2 règles (les variantes + b→c).

Base de règles	
$\begin{cases} A_1, ..., A_n \rightarrow B \\ \neg A_1 \rightarrow C_1 \\ \neg A_2 \rightarrow C_2 \\ ..... \\ \neg A_n \rightarrow C_n \end{cases}$	la base initiale contient n+1 règles et 2n+1 atomes. AR(Br) contient 2^n+n clauses (donc exponentiel). La base compilée contient n+1 clauses (les mêmes) ou 3n+1 règles (donc linéaire).

Base de règles	
$i=1...n \begin{cases} A_1^i, A_2^i \rightarrow B^i \\ \neg A_1^i \rightarrow C \\ \neg A_2^i \rightarrow C \end{cases}$ $B^1, ..., B^n \rightarrow D$	la base initiale contient 3n+1 règles et 3n+2 atomes. AR(Br) contient 2^n+4n+1 clauses (donc exponentiel). La base compilée contient 4n+2 clauses (les mêmes + les $b^i \vee c$ + $c \vee d$).

Base de règles	
$i=1...n \begin{cases} A_1^i, A_2^i \rightarrow B^i \\ \neg A_1^i \rightarrow C^i \\ \neg A_2^i \rightarrow C^i \end{cases}$ $B^1, ..., B^n \rightarrow D$	la base initiale contient 3n+1 règles et 4n+1 atomes. AR(Br) contient 3^n+4n clauses (donc exponentiel). La base compilée contient 4n+1 clauses (les mêmes + les $b^i \vee c^i$).

Base de règles	
$i=1...n \begin{cases} A_1^i, A_2^i \rightarrow B^i \\ \neg A_1^i \rightarrow C^{i+1} \\ \neg A_2^i \rightarrow C^{i+1} \\ B^i \rightarrow \neg C^i \end{cases}$	la base initiale contient 4n règles et 4n+1 atomes. AR(Br) contient $6n^2$+n clauses (donc polynomial). La base compilée contient 5n clauses (les mêmes + les $b^i \vee c^{i+1}$). Et pourtant il manque les règles de la forme $C^i \rightarrow C^{i+1}$.

Base de règles	
$i=1...n \begin{cases} A_{2i}, A_{2i+1} \rightarrow D_i \\ \neg E_i, \neg F_i \rightarrow \neg D_i \\ E_i \rightarrow A_i \\ F_i \rightarrow A_i \end{cases}$	la base initiale contient 4n règles et 4n+1 atomes. La base compilée contient 5n clauses (les mêmes + les $a_i \vee \neg d_i$).

6. Calcul des clauses conséquences.

Pendant longtemps, les recherches en démonstration automatique et en programmation logique se sont attachées à découvrir des méthodes permettant de dire si oui ou non une clause est conséquence d'un ensemble de clauses. Ce problème a abouti notamment à toutes les méthodes citées précédemment comme la résolution, la SL-résolution, la procédure de Davis et Putnam ou encore les résolutions Input-ordonnées qui ont donné naissance au langage Prolog. On trouvera un panorama complet de toutes ces méthodes dans [CL73][Lov 78][Kow 79] et plus récemment dans [Del 86] et [FG 87]. Malheureusement dans le cas général, ces méthodes sont soit exponentielles pour les unes soit incomplètes pour les autres. Il serait donc intéressant de pouvoir répondre en un temps polynomial à cette question. Il est par exemple possible de stocker toutes les conséquences bivaluées minimales dans un fichier et de vérifier à chaque fois si la clause recherchée est dedans, mais malheureusement l'ensemble de toutes les conséquences est souvent très gros et donc très coûteux à stocker. Par contre, une base compilée logiquement est relativement petite puisqu'elle semble ne croître que de manière linéaire. De plus le chaînage avant est de complexité polynomiale ce qui permet un calcul très rapide. Nous proposons donc d'adapter la compilation logique et l'achèvement des bases de règles au calcul des clauses conséquences d'une base de règles par le théorème suivant:

Théorème 31.
Soit Br une base complètement achevée et C une clause,
$Br \models C$ avec C de la forme $c_1 \vee ... \vee c_n$ ssi $c_1 \subset Chavt(Br \cup \{\neg c_2, ..., \neg c_n\})$.

Démonstration.
L'un des théorèmes de base de la démonstration automatique (Voir Théorème 5) montre que si C ($c_1 \vee ... \vee c_n$) est une clause conséquence d'un ensemble de clauses S alors $S \cup \neg C$ est insatisfiable.

De la même manière que pour le théorème 13 on montre que si $S \cup \neg C$ est insatisfiable alors $S \cup \{\neg c_2, ..., \neg c_n\} \models c_1$. Or nous savons que pour une base complètement achevée le chaînage

avant calcule toutes les conséquences bivaluées de n'importe quelle base de faits ajoutée par la suite. Il en résulte immédiatement que dire que Br est une base complètement achevée et que Br a pour conséquence bivaluée une clause C de la forme $c_1 \vee ... \vee c_n$ est équivalent à $c_1 \subset Chavt(Br \cup \{\neg c_2,...,\neg c_n\})$.♦

Exemple.

Base initiale	Base compilée.
SI a ET b ALORS d.	SI d ALORS non(c).
SI non(a) ALORS e.	SI c ALORS non(d).
SI non(b) ALORS e.	SI a ET b ALORS d.
SI c ALORS non(d).	SI non(d) ET b ALORS non(a).
	SI non(d) ET a ALORS non(b).
	SI non(e) ALORS a.
	SI non(e) ALORS b.
	SI non(a) ALORS e.
	SI non(b) ALORS e.
	SI non(d) ALORS e.

Nous voulons savoir si $\neg c \vee e$ est conséquence de la base initiale. Il suffit pour cela de lancer un chaînage avant sur la base compilée à partir de c comme base de faits, ce qui nous donne non(d) et e. Donc $\neg c \vee e$ est conséquence de la base initiale. On aurait d'ailleurs aussi bien pu prendre pour base de faits $\neg e$, ce qui nous aurait donné a,b,d,non(c).

7. Extension du langage de règles.

Un problème important du chaînage avant classique est d'être complètement inadapté au traitement des règles avec des 'ou' en partie conclusion.

Par exemple une règle du type: SI NON voyage cher ALORS train OU voiture

n'est pas acceptée alors qu'elle exprime une connaissance réelle.

Dans les chapitres précédents nos méthodes s'appliquaient aux ensembles de clauses. Chaque base de règles devait initialement être transformée en ensemble de clauses avant d'appliquer les méthodes d'achèvement.

Pourquoi alors se restreindre à des bases de règles simples. En effet, comme toute règle peut être transformée en un ensemble de clauses, on peut tout à fait étendre le langage de base aux règles à conclusions dubitatives (avec des 'ou' en partie conclusion). Le sens du connecteur 'OU' est alors le sens bivalué usuel. Nos méthodes nous permettent donc de transformer une base de règles avec des 'ou' en conclusion en une base de règles équivalente au sens bivalué mais sans 'ou' en conclusion, donc exploitable par un chaînage avant classique. D'autres

approches avaient été faites dans ce sens notamment par Terrier, Descamps et Bonnemay dans [TDB 86] et [Ter 87] mais seul le problème du 'ou' était traité. Leur méthode aurait donc du être classée parmi les méthodes incomplètes. Cette méthode consiste à utiliser récursivement les trois règles de réécriture suivantes:

% Si (a ou b) est vrai et b est faux alors a est vrai.

$a \rightarrow (b \vee c)$	$a \rightarrow (b \vee c)$
	$(b \vee c) \wedge \neg c \rightarrow b$
	$(b \vee c) \wedge \neg b \rightarrow a$

% Transitivité

$a \rightarrow b$	$a \rightarrow b$
$b \rightarrow c$	$b \rightarrow c$
	$a \rightarrow c$

% réunion

$a \rightarrow c$	$a \vee b \rightarrow c$
$b \rightarrow c$	

... Puis à considérer chaque disjonction comme un fait à part entière.

Malheureusement aucune preuve formelle n'a été donnée dans les articles cités sur la complétude de cette méthode relativement au 'ou' en conclusion, ce qui d'ailleurs serait insuffisant car nous avons présenté ici de nombreux exemples (voir II.3) de 'ou cachés' qui ne sont évidemment pas traités par cette méthode puisqu'ils ne figurent pas en conclusion de règles. Ce traitement est donc incomplet relativement à la logique bivaluée contrairement à la compilation logique comme nous pouvons le voir sur les exemples suivants.

Exemple.

Base	Base achevée réduite.
$x \rightarrow a \vee b$	$\neg b, x \rightarrow a$
$y \rightarrow c$	$\neg a, x \rightarrow b$
$c, z \rightarrow \neg b$	$\neg a, \neg b \rightarrow \neg x$
	$y \rightarrow c$
	$\neg c \rightarrow \neg y$
	$b, c \rightarrow \neg z$
	$z, c \rightarrow \neg b$
	z et $b \rightarrow \neg c$

Avec x, y et z on en déduit bien le littéral a

De même avec des bases plus complexes comme celle-ci:

$u \rightarrow a \vee b$
$v \rightarrow c \vee d$
$a, w \rightarrow e$
$e, x \rightarrow f$
$b, y \rightarrow \neg c$
$d, z \rightarrow f$

On obtient bien f à partir de u, v, w, x, y et z par chaînage avant sur la base achevée réduite.

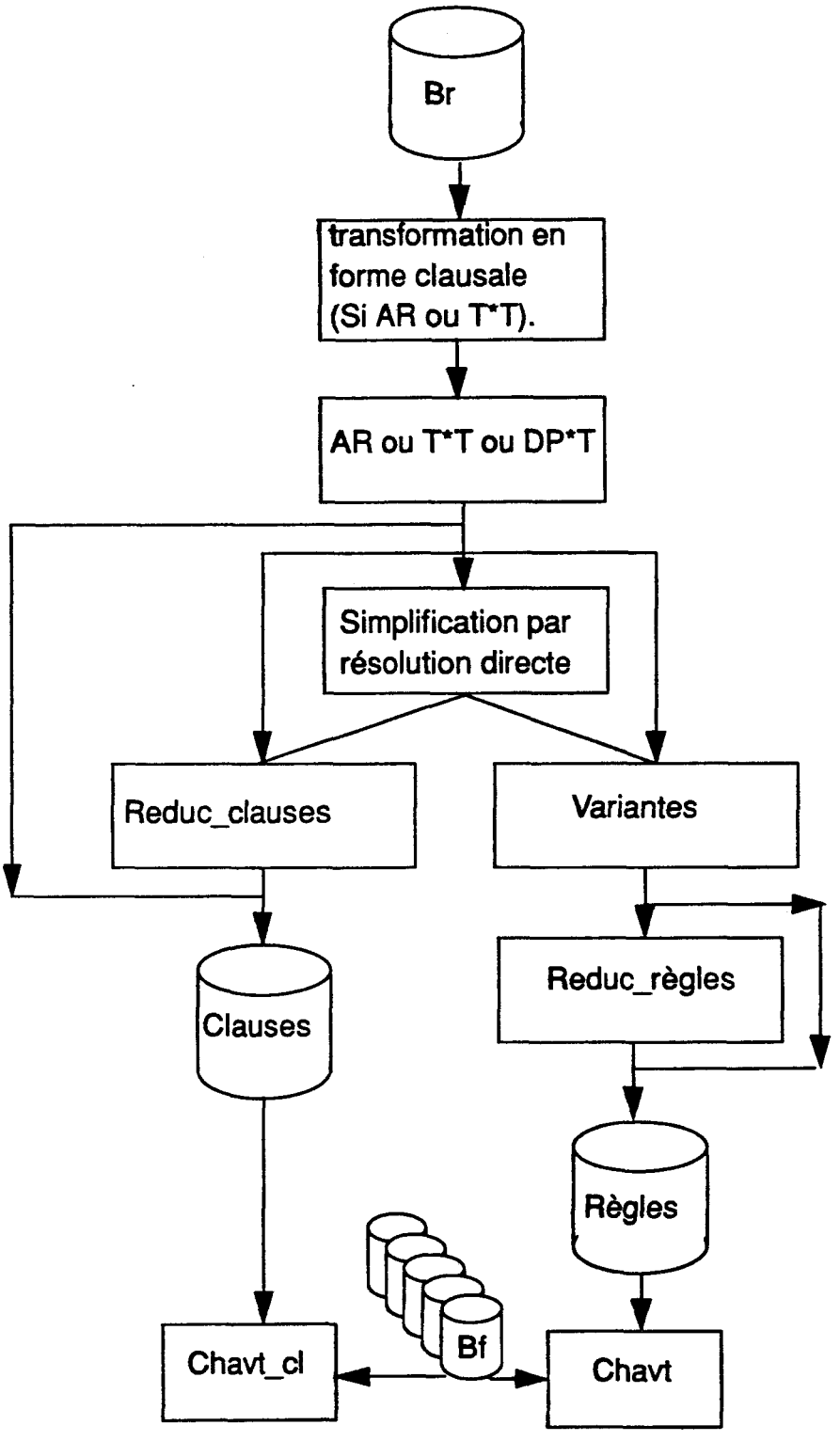
8. Conclusion

Après avoir défini les concepts d'achèvement, d'achèvement relatif, d'achèvement partiel et d'achèvement complet, nous avons décrit quatre opérations qui permettent d'achever complètement une base de connaissances construite avec un langage étendu ('ou' en conclusion ou clauses). L'achèvement par contradictions bivaluées, qui n'est pas efficace, l'achèvement par résolution, plus efficace et surtout facilement applicable aux bases avec variables sans symboles de fonctions, l'achèvement par traversée de matrice, sans doute le plus simple à concevoir et enfin l'achèvement par arbre sémantique, de loin la plus efficace de ces quatre méthodes. Cet achèvement permet alors de calculer les littéraux conséquences bivaluées d'une base de connaissances avec un chaînage avant sur les règles de la base achevée et cela quelle que soit la base de faits ajoutée par la suite. De plus les bases inconsistantes sont aussitôt détectées car dans ce cas l'achèvement génère une base vide.

Sur les bases obtenues après ces opérations nous avons établi trois méthodes de réduction du résultat obtenu. L'une sur les règles, laissant un grand nombre de règles mais pouvant être utilisée par n'importe quel chaînage avant naïf , l'autre sur les clauses, avec beaucoup moins de place mémoire occupée si on utilise un chaînage avant sur les clauses, la troisième encore sur des clauses, très facilement implémentable et très rapide si la base est achevée par résolution car elle se base sur une propriété des résolvantes. Malheureusement cette dernière réduction n'est pas une opération de réduction totale en général bien qu'une très grande partie des clauses redondantes soient supprimées.

Enfin nous avons présenté des propriétés et des applications intéressantes de l'achèvement complet, notamment des réflexions sur la complexité en taille de la base obtenue qui laisse à penser que les littéraux conséquences bivaluées de la base initiale peuvent être obtenus en un temps polynomial, ce qui amène une extension de ce calcul au calcul des clauses conséquences lui-aussi obtenu en temps qui semble être polynomial.

L'utilisation de ces différentes techniques peut se résumer dans le schéma suivant:



En guise de recherches futures il serait intéressant de trouver d'autres mèthodes permettant d'achever des bases de connaissances encore plus rapidement. Nous nous intéressons de plus à la possibilité de trouver des algorithmes plus adaptés à l'achèvement des bases avec variables sans symboles de fonctions (ce qui est déjà possible) mais aussi à l'achèvement des bases avec

symboles de fonctions. Il serait intéressant de trouver d'autres possibilités d'achèvement relatif et d'achèvement partiel ce qui permettrait d'obtenir un compromis entre un calcul complet et un calcul rapide. Cela permettrait de plus d'établir une méthodologie de construction de bases de connaissances adaptée à l'achèvement. Enfin, nous étudions la possibilité d'intégrer la compilation logique dans les Prolog avec contraintes booléennes et notamment de l'adapter à la compilation d'un programme Prolog avec contraintes comme Prolog III [Col 87] ou Chip [Dinc 88].

V. Exemples.

Ce chapitre contient 8 exemples de bases qui rassemblent les principales difficultés rencontrées pour achever une base de règles. Ces bases sont achevées par résolution (méthode AR).

Chaque exemple n'est pas toujours le plus simple dans sa catégorie. Nous nous sommes efforcés de toujours citer le cas d'une conséquence non obtenue après l'ajout d'un littéral.

Pour chaque exemple, un court commentaire de la difficulté est précisé, puis l'ensemble de clauses saturé par résolution avec simplification est donné.

Toute clause précédée d'une '*' est une clause que l'on peut obtenir par résolution directe avec une intersection vide des littéraux de ses parents.

Toute clause précédée d'un '+' est une clause complètement enlevable, c'est-à-dire dont aucune de ses variantes n'est utile au chaînage avant.

Enfin la base de clauses réduite est donnée, ainsi que la base de règles après suppression des variantes inutiles.

Les trois types de simplification sont effectués en prenant les clauses (ou les règles) dans l'ordre inverse de leur création.

Des commentaires sont mis en italiques aux points importants.

A. Exemples formels.

Exemple 1.

Un littéral (B) et sa négation (¬B) dépendent du même littéral (A).

Si a ALORS b. De C on ne peut obtenir D.
Si a ALORS non(b).
Si non(a) ET c ALORS d.
------ 3 règles ------

L'achèvement par résolution donne les clauses

non(a)
d,non(c)

Aucune clause n'est enlevable par résolution directe.

Aucune clause n'est complètement enlevable. Il reste alors 2 clauses.

La réduction des variantes donne

non(a).
Si c ALORS d.
Si non(d) ALORS non(c).
------ 3 règles ------

Exemple 2.

Un littéral (B) dépend d'un littéral (A) et de sa négation (¬A).

Si a ALORS b. De C on ne peut obtenir D.
Si non(a) ALORS b.
Si b ET c ALORS d.
------ 3 règles -------

L'achèvement par résolution donne les clauses

b
d,non(c)

Aucune clause n'est enlevable par résolution directe.

Aucune clause n'est complètement enlevable. Il reste alors 2 clauses.

La réduction des variantes donne

b.
Si c ALORS d.
Si non(d) ALORS non(c).
------ 3 règles ------

Exemple 3.

Les variantes peuvent suffire dans certains cas.

Si a ALORS b. De A on ne peut obtenir D.
Si c ALORS d.
Si non(c) ALORS non(b).
------ 3 règles -------

L'achèvement par résolution donne les clauses

b,non(a)
d,non(c)
c,non(b)
* + c,non(a)
* + d,non(b)
* + d,non(a)

Après avoir enlevé les clauses obtenues par résolution directe, il reste 3 clauses.

Après avoir enlevé les clauses complètement enlevables, il reste les 3 clauses suivantes:

b,non(a)
d,non(c)
c,non(b)

La réduction des variantes donne

Si a ALORS b.
Si non(b) ALORS non(a).
Si c ALORS d.
Si non(d) ALORS non(c).
Si b ALORS c.
Si non(c) ALORS non(b).
------ 6 règles ------

Exemple 4.

Un 'et' simulé artificiellement. (Voir aussi l'exemple 6, très ressemblant). C'est un autre cas plus subtil où les variantes suffisent.

Si c ALORS non(d). De C on ne peut obtenir E.
Si a ALORS d.
Si b ALORS d.
Si non(a) ET non(b) ALORS e.
------ 4 règles ------

L'achèvement par résolution donne les clauses

non(c),non(d)
d,non(a)
d,non(b)
a,b,e
* + non(a),non(c)
* + non(b),non(c)
d,e
* + e,non(c)

Après avoir enlevé les clauses obtenues par résolution directe, il reste 5 clauses.

Après avoir enlevé les clauses complètement enlevables, il reste les 5 clauses suivantes:

non(c),non(d)
d,non(a)
d,non(b)
a,b,e
d,e

La réduction des variantes donne

Si d ALORS non(c).
Si c ALORS non(d).
Si a ALORS d.
Si non(d) ALORS non(a).
Si b ALORS d.
Si non(d) ALORS non(b).
Si non(b) ET non(e) ALORS a.
Si non(a) ET non(e) ALORS b.
Si non(a) ET non(b) ALORS e.
Si non(e) ALORS d. une des deux variantes a été supprimée.
—— 10 règles ——

Exemple 5.

La simplification d'une base 'mal écrite' est parfois nécessaire.

Si a ET c ALORS b. De A on ne peut obtenir B.
Si a ET non(c) ALORS b.
—— 2 règles ——

L'achèvement par résolution donne la clause
b,non(a)

Aucune clause n'est enlevable par résolution directe.

Aucune clause n'est complètement enlevable. Il reste alors 1 clause.

La réduction des variantes donne

Si a ALORS b.
Si non(b) ALORS non(a).
—— 2 règles ——

Exemple 6.

Une base hiérarchique stricte, même "sensée", n'est pas achevée à partir des faits de base. C'est un 'ou' simulé artificiellement mais contrairement à l'exemple 4 les variantes ne suffisent plus.

Si c ALORS non(d). De C on ne peut obtenir E.
Si a ET b ALORS d.
Si non(a) ALORS e.
Si non(b) ALORS e.
—— 4 règles ——

L'achèvement par résolution donne les clauses

non(c),non(d)
d,non(a),non(b)
a,e
b,e
* + non(a),non(b),non(c)
d,e
* + e,non(c)

Après avoir enlevé les clauses obtenues par résolution directe, il reste 5 clauses.

Après avoir enlevé les clauses complètement enlevables, il reste les 5 clauses suivantes:

non(c),non(d)
d,non(a),non(b)
a,e
b,e
d,e

La réduction des variantes donne

Si d ALORS non(c).
Si c ALORS non(d).
Si a ET b ALORS d.
Si non(d) ET b ALORS non(a).
Si non(d) ET a ALORS non(b).
Si non(e) ALORS a.
Si non(a) ALORS e.
Si non(e) ALORS b.

Si non(b) ALORS e.
Si non(d) ALORS e.
—— 10 règles ——

une des deux variantes a été supprimée.

Exemple 7.

Une base non Horn complétée au sens trivalué n'est pas en général achevée.

Si a ALORS b.
Si a ET b ALORS c.
Si d ALORS non(c).
Si non(a) ALORS non(b).
Si non(a) ALORS non(c).Complétion
Si non(b) ALORS non(c).
Si non(d) ALORS c.
—— 7 règles ——

Base initiale.

De A on ne peut obtenir ¬D

L'achèvement par résolution donne les clauses

```
        b,non(a)
        non(c),non(d)
        a,non(b)
        a,non(c)
* +     b,non(c)
        c,d
        c,non(a)
* +     c,non(b)
* +     a,d
* +     b,d
* +     non(a),non(d)
* +     non(b),non(d)
```

L'ordre dans lequel les clauses sont traitées est ici particulièrement important pour les différentes simplifications. Un autre ordre donnerait des résultats différents.

Après avoir enlevé les clauses obtenues par résolution directe, il reste 6 clauses.

Après avoir enlevé les clauses complètement enlevables, il reste les 6 clauses suivantes:

b,non(a)
non(c),non(d)
a,non(b)
a,non(c)
c,d
c,non(a)

La réduction des variantes donne

Si a ALORS b.
Si non(b) ALORS non(a).
Si c ALORS non(d).
Si d ALORS non(c).
Si b ALORS a.
Si non(a) ALORS non(b).
Si c ALORS a.
Si non(a) ALORS non(c).
Si non(c) ALORS d.
Si non(d) ALORS c.
Si a ALORS c.
Si non(c) ALORS non(a).
—— 12 règles ——

Exemple 8.

L'achèvement permet de simplifier une base mal écrite (ici une base bivaluée).

Si b ET d ET e ALORS f.
Si g ET d ALORS a.
Si c ET f ALORS a.
Si b ALORS x.
Si d ALORS e.
Si x ET a ALORS h.
Si c ALORS d.
Si x ET c ALORS a.
Si x ET b ALORS d.
—— 9 règles ——

L'achèvement par résolution donne les clauses

```
        a,non(d),non(g)
        a,non(c),non(f)
        x,non(b)
        e,non(d)
        h,non(a),non(x)
        d,non(c)
        a,non(c),non(x)
* +     h,non(d),non(g),non(x)
```

* + h,non(a),non(b)
* + a,non(c),non(g)
* + e,non(c)
* + a,non(b),non(c)
h,non(c),non(x)
d,non(b)
* + a,non(b),non(g)
* + e,non(b)
* + h,non(b),non(c)
f,non(b)
h,non(b),non(g)

Après avoir enlevé les clauses obtenues par résolution directe, il reste 11 clauses.

Après avoir enlevé les clauses complètement enlevables il reste les 11 clauses suivantes:

a,non(d),non(g)
a,non(c),non(f)
x,non(b)
e,non(d)
h,non(a),non(x)
d,non(c)
a,non(c),non(x)
h,non(c),non(x)
d,non(b)
f,non(b)
h,non(b),non(g)

La réduction des variantes donne

SI d ET g ALORS a.	SI d ET g ALORS a.
SI non(a) ET g ALORS non(d).	
SI non(a) ET d ALORS non(g).	
SI c ET f ALORS a.	SI c ET f ALORS a.
SI non(a) ET f ALORS non(c).	
SI non(a) ET c ALORS non(f).	
SI b ALORS x.	SI b ALORS x.
SI non(x) ALORS non(b).	
SI d ALORS e.	SI d ALORS e.
SI non(e) ALORS non(d).	
SI a ET x ALORS h.	SI a ET x ALORS h.
SI non(h) ET x ALORS non(a).	
SI non(h) ET a ALORS non(x).	
SI c ALORS d.	SI c ALORS d.
SI non(d) ALORS non(c).	
SI c ET x ALORS a.	SI c ET x ALORS a.
SI non(a) ET x ALORS non(c).	
SI non(a) ET c ALORS non(x).	
SI non(h) ET c ALORS non(x).	
SI b ALORS d.	SI b ALORS d.
SI non(d) ALORS non(b).	
SI b ALORS f.	SI b ALORS f.
SI non(f) ALORS non(b).	
SI non(h) ET g ALORS non(b).	
------ 24 règles -------	dont 9 positives !! (si on garde la structure bivaluée)

B. Les logiciens.

Pour des raisons de place, nous ne pouvions pas montrer le traitement de beaucoup de bases de connaissances. Il nous fallait des bases de connaissances avec négations en prémisses et en conclusions et nécessitant de nombreuses bases de faits. Nous avons appris qu'EDF utilise régulièrement ce genre de bases notamment pour la surveillance des centrales atomiques, mais malheureusement ces bases sont bien évidemment confidentielles et nous n'avons pu les obtenir. Notre choix s'est donc porté sur trois exemples classiques, relativement importants et représentatifs des problèmes liés à l'incomplétude du chaînage avant. Pour chaque exemple nous présentons le problème sous sa forme "Française" puis nous le formalisons sous forme de clauses et enfin nous donnons l'ensemble de règles achevé associé car il est sans doute plus facile à lire que la base de clauses associée.

Le premier de ces exemples est un problème décrit par lewis Caroll dans [car 66] sous le titre

"Le logicien".

Un logicien qui dîne de deux côtelettes de porc risque de se ruiner.

Un joueur dont l'appétit n'est pas féroce risque de se ruiner.

Un homme déprimé parce qu'il s'est ruiné et qu'il risque de se ruiner à nouveau se lève toujours à cinq heures du matin.

Un homme qui ne joue pas et qui ne dîne pas de deux côtelettes de porc est assuré d'avoir un appétit féroce.

Un homme dynamique qui se couche avant quatre heures du matin aurait intérêt à être conducteur de fiacre.

Un homme doué d'un appétit féroce, qui ne s'est pas ruiné et qui ne se lève pas à cinq heures du matin, dîne toujours de deux côtelettes de porc.

Un logicien qui risque de se ruiner aurait intérêt à devenir conducteur de fiacre.

Un joueur convaincu qui est déprimé bien qu'il ne soit pas ruiné, ne court aucun risque de se ruiner.

Un homme qui ne joue pas et dont l'appétit n'est pas féroce est toujours dynamique.

Un logicien dynamique qui est véritablement convaincu ne risque pas de se ruiner.

Un homme doué d'appétit féroce n'a nul besoin de devenir conducteur de fiacre s'il est vraiment convaincu.

Un Joueur qui est déprimé bien qu'il ne risque pas de se ruiner, reste debout jusqu'à quatre heures du matin.

Un homme qui s'est ruiné et qui ne dîne pas de deux côtelettes de porc, aurait intérêt à devenir conducteur de fiacre à moins qu'il ne se lève à cinq heures du matin.

Un joueur qui se couche avant quatre heures du matin n'a nul besoin de devenir conducteur de fiacre à moins qu'il ne soit doué d'un appétit féroce.

Un homme doué d'appétit féroce et qui est déprimé bien qu'il ne risque nullement de se ruiner est un joueur.

Ce qui se transforme en 15 clauses [OR 89] de la façon suivante:

→ non logicien OU non dine_de_deux_cotelettes OU risque_de_se_ruiner.

→ non joueur OU doue_d_appetit_feroce OU risque_de_se_ruiner .

→ dynamique OU non s_est_ruine OU non risque_de_se_ruiner OU se_leve_a_cinq_heures.

→ joueur OU dine_de_deux_cotelettes OU doue_d_appetit_feroce .

→ non dynamique OU ne_se_couche_pas_avant_quatre_heures OU devrait_etre_conducteur_de_fiacre .

→ non doue_d_appetit_feroce OU s_est_ruine OU se_leve_a_cinq_heures OU dine_de_deux_cotelettes .

→ non logicien OU non risque_de_se_ruiner OU devrait_etre_conducteur_de_fiacre .

→ non joueur OU non convaincu OU dynamique OU s_est_ruine OU non risque_de_se_ruiner .

→ joueur OU doue_d_appetit_feroce OU dynamique .

→ non logicien OU non dynamique OU non convaincu OU non risque_de_se_ruiner .

→ non doue_d_appetit_feroce OU non convaincu OU non devrait_etre_conducteur_de_fiacre .

→ non joueur OU dynamique OU risque_de_se_ruiner OU ne_se_couche_pas_avant_quatre_heures .

→ non s_est_ruine OU dine_de_deux_cotelettes OU se_leve_a_cinq_heures OU devrait_etre_conducteur_de_fiacre .

→ non joueur OU ne_se_couche_pas_avant_quatre_heures OU doue_d_appetit_feroce OU non devrait_etre_conducteur_de_fiacre .

→ non doue_d_appetit_feroce OU dynamique OU risque_de_se_ruiner OU joueur .

La mise sous forme de règles de cette base se fait (par calcul des variantes) en 54 règles et 11 propositions.

De nombreuses déductions sont impossibles à faire sur la base initiale avec un chaînage avant même après calcul de toutes les variantes. Par exemple:

De **{logicien, convaincu}** on ne peut déduire **se_lève_à_cinq_heures** ou encore **ne_se_couche_pas_avant_quatre_heures.**

De **{joueur, dynamique, convaincu}** on ne peut déduire **ne_se_couche_pas_avant_quatre_heures.**

De **{logicien, non doue_d_appetit_feroce}** on ne peut déduire **risque_de_se_ruiner** ou

devrait_etre_conducteur_de_fiacre.

De **{non joueur, non dynamique, non doue_d_appetit_feroce}** on ne peut déduire **risque_de_se_ruiner.**

De **{non joueur, convaincu, non doue_d_appetit_feroce}** on ne peut déduire **non logicien.**

La base de clauses obtenue après achèvement et simplifications possède 69 clauses et la base de règles obtenue contient les 120 règles que voici:

SI dine_de_deux_cotelettes ET non(risque_de_se_ruiner) ALORS non(logicien).
SI logicien ET non(risque_de_se_ruiner) ALORS non(dine_de_deux_cotelettes).
SI logicien ET dine_de_deux_cotelettes ALORS risque_de_se_ruiner.
SI joueur ET non(doue_d_appetit_feroce) ALORS risque_de_se_ruiner.
SI non(risque_de_se_ruiner) ET non(doue_d_appetit_feroce) ALORS non(joueur).
SI non(risque_de_se_ruiner) ET joueur ALORS doue_d_appetit_feroce.
SI non(dynamique) ET s_est_ruine ET non(se_leve_a_cinq_heures) ALORS non(risque_de_se_ruiner).
SI risque_de_se_ruiner ET s_est_ruine ET non(se_leve_a_cinq_heures) ALORS dynamique.
SI risque_de_se_ruiner ET non(dynamique) ET non(se_leve_a_cinq_heures) ALORS non(s_est_ruine).
SI risque_de_se_ruiner ET non(dynamique) ET s_est_ruine ALORS se_leve_a_cinq_heures.
SI non(joueur) ET non(doue_d_appetit_feroce) ALORS dine_de_deux_cotelettes.
SI non(dine_de_deux_cotelettes) ET non(doue_d_appetit_feroce) ALORS joueur.
SI non(dine_de_deux_cotelettes) ET non(joueur) ALORS doue_d_appetit_feroce.
SI non(ne_se_couche_pas_avant_quatre_heures) ET non(devrait_etre_conducteur_de_fiacre) ALORS non(dynamique).
SI dynamique ET non(devrait_etre_conducteur_de_fiacre) ALORS ne_se_couche_pas_avant_quatre_heures.
SI dynamique ET non(ne_se_couche_pas_avant_quatre_heures) ALORS devrait_etre_conducteur_de_fiacre.
SI doue_d_appetit_feroce ET non(s_est_ruine) ET non(se_leve_a_cinq_heures) ALORS dine_de_deux_cotelettes.
SI non(dine_de_deux_cotelettes) ET non(s_est_ruine) ET non(se_leve_a_cinq_heures) ALORS non(doue_d_appetit_feroce).
SI non(dine_de_deux_cotelettes) ET doue_d_appetit_feroce ET non(se_leve_a_cinq_heures) ALORS s_est_ruine.
SI non(dine_de_deux_cotelettes) ET doue_d_appetit_feroce ET non(s_est_ruine) ALORS se_leve_a_cinq_heures.
SI risque_de_se_ruiner ET non(devrait_etre_conducteur_de_fiacre) ALORS non(logicien).
SI logicien ET non(devrait_etre_conducteur_de_fiacre) ALORS non(risque_de_se_ruiner).
SI logicien ET risque_de_se_ruiner ALORS devrait_etre_conducteur_de_fiacre.
SI joueur ET non(dynamique) ET non(s_est_ruine) ET convaincu ALORS non(risque_de_se_ruiner).
SI risque_de_se_ruiner ET non(dynamique) ET non(s_est_ruine) ET convaincu ALORS non(joueur).
SI risque_de_se_ruiner ET joueur ET non(s_est_ruine) ET convaincu ALORS dynamique.
SI risque_de_se_ruiner ET joueur ET non(dynamique) ET convaincu ALORS s_est_ruine.
SI risque_de_se_ruiner ET joueur ET non(dynamique) ET non(s_est_ruine) ALORS non(convaincu).
SI non(doue_d_appetit_feroce) ET non(dynamique) ALORS joueur.
SI non(joueur) ET non(dynamique) ALORS doue_d_appetit_feroce.
SI non(joueur) ET non(doue_d_appetit_feroce) ALORS dynamique.
SI risque_de_se_ruiner ET dynamique ET convaincu ALORS non(logicien).
SI logicien ET dynamique ET convaincu ALORS non(risque_de_se_ruiner).
SI logicien ET risque_de_se_ruiner ET convaincu ALORS non(dynamique).
SI devrait_etre_conducteur_de_fiacre ET convaincu ALORS non(doue_d_appetit_feroce).
SI doue_d_appetit_feroce ET convaincu ALORS non(devrait_etre_conducteur_de_fiacre).
SI doue_d_appetit_feroce ET devrait_etre_conducteur_de_fiacre ALORS non(convaincu).
SI s_est_ruine ET non(se_leve_a_cinq_heures) ET non(devrait_etre_conducteur_de_fiacre) ALORS dine_de_deux_cotelettes.
SI non(dine_de_deux_cotelettes) ET non(se_leve_a_cinq_heures) ET non(devrait_etre_conducteur_de_fiacre) ALORS non(s_est_ruine).
SI non(dine_de_deux_cotelettes) ET s_est_ruine ET non(devrait_etre_conducteur_de_fiacre) ALORS se_leve_a_cinq_heures.

SI non(dine_de_deux_cotelettes) ET s_est_ruine ET non(se_leve_a_cinq_heures) ALORS devrait_etre_conducteur_de_fiacre.

SI non(doue_d_appetit_feroce) ET non(ne_se_couche_pas_avant_quatre_heures) ET devrait_etre_conducteur_de_fiacre ALORS non(joueur).

SI joueur ET non(ne_se_couche_pas_avant_quatre_heures) ET devrait_etre_conducteur_de_fiacre ALORS doue_d_appetit_feroce.

SI joueur ET non(doue_d_appetit_feroce) ET devrait_etre_conducteur_de_fiacre ALORS ne_se_couche_pas_avant_quatre_heures.

SI joueur ET non(doue_d_appetit_feroce) ET non(ne_se_couche_pas_avant_quatre_heures) ALORS non(devrait_etre_conducteur_de_fiacre).

SI dine_de_deux_cotelettes ET non(devrait_etre_conducteur_de_fiacre) ALORS non(logicien).

SI dine_de_deux_cotelettes ET dynamique ET convaincu ALORS non(logicien).

SI non(dine_de_deux_cotelettes) ET non(risque_de_se_ruiner) ALORS doue_d_appetit_feroce.

SI non(dine_de_deux_cotelettes) ET risque_de_se_ruiner ET doue_d_appetit_feroce ET non(dynamique) ALORS se_leve_a_cinq_heures.

SI joueur ET non(dynamique) ET non(se_leve_a_cinq_heures) ET convaincu ALORS non(risque_de_se_ruiner).

SI risque_de_se_ruiner ET joueur ET non(se_leve_a_cinq_heures) ET convaincu ALORS dynamique.

SI non(joueur) ET non(s_est_ruine) ET non(se_leve_a_cinq_heures) ALORS dine_de_deux_cotelettes.

SI non(dine_de_deux_cotelettes) ET non(ne_se_couche_pas_avant_quatre_heures) ET devrait_etre_conducteur_de_fiacre ALORS doue_d_appetit_feroce.

SI joueur ET non(doue_d_appetit_feroce) ET dynamique ALORS ne_se_couche_pas_avant_quatre_heures.

SI doue_d_appetit_feroce ET non(se_leve_a_cinq_heures) ET non(devrait_etre_conducteur_de_fiacre) ALORS dine_de_deux_cotelettes.

SI non(dine_de_deux_cotelettes) ET doue_d_appetit_feroce ET non(devrait_etre_conducteur_de_fiacre) ALORS se_leve_a_cinq_heures.

SI non(dynamique) ET non(ne_se_couche_pas_avant_quatre_heures) ET devrait_etre_conducteur_de_fiacre ALORS doue_d_appetit_feroce.

SI non(joueur) ET non(dynamique) ALORS risque_de_se_ruiner.

SI non(risque_de_se_ruiner) ET non(dynamique) ALORS joueur.

SI non(risque_de_se_ruiner) ET non(joueur) ALORS dynamique.

SI joueur ET non(ne_se_couche_pas_avant_quatre_heures) ET convaincu ALORS non(devrait_etre_conducteur_de_fiacre).

SI logicien ET non(doue_d_appetit_feroce) ALORS risque_de_se_ruiner.

SI non(doue_d_appetit_feroce) ET s_est_ruine ET non(se_leve_a_cinq_heures) ALORS dynamique.

SI non(joueur) ET s_est_ruine ET non(se_leve_a_cinq_heures) ALORS dynamique.

SI non(joueur) ET non(se_leve_a_cinq_heures) ET non(devrait_etre_conducteur_de_fiacre) ALORS dine_de_deux_cotelettes.

SI joueur ET dynamique ET convaincu ALORS ne_se_couche_pas_avant_quatre_heures.

SI non(risque_de_se_ruiner) ET non(s_est_ruine) ET non(se_leve_a_cinq_heures) ALORS dine_de_deux_cotelettes.

SI non(s_est_ruine) ET non(se_leve_a_cinq_heures) ET non(ne_se_couche_pas_avant_quatre_heures) ET devrait_etre_conducteur_de_fiacre ALORS dine_de_deux_cotelettes.

SI non(dine_de_deux_cotelettes) ET non(se_leve_a_cinq_heures) ET convaincu ALORS non(doue_d_appetit_feroce).

SI non(dynamique) ET non(s_est_ruine) ET convaincu ALORS doue_d_appetit_feroce.

SI non(doue_d_appetit_feroce) ET non(s_est_ruine) ET convaincu ALORS dynamique.

SI non(dine_de_deux_cotelettes) ET non(ne_se_couche_pas_avant_quatre_heures) ET convaincu ALORS non(devrait_etre_conducteur_de_fiacre).

SI non(dynamique) ET non(ne_se_couche_pas_avant_quatre_heures) ET convaincu ALORS non(devrait_etre_conducteur_de_fiacre).

SI non(dynamique) ET non(ne_se_couche_pas_avant_quatre_heures) ALORS risque_de_se_ruiner.

SI non(risque_de_se_ruiner) ET non(ne_se_couche_pas_avant_quatre_heures) ALORS dynamique.

SI non(risque_de_se_ruiner) ET non(dynamique) ALORS ne_se_couche_pas_avant_quatre_heures.

SI logicien ET non(joueur) ET convaincu ALORS doue_d_appetit_feroce.

SI non(risque_de_se_ruiner) ET non(se_leve_a_cinq_heures) ET non(devrait_etre_conducteur_de_fiacre) ALORS dine_de_deux_cotelettes.

SI risque_de_se_ruiner ET non(joueur) ET convaincu ALORS non(logicien).

SI logicien ET non(s_est_ruine) ET non(se_leve_a_cinq_heures) ALORS risque_de_se_ruiner.

SI dine_de_deux_cotelettes ET non(joueur) ET convaincu ALORS non(logicien).

SI s_est_ruine ET non(se_leve_a_cinq_heures) ET non(ne_se_couche_pas_avant_quatre_heures) ALORS dynamique.

SI non(dine_de_deux_cotelettes) ET non(dynamique) ET non(se_leve_a_cinq_heures) ALORS joueur.

SI non(joueur) ET non(se_leve_a_cinq_heures) ET convaincu ALORS dine_de_deux_cotelettes.

SI non(dine_de_deux_cotelettes) ET dynamique ET convaincu ALORS ne_se_couche_pas_avant_quatre_heures.

SI non(risque_de_se_ruiner) ET non(devrait_etre_conducteur_de_fiacre) ALORS ne_se_couche_pas_avant_quatre_heures.

SI non(doue_d_appetit_feroce) ET non(devrait_etre_conducteur_de_fiacre) ALORS non(logicien).

SI non(dynamique) ET non(se_leve_a_cinq_heures) ET convaincu ALORS doue_d_appetit_feroce.

SI non(doue_d_appetit_feroce) ET non(se_leve_a_cinq_heures) ET convaincu ALORS dynamique.

SI logicien ET non(dynamique) ET convaincu ALORS joueur.

SI non(doue_d_appetit_feroce) ET dynamique ET convaincu ALORS non(logicien).

SI risque_de_se_ruiner ET non(s_est_ruine) ET convaincu ALORS non(logicien).

SI logicien ET non(s_est_ruine) ET convaincu ALORS non(risque_de_se_ruiner).

SI non(dynamique) ET non(s_est_ruine) ET devrait_etre_conducteur_de_fiacre ALORS non(convaincu).

SI logicien ET non(joueur) ET devrait_etre_conducteur_de_fiacre ALORS non(convaincu).

SI non(risque_de_se_ruiner) ET non(se_leve_a_cinq_heures) ET convaincu ALORS dine_de_deux_cotelettes.

SI non(dine_de_deux_cotelettes) ET non(dynamique) ET non(se_leve_a_cinq_heures) ET devrait_etre_conducteur_de_fiacre ALORS ne_se_couche_pas_avant_quatre_heures.

SI non(s_est_ruine) ET non(ne_se_couche_pas_avant_quatre_heures) ET convaincu ALORS non(joueur).

SI joueur ET non(s_est_ruine) ET convaincu ALORS ne_se_couche_pas_avant_quatre_heures.

SI joueur ET non(s_est_ruine) ET non(ne_se_couche_pas_avant_quatre_heures) ALORS non(convaincu).

SI non(dynamique) ET non(se_leve_a_cinq_heures) ET devrait_etre_conducteur_de_fiacre ALORS non(convaincu).

SI non(se_leve_a_cinq_heures) ET non(ne_se_couche_pas_avant_quatre_heures) ET convaincu ALORS non(joueur).

SI joueur ET non(se_leve_a_cinq_heures) ET convaincu ALORS ne_se_couche_pas_avant_quatre_heures.

SI joueur ET non(se_leve_a_cinq_heures) ET non(ne_se_couche_pas_avant_quatre_heures) ALORS non(convaincu).

SI logicien ET dynamique ET devrait_etre_conducteur_de_fiacre ALORS non(convaincu).

SI dine_de_deux_cotelettes ET non(s_est_ruine) ET convaincu ALORS non(logicien).

SI s_est_ruine ET non(se_leve_a_cinq_heures) ET non(devrait_etre_conducteur_de_fiacre) ALORS ne_se_couche_pas_avant_quatre_heures.

SI logicien ET non(ne_se_couche_pas_avant_quatre_heures) ALORS devrait_etre_conducteur_de_fiacre.

SI non(se_leve_a_cinq_heures) ET non(devrait_etre_conducteur_de_fiacre) ALORS non(logicien).

SI logicien ET non(se_leve_a_cinq_heures) ALORS devrait_etre_conducteur_de_fiacre.

SI logicien ET non(s_est_ruine) ET devrait_etre_conducteur_de_fiacre ALORS non(convaincu).

SI non(dine_de_deux_cotelettes) ET non(dynamique) ET convaincu ALORS se_leve_a_cinq_heures.

SI non(dine_de_deux_cotelettes) ET non(dynamique) ET non(se_leve_a_cinq_heures) ALORS non(convaincu).

SI non(dine_de_deux_cotelettes) ET non(se_leve_a_cinq_heures) ET non(ne_se_couche_pas_avant_quatre_heures) ALORS non(convaincu).

SI non(ne_se_couche_pas_avant_quatre_heures) ET convaincu ALORS non(logicien).

SI logicien ET convaincu ALORS ne_se_couche_pas_avant_quatre_heures.

SI logicien ET non(ne_se_couche_pas_avant_quatre_heures) ALORS non(convaincu).

SI non(se_leve_a_cinq_heures) ET convaincu ALORS non(logicien).

SI logicien ET convaincu ALORS se_leve_a_cinq_heures.

SI logicien ET non(se_leve_a_cinq_heures) ALORS non(convaincu).

Maintenant, grâce à la base compilée, nous vérifions que l'on peut obtenir par exemple **ne_se_couche_pas_avant_quatre_heures** de la base de faits **{joueur, dynamique, convaincu}** à l'aide d'un simple chaînage avant. On peut aussi vérifier que la clause **non(logicien) OU non(doue_d_appetit_feroce) OU risque_de_se_ruiner** est bien conséquence logique de la base par chaînage avant. Il suffit pour cela d'ajouter **{logicien, doue_d_appetit_feroce}** et de vérifier que l'on obtient bien **risque_de_se_ruiner**.

C. Les députés.

Le deuxième exemple traité est, comme le précédent, un exemple tiré de Lewis Caroll [Car 66]. Il est présenté sous le titre "Les députés".

Tout individu apte à être député et qui ne passe pas son temps à faire des discours, est un bienfaiteur du peuple.

Les gens à l'esprit clair et qui s'expriment bien ont recu une éducation convenable.

Une femme digne d'éloges est une femme qui sait garder un secret.

Les gens qui rendent des services au peuple mais qui n'emploient pas leur influence à des fins méritoires ne sont pas aptes à être députés.

Les gens qui valent leur pesant d'or et qui sont dignes d'éloges sont toujours sans prétention.

Les bienfaiteurs du peuple qui emploient leur influence à des fins méritoires sont dignes d'éloges.

Les gens qui sont impopulaires et qui ne valent pas leur pesant d'or ne savent pas garder un secret.

Les gens qui savent parler pendant des heures et des heures et qui sont aptes à être députés sont dignes d'éloges.

Tout individu qui sait garder un secert et qui est sans prétention est un bienfaiteur du peuple dont le souvenir demeurera impérissable.

Une femme qui rend des services au peuple est toujours populaire.

Les gens qui valent leur pesant d'or, qui ne cessent de discourir et dont le souvenir demeure impérissable sont précisément les gens dont on voit la photographie dans toutes les vitrines.

Une femme, si elle n'a pas l'esprit clair et qu'elle n'a pas recu une bonne éducation n'est pas apte à être député.

Tout individu qui sait garder un secret et qui sait ne pas discourir sans cesse peut être certain d'être impopulaire.

Un individu qui a l'esprit clair, qui a de l'influence et qui emploie son influence à des fins méritoires est un bienfaiteur du peuple.

Un bienfaiteur du peuple sans prétention n'est pas le genre de personne dont la photographie est affichée dans toutes les vitrines.

Les gens qui savent garder un secret et qui emploient leur influence à des fins méritoires valent leur pesant d'or.

Une personne qui ne sait pas s'exprimer et qui ne possède pas d'influence n'est surement pas une femme.

Les gens qui sont populaires et dignes d'éloges sont, soit des bienfaiteurs du peuple soit des gens sans prétention.

Celui qui emploie son influence à des fins méritoires possède évidemment une influence.

Ce qui se transforme en 20 clauses [OR 89] de la façon suivante:

→ non(apte_a_etre_depute) OU discours_sans_cesse OU bienfaiteur_du_peuple.

→ non(esprit_clair) OU non(s_exprimant_bien) OU bien_eduque.

→ non(femme) OU non(digne_d_eloges) OU gardant_secret.

→ non(bienfaiteur_du_peuple) OU emploie_influence_a_des_fins_meritoires OU non(apte_a_etre_depute).

→ non(valant_son_pesant_d_or) OU non(digne_d_eloges) OU sans_pretention.

→ non(bienfaiteur_du_peuple) OU non(emploie_influence_a_des_fins_meritoires) OU digne_d_eloges.

→ populaire OU valant_son_pesant_d_or OU non(gardant_secret).

→ non(discours_sans_cesse) OU non(apte_a_etre_depute) OU digne_d_eloges.

→ non(gardant_secret) OU non(sans_pretention) OU bienfaiteur_du_peuple.

→ non(gardant_secret) OU non(sans_pretention) OU souvenir_demeurant_imperissable.

→ non(femme) OU non(bienfaiteur_du_peuple) OU populaire.

→ non(valant_son_pesant_d_or) OU non(discours_sans_cesse) OU non(souvenir_demeurant_imperissable) OU affiche_dans_vitrines.

→ non(femme) OU esprit_clair OU bien_eduque OU non(apte_a_etre_depute).

→ non(gardant_secret) OU discours_sans-cesse OU non(populaire).

→ non(esprit-clair) OU non(emploie-influence-a-des-fins-meritoires) OU bienfaiteur-du-peuple.

→ non(bienfaiteur-du-peuple) OU non(sans-pretention) OU non(affiche-dans-vitrines).

→ non(gardant-secret) OU non(emploie-influence-a-des-fins-meritoires) OU valant-son-pesant-d-or.

→ s-exprimant-bien OU possede-une-influence OU non(femme).

→ non(populaire) OU non(digne-d-eloges) OU bienfaiteur-du-peuple OU sans-pretention.

→ non(emploie-influence-a-des-fins-meritoires) OU possede-une-influence.

La mise sous forme de règles de cette base se fait (par calcul des variantes) en 62 règles et 16 propositions.

De **{non digne_d_eloges}** on ne peut déduire **non apte_a_etre_depute.**

De **{digne_d_eloges, non populaire}** on ne peut déduire **non femme.**

De **{gardant_secret, non sans_pretention}** on ne peut déduire **non apte_a_etre_depute.**

De **{gardant_secret, populaire}** on ne peut déduire **discours_sans_cesse** ou **non apte_a_etre_depute.**

De **{gardant_secret, digne_d_eloges}** on ne peut déduire **bienfaiteur_du_peuple.**

De **{femme}** on ne peut déduire **non apte_a_etre_depute.**

La base de clauses obtenue après achèvement et simplifications possède 66 clauses et la base de règles obtenue contient les 113 règles que voici:

SI non(discours_sans_cesse) ET non(bienfaiteur_du_peuple) ALORS non(apte_a_etre_depute).
SI apte_a_etre_depute ET non(bienfaiteur_du_peuple) ALORS discours_sans_cesse.
SI apte_a_etre_depute ET non(discours_sans_cesse) ALORS bienfaiteur_du_peuple.
SI s_exprimant_bien ET non(bien_eduque) ALORS non(esprit_clair).
SI esprit_clair ET non(bien_eduque) ALORS non(s_exprimant_bien).
SI esprit_clair ET s_exprimant_bien ALORS bien_eduque.
SI digne_d_eloges ET non(gardant_secret) ALORS non(femme).
SI femme ET non(gardant_secret) ALORS non(digne_d_eloges).
SI femme ET digne_d_eloges ALORS gardant_secret.
SI bienfaiteur_du_peuple ET non(emploie_influence_a_des_fins_meritoires) ALORS non(apte_a_etre_depute).
SI apte_a_etre_depute ET non(emploie_influence_a_des_fins_meritoires) ALORS non(bienfaiteur_du_peuple).
SI apte_a_etre_depute ET bienfaiteur_du_peuple ALORS emploie_influence_a_des_fins_meritoires.
SI valant_son_pesant_d_or ET non(sans_pretention) ALORS non(digne_d_eloges).
SI digne_d_eloges ET non(sans_pretention) ALORS non(valant_son_pesant_d_or).
SI digne_d_eloges ET valant_son_pesant_d_or ALORS sans_pretention.
SI non(digne_d_eloges) ET emploie_influence_a_des_fins_meritoires ALORS non(bienfaiteur_du_peuple).
SI bienfaiteur_du_peuple ET emploie_influence_a_des_fins_meritoires ALORS digne_d_eloges.
SI bienfaiteur_du_peuple ET non(digne_d_eloges) ALORS non(emploie_influence_a_des_fins_meritoires).
SI non(valant_son_pesant_d_or) ET non(populaire) ALORS non(gardant_secret).
SI gardant_secret ET non(populaire) ALORS valant_son_pesant_d_or.
SI gardant_secret ET non(valant_son_pesant_d_or) ALORS populaire.
SI gardant_secret ET sans_pretention ALORS bienfaiteur_du_peuple.
SI non(bienfaiteur_du_peuple) ET sans_pretention ALORS non(gardant_secret).
SI non(bienfaiteur_du_peuple) ET gardant_secret ALORS non(sans_pretention).
SI sans_pretention ET non(souvenir_demeurant_imperissable) ALORS non(gardant_secret).
SI gardant_secret ET non(souvenir_demeurant_imperissable) ALORS non(sans_pretention).
SI gardant_secret ET sans_pretention ALORS souvenir_demeurant_imperissable.
SI femme ET non(populaire) ALORS non(bienfaiteur_du_peuple).
SI bienfaiteur_du_peuple ET non(populaire) ALORS non(femme).
SI bienfaiteur_du_peuple ET femme ALORS populaire.
SI valant_son_pesant_d_or ET souvenir_demeurant_imperissable ET non(affiche_dans_vitrines) ALORS non(discours_sans_cesse).
SI discours_sans_cesse ET souvenir_demeurant_imperissable ET non(affiche_dans_vitrines) ALORS non(valant_son_pesant_d_or).
SI discours_sans_cesse ET valant_son_pesant_d_or ET non(affiche_dans_vitrines) ALORS non(souvenir_demeurant_imperissable).
SI discours_sans_cesse ET valant_son_pesant_d_or ET souvenir_demeurant_imperissable ALORS affiche_dans_vitrines.
SI gardant_secret ET populaire ALORS discours_sans_cesse.
SI non(discours_sans_cesse) ET populaire ALORS non(gardant_secret).

SI non(discours_sans_cesse) ET gardant_secret ALORS non(populaire).

SI esprit_clair ET emploie_influence_a_des_fins_meritoires ALORS bienfaiteur_du_peuple.

SI non(bienfaiteur_du_peuple) ET emploie_influence_a_des_fins_meritoires ALORS non(esprit_clair).

SI non(bienfaiteur_du_peuple) ET esprit_clair ALORS non(emploie_influence_a_des_fins_meritoires).

SI sans_pretention ET affiche_dans_vitrines ALORS non(bienfaiteur_du_peuple).

SI bienfaiteur_du_peuple ET affiche_dans_vitrines ALORS non(sans_pretention).

SI bienfaiteur_du_peuple ET sans_pretention ALORS non(affiche_dans_vitrines).

SI emploie_influence_a_des_fins_meritoires ET non(valant_son_pesant_d_or) ALORS non(gardant_secret).

SI gardant_secret ET non(valant_son_pesant_d_or) ALORS non(emploie_influence_a_des_fins_meritoires).

SI gardant_secret ET emploie_influence_a_des_fins_meritoires ALORS valant_son_pesant_d_or.

SI femme ET non(possede_une_influence) ALORS s_exprimant_bien.

SI non(s_exprimant_bien) ET non(possede_une_influence) ALORS non(femme).

SI non(s_exprimant_bien) ET femme ALORS possede_une_influence.

SI digne_d_eloges ET non(sans_pretention) ET populaire ALORS bienfaiteur_du_peuple.

SI non(bienfaiteur_du_peuple) ET non(sans_pretention) ET populaire ALORS non(digne_d_eloges).

SI non(bienfaiteur_du_peuple) ET digne_d_eloges ET populaire ALORS sans_pretention.

SI non(bienfaiteur_du_peuple) ET digne_d_eloges ET non(sans_pretention) ALORS non(populaire).

SI non(possede_une_influence) ALORS non(emploie_influence_a_des_fins_meritoires).

SI emploie_influence_a_des_fins_meritoires ALORS possede_une_influence.

SI non(discours_sans_cesse) ET non(emploie_influence_a_des_fins_meritoires) ALORS non(apte_a_etre_depute).

SI esprit_clair ET non(digne_d_eloges) ALORS non(emploie_influence_a_des_fins_meritoires).

SI non(discours_sans_cesse) ET non(valant_son_pesant_d_or) ALORS non(gardant_secret).

SI gardant_secret ET affiche_dans_vitrines ALORS non(sans_pretention).

SI discours_sans_cesse ET bienfaiteur_du_peuple ET digne_d_eloges ET souvenir_demeurant_imperissable ALORS non(valant_son_pesant_d_or).

SI emploie_influence_a_des_fins_meritoires ET valant_son_pesant_d_or ET affiche_dans_vitrines ALORS non(bienfaiteur_du_peuple).

SI bienfaiteur_du_peuple ET gardant_secret ET non(sans_pretention) ALORS non(emploie_influence_a_des_fins_meritoires).

SI digne_d_eloges ET non(populaire) ET non(souvenir_demeurant_imperissable) ALORS non(gardant_secret).

SI non(discours_sans_cesse) ET femme ET sans_pretention ALORS non(gardant_secret).

SI digne_d_eloges ET emploie_influence_a_des_fins_meritoires ET non(souvenir_demeurant_imperissable) ALORS non(gardant_secret).

SI emploie_influence_a_des_fins_meritoires ET populaire ET souvenir_demeurant_imperissable ET non(affiche_dans_vitrines) ALORS non(gardant_secret).

SI non(digne_d_eloges) ALORS non(apte_a_etre_depute).

SI apte_a_etre_depute ALORS digne_d_eloges.

SI digne_d_eloges ET non(populaire) ET affiche_dans_vitrines ALORS non(gardant_secret).

SI non(discours_sans_cesse) ET digne_d_eloges ET non(souvenir_demeurant_imperissable) ALORS non(gardant_secret).

SI esprit_clair ET gardant_secret ET non(sans_pretention) ALORS non(emploie_influence_a_des_fins_meritoires).

SI digne_d_eloges ET emploie_influence_a_des_fins_meritoires ET affiche_dans_vitrines ALORS non(gardant_secret).

SI discours_sans_cesse ET valant_son_pesant_d_or ET sans_pretention ALORS non(gardant_secret).

SI discours_sans_cesse ET gardant_secret ET valant_son_pesant_d_or ALORS non(sans_pretention).

SI non(discours_sans_cesse) ET valant_son_pesant_d_or ET affiche_dans_vitrines ALORS non(apte_a_etre_depute).

SI digne_d_eloges ET non(populaire) ALORS non(femme).

SI non(emploie_influence_a_des_fins_meritoires) ET non(sans_pretention) ET populaire ALORS non(apte_a_etre_depute).

SI discours_sans_cesse ET digne_d_eloges ET gardant_secret ALORS non(valant_son_pesant_d_or).

SI discours_sans_cesse ET sans_pretention ET non(populaire) ALORS non(gardant_secret).

SI digne_d_eloges ET gardant_secret ALORS bienfaiteur_du_peuple.

SI non(bienfaiteur_du_peuple) ET gardant_secret ALORS non(digne_d_eloges).

SI non(bienfaiteur_du_peuple) ET digne_d_eloges ALORS non(gardant_secret).

SI valant_son_pesant_d_or ET sans_pretention ET populaire ALORS non(gardant_secret).

SI discours_sans_cesse ET emploie_influence_a_des_fins_meritoires ET sans_pretention ALORS non(gardant_secret).

SI non(discours_sans_cesse) ET digne_d_eloges ET affiche_dans_vitrines ALORS non(gardant_secret).

SI esprit_clair ET valant_son_pesant_d_or ET affiche_dans_vitrines ALORS non(emploie_influence_a_des_fins_meritoires).

SI femme ET valant_son_pesant_d_or ET sans_pretention ALORS non(gardant_secret).

Si femme ET gardant_secret ET valant_son_pesant_d_or ALORS non(sans_pretention).
Si non(bienfaiteur_du_peuple) ET femme ALORS non(digne_d_eloges).
Si non(discours_sans_cesse) ET digne_d_eloges ALORS non(femme).
Si non(discours_sans_cesse) ET femme ALORS non(digne_d_eloges).
Si discours_sans_cesse ET digne_d_eloges ET non(populaire) ALORS non(gardant_secret).
Si apte_a_etre_depute ET non(valant_son_pesant_d_or) ALORS non(gardant_secret).
Si emploie_influence_a_des_fins_meritoires ET sans_pretention ET populaire ALORS non(gardant_secret).
Si discours_sans_cesse ET digne_d_eloges ET emploie_influence_a_des_fins_meritoires ALORS non(gardant_secret).
Si discours_sans_cesse ET bienfaiteur_du_peuple ET gardant_secret ALORS non(emploie_influence_a_des_fins_meritoires).
Si discours_sans_cesse ET esprit_clair ET gardant_secret ALORS non(emploie_influence_a_des_fins_meritoires).
Si digne_d_eloges ET emploie_influence_a_des_fins_meritoires ET populaire ALORS non(gardant_secret).
Si gardant_secret ET non(emploie_influence_a_des_fins_meritoires) ALORS non(apte_a_etre_depute).
Si gardant_secret ET non(sans_pretention) ALORS non(apte_a_etre_depute).
Si digne_d_eloges ET valant_son_pesant_d_or ALORS non(femme).
Si femme ET valant_son_pesant_d_or ALORS non(digne_d_eloges).
Si apte_a_etre_depute ET non(souvenir_demeurant_imperissable) ALORS non(gardant_secret).
Si apte_a_etre_depute ET populaire ALORS non(gardant_secret).
Si apte_a_etre_depute ET affiche_dans_vitrines ALORS non(gardant_secret).
Si femme ET emploie_influence_a_des_fins_meritoires ALORS non(bienfaiteur_du_peuple).
Si bienfaiteur_du_peuple ET femme ALORS non(emploie_influence_a_des_fins_meritoires).
Si discours_sans_cesse ET gardant_secret ALORS non(apte_a_etre_depute).
Si apte_a_etre_depute ET discours_sans_cesse ALORS non(gardant_secret).
Si digne_d_eloges ET emploie_influence_a_des_fins_meritoires ALORS non(femme).
Si esprit_clair ET femme ALORS non(emploie_influence_a_des_fins_meritoires).
Si femme ALORS non(apte_a_etre_depute).
Si apte_a_etre_depute ALORS non(femme).

Maintenant, grâce à la base compilée, nous vérifions que l'on peut obtenir par exemple **non(apte_a_etre_depute)** de la base de faits **{gardant_secret, populaire}** à l'aide d'un simple chaînage avant. On peut aussi vérifier que la clause **non(femme) OU non(apte_a_etre_depute)** est bien conséquence logique de la base par chaînage avant. Il suffit pour cela d'ajouter **{femme}** et de vérifier que l'on obtient bien **non(apte_a_etre_depute).**

D. La centrale.

Le troisième exemple traité est un problème présenté pour la première fois dans [Sie 87] et traitant d'une centrale nucléaire hybride. Il est présenté sous le titre "La centrale". Ce problème a été inspiré à Pierre Siegel par un problème réel posé par EDF au sujet d'une centrale nucléaire.

Si le voyant de température est rouge alors d'une part il n'y a plus d'eau ou il n'y a plus d'huile ou la courroie est cassée, et, d'autre part les circuits secondaires sont alimentés.

Le voyant d'huile est rouge si et seulement si les circuits secondaires sont alimentés et il n'y a plus d'huile ou le moteur ne tourne pas.

Le compte tour est positif si et seulement si le moteur tourne et la courroie n'est pas cassée.

La bobine est alimentée si et seulement si l'ampèremètre de charge est positif et il n'y a pas de court circuit, ou, les circuits secondaires sont alimentés et le fusible n'est pas fondu.

Il y a un court circuit si et seulement si la batterie est en court circuit et le fusible n'est pas fondu.

Une batterie en court circuit est vide.

Si l'ampèremètre de charge est positif et la batterie est en court circuit alors le fusible est fondu.

Si le moteur tourne alors la bobine est alimentée.

Si les phares marchent alors les circuits secondaires sont alimentés.

Ce problème se formule en 30 clauses [Sie 87] que voici et qui donnent par calcul des variantes 79 règles et 16 propositions.

→ non voyant_temperature_rouge OU plus_d_eau OU plus_d_huile OU courroie_cassee .
→ non voyant_temperature_rouge OU circuits_secondaires_alimentes .
→ non voyant_d_huile_rouge OU circuits_secondaires_alimentes .
→ non voyant_d_huile_rouge OU plus_d_huile OU non moteur_tourne .
→ non plus_d_huile OU non circuits_secondaires_alimentes OU voyant_d_huile_rouge .
→ moteur_tourne OU non circuits_secondaires_alimentes OU voyant_d_huile_rouge .
→ non moteur_tourne OU courroie_cassee OU compte_tour_positif .
→ non compte_tour_positif OU moteur_tourne .
→ non compte_tour_positif OU non courroie_cassee .
→ non amperemetre_charge_positif OU moteur_tourne .
→ non amperemetre_charge_positif OU non redresseur_coupe .
→ non amperemetre_charge_positif OU non courroie_cassee .
→ non moteur_tourne OU redresseur_coupe OU courroie_cassee OU amperemetre_charge_positif .
→ non bobine_alimentee OU amperemetre_charge_positif OU circuits_secondaires_alimentes .
→ non bobine_alimentee OU amperemetre_charge_positif OU non fusible_fondu .
→ non bobine_alimentee OU non court_circuit OU circuits_secondaires_alimentes .
→ non bobine_alimentee OU non court_circuit OU non fusible_fondu .
→ non amperemetre_charge_positif OU court_circuit OU bobine_alimentee .
→ non circuits_secondaires_alimentes OU fusible_fondu OU bobine_alimentee .
→ non circuits_secondaires_alimentes OU non batterie_vide OU bobine_alimentee .
→ non circuits_secondaires_alimentes OU non batterie_vide OU non fusible_fondu .
→ batterie_vide OU circuits_secondaires_alimentes .
→ non bobine_alimentee OU fusible_fondu OU circuits_secondaires_alimentes .
→ non court_circuit OU batterie_court_circuit .
→ non court_circuit OU non fusible_fondu .
→ non batterie_court_circuit OU batterie_vide .
→ non batterie_court_circuit OU fusible_fondu OU court_circuit .
→ non amperemetre_charge_positif OU non batterie_court_circuit OU fusible_fondu .
→ non moteur_tourne OU bobine_alimentee .
→ non phares_marchent OU circuits_secondaires_alimentes .

De **{amperemetre_charge_positif}** on ne peut déduire **non court_circuit.**

De **{non compte_tour_positif}** on ne peut déduire **non amperemetre_charge_positif.**

La base de clauses obtenue après achèvement et simplifications possède 35 clauses. La base de règles obtenue contient les 69 règles que voici:

SI non(plus_d_eau) ET non(plus_d_huile) ET non(courroie_cassee) ALORS non(voyant_temperature_rouge).
SI voyant_temperature_rouge ET non(plus_d_huile) ET non(courroie_cassee) ALORS plus_d_eau.
SI voyant_temperature_rouge ET non(plus_d_eau) ET non(courroie_cassee) ALORS plus_d_huile.
SI voyant_temperature_rouge ET non(plus_d_eau) ET non(plus_d_huile) ALORS courroie_cassee.
SI non(circuits_secondaires_alimentes) ALORS non(voyant_temperature_rouge).
SI voyant_temperature_rouge ALORS circuits_secondaires_alimentes.

SI voyant_d_huile_rouge ALORS circuits_secondaires_alimentes.
SI non(circuits_secondaires_alimentes) ALORS non(voyant_d_huile_rouge).
SI voyant_d_huile_rouge ET moteur_tourne ALORS plus_d_huile.
SI non(plus_d_huile) ET moteur_tourne ALORS non(voyant_d_huile_rouge).
SI non(plus_d_huile) ET voyant_d_huile_rouge ALORS non(moteur_tourne).
SI circuits_secondaires_alimentes ET non(voyant_d_huile_rouge) ALORS non(plus_d_huile).
SI plus_d_huile ET non(voyant_d_huile_rouge) ALORS non(circuits_secondaires_alimentes).
SI plus_d_huile ET circuits_secondaires_alimentes ALORS voyant_d_huile_rouge.
SI non(voyant_d_huile_rouge) ET non(moteur_tourne) ALORS non(circuits_secondaires_alimentes).
SI circuits_secondaires_alimentes ET non(moteur_tourne) ALORS voyant_d_huile_rouge.
SI circuits_secondaires_alimentes ET non(voyant_d_huile_rouge) ALORS moteur_tourne.
SI moteur_tourne ET non(compte_tour_positif) ALORS courroie_cassee.
SI non(courroie_cassee) ET non(compte_tour_positif) ALORS non(moteur_tourne).
SI non(courroie_cassee) ET moteur_tourne ALORS compte_tour_positif.
SI compte_tour_positif ALORS moteur_tourne.
SI non(moteur_tourne) ALORS non(compte_tour_positif).
SI compte_tour_positif ALORS non(courroie_cassee).
SI courroie_cassee ALORS non(compte_tour_positif).
SI amperemetre_charge_positif ALORS moteur_tourne.
SI redresseur_coupe ALORS non(amperemetre_charge_positif).
SI amperemetre_charge_positif ALORS non(redresseur_coupe).
SI amperemetre_charge_positif ALORS non(courroie_cassee).
SI non(courroie_cassee) ET moteur_tourne ET non(redresseur_coupe) ALORS amperemetre_charge_positif.
SI non(courroie_cassee) ET moteur_tourne ET non(amperemetre_charge_positif) ALORS redresseur_coupe.
SI non(circuits_secondaires_alimentes) ET non(amperemetre_charge_positif) ALORS non(bobine_alimentee).
SI bobine_alimentee ET fusible_fondu ALORS amperemetre_charge_positif.
SI non(amperemetre_charge_positif) ET fusible_fondu ALORS non(bobine_alimentee).
SI non(amperemetre_charge_positif) ET bobine_alimentee ALORS non(fusible_fondu).
SI non(bobine_alimentee) ET non(fusible_fondu) ALORS non(circuits_secondaires_alimentes).
SI circuits_secondaires_alimentes ET non(fusible_fondu) ALORS bobine_alimentee.
SI circuits_secondaires_alimentes ET non(bobine_alimentee) ALORS fusible_fondu.
SI non(bobine_alimentee) ET batterie_vide ALORS non(circuits_secondaires_alimentes).
SI fusible_fondu ET batterie_vide ALORS non(circuits_secondaires_alimentes).
SI circuits_secondaires_alimentes ET batterie_vide ALORS non(fusible_fondu).
SI circuits_secondaires_alimentes ET fusible_fondu ALORS non(batterie_vide).
SI non(batterie_vide) ALORS circuits_secondaires_alimentes.
SI non(circuits_secondaires_alimentes) ALORS batterie_vide.
SI bobine_alimentee ET non(fusible_fondu) ALORS circuits_secondaires_alimentes.
SI non(circuits_secondaires_alimentes) ET non(fusible_fondu) ALORS non(bobine_alimentee).
SI non(circuits_secondaires_alimentes) ET bobine_alimentee ALORS fusible_fondu.
SI non(batterie_court_circuit) ALORS non(court_circuit).
SI court_circuit ALORS batterie_court_circuit.
SI court_circuit ALORS non(fusible_fondu).
SI fusible_fondu ALORS non(court_circuit).
SI batterie_court_circuit ALORS batterie_vide.
SI non(batterie_vide) ALORS non(batterie_court_circuit).
SI non(court_circuit) ET batterie_court_circuit ALORS fusible_fondu.
SI non(fusible_fondu) ET batterie_court_circuit ALORS court_circuit.
SI non(fusible_fondu) ET non(court_circuit) ALORS non(batterie_court_circuit).
SI non(fusible_fondu) ET batterie_court_circuit ALORS non(amperemetre_charge_positif).
SI non(bobine_alimentee) ALORS non(moteur_tourne).
SI moteur_tourne ALORS bobine_alimentee.
SI phares_marchent ALORS circuits_secondaires_alimentes.
SI non(circuits_secondaires_alimentes) ALORS non(phares_marchent).

Si non(plus_d_eau) ET non(courroie_cassee) ET non(voyant_d_huile_rouge) ALORS non(voyant_temperature_rouge).

Si non(voyant_d_huile_rouge) ET bobine_alimentee ALORS moteur_tourne.

Si non(amperemetre_charge_positif) ET non(redresseur_coupe) ALORS non(compte_tour_positif).

Si non(compte_tour_positif) ALORS non(amperemetre_charge_positif).

Si amperemetre_charge_positif ALORS non(court_circuit).

Si circuits_secondaires_alimentes ET non(court_circuit) ALORS non(batterie_court_circuit).

Si non(courroie_cassee) ET non(voyant_d_huile_rouge) ET non(redresseur_coupe) ET batterie_court_circuit ALORS non(circuits_secondaires_alimentes).

Si non(plus_d_eau) ET non(voyant_d_huile_rouge) ET fusible_fondu ALORS non(voyant_temperature_rouge).

Si voyant_temperature_rouge ET non(plus_d_eau) ET fusible_fondu ALORS voyant_d_huile_rouge.

Maintenant, grâce à la base compilée, nous vérifions que l'on peut obtenir par exemple **non(court_circuit)** de la base de faits **{amperemetre_charge_positif}** à l'aide d'un simple chaînage avant.

Les performances de l'achèvement sur ces trois exemples peuvent se résumer dans le tableau suivant :

	Logiciens [Car 66]	Députés [Car 66]	Centrale [Sie 87]
nb initial de clauses	15	20	30
Equivalence en règles	54	62	79
nb de propositions	11	16	16
nb de modèles calculés	40	148	33
temps de compilation	0.86sec	21sec65	3sec75
nb de clauses obtenues	66	66	32
équivalence en règles	120	113	69

Le temps de compilation a été obtenu par la méthode de l'arbre sémantique suivie de la traversée de matrices ainsi que les optimisations présentées dans cette thèse avec une machine SUN 3/60.

Le nombre initial de règles est obtenu par calcul des variantes sur l'ensemble initial de clauses.

Le nombre de règles obtenues n'est pas représentatif puisque nous utilisons par la suite un chaînage avant sur les clauses. Seul le nombre de clauses obtenues est intéressant.

Le calcul des questions utiles dans les systèmes experts

I. Introduction.

Après avoir traité dans la première partie de cette thèse du chaînage avant trivalué nous allons aborder maintenant le problème du chaînage mixte utilisé dans de très nombreux générateurs de systèmes experts. Ce chaînage mixte, lancé sur un but particulier, réunit un algorithme de chaînage avant pour effectuer les déductions et un algorithme de calcul de questions à poser à l'utilisateur permettant de converger vers le but recherché. Malheureusement dans la plupart des systèmes, le calcul de cette question n'est pas cohérent avec la manière dont le chaînage avant effectue ses déductions. Il en résulte que ces systèmes posent des questions qui n'amènent souvent aucune information sur le but recherché et qui sont donc inutiles comme dans l'exemple suivant:

Si non b alors a
Si c alors b
Si d alors non b
Si e alors non c
Si f alors d

En lançant le chaînage mixte sur le but a, la plupart des systèmes posent la question "Est-ce que e?". Or e ne permet pas d'avancer dans la déduction de a car e donne non c et que rien n'est plus alors déduit. La seule question intéressante ici est "Est-ce que f?"

Nous présentons dans cette partie une structuration des différents algorithmes de calcul d'une question en définissant tout d'abord précisément deux types de questions: les questions pertinentes et les questions intelligentes. Nous établissons ensuite deux niveaux distincts dans le calcul des questions: le niveau logique et le niveau heuristique. Nous étendons ensuite nos algorithmes aux bases de connaissances avec faits symboliques pour permettre de résoudre des problèmes tels que celui-ci:

Si b<>'1' et b<>'2' alors a
Si d alors b=c
Si x alors c='1'
Si y alors c='2'
Si z alors c='3'

En lançant le chaînage mixte sur le but a, la plupart des systèmes posent d'abord la question "Est-ce que d?" ce qui est correct, mais si d est vrai ils posent ensuite les questions "Est-ce que x?" puis "Est-ce que y?" et enfin "Est-ce que z?" alors que seule la dernière est utile pour déduire a. Les deux premières questions posées sont parfaitement inutiles.

Les algorithmes sont enfin étendus aux méta-valeurs Connu, Inconnu et Indéterminé utilisées notamment dans IS [IS 86] ou SPT [Del 87a].

II. Présentation du cas booléen pur.

1. Représentation externe.

On appelle atome une variable propositionnelle. Un littéral est un atome ou la négation d'un atome. Cette négation est représentée par le symbole '¬' devant le nom de l'atome. Ainsi A et ¬A sont deux littéraux distincts, l'un positif, l'autre négatif, composés tous deux de l'atome A. Par la suite nous nous efforcerons d'utiliser les termes 'atomes' et 'littéraux' plutôt que le terme 'fait' qui peut prêter à confusion. Comme nous l'avons dit précédemment, pour que nous puissions avoir un système à trois valeurs il faut pouvoir représenter l'information négative aussi bien en partie condition qu'en partie conclusion de règle. Nous obtenons alors les définitions suivantes:

Définitions.
Soit Litt l'ensemble infini dénombrable des littéraux. Une règle est une formule propositionnelle de la forme $L_1,...,L_n \rightarrow L$ avec $n>0$ et $L,L_1,...,L_n \in$ Litt. Une **base de règles** est un ensemble de règles. Une **mémoire de travail** est un ensemble de littéraux. Si p et q sont des littéraux on note $p \angle q$ et l'on dit que p dépend strictement de q ssi il existe une règle ayant q dans sa partie condition et p comme conclusion (ex $q \rightarrow p$). On note $<$ la cloture transitive de $\angle$. $p<q$ indique que p dépend au sens large de q. On dira qu'une base est **sans cycle** ssi il n'existe pas de littéral p tel que $p<p$. On dira que p est un **littéral de base** s'il n'existe pas de littéral q tel que $p<q$.

Les bases que nous traiterons ici seront toujours des bases sans cycle.

Pour des raisons de simplicité nous présenterons chaque formalisme utilisé à l'aide d'une grammaire BNF (Backus Naur Form)

Méta-syntaxe.

::=	symbole de définition syntaxique
<...>	délimiteurs d'entité syntaxique
|	alternative exclusive
e	mot vide

<base> ::= <regle> | <regle> <base>
<regle> ::= <condition> → <conclusion>
<condition> ::= <fait> , <condition> | <fait>
<conclusion> ::= <fait>
<fait> ::= <litteral>
<litteral> ::= <atome> | ¬ <atome>

Le symbole ',' correspond au connecteur 'Et'. La mémoire de travail (ensemble de littéraux déduits par le chaînage avant pendant une cession de travail, aussi appelée base de faits) est initialement constituée des littéraux connus dès le départ (c'est pourquoi les règles avec une partie condition vide ne sont pas acceptées par notre syntaxe dans la base de règles). S'il n'y en a pas la mémoire de travail est alors vide. Les faits apparaissant en partie condition d'une règle sont parfois appelés prémisses de cette règle.

Le choix de représenter une règle à l'aide du symbole '→' et non à l'aide du Si-Alors classique a été effectué uniquement pour pouvoir représenter les règles de manière plus concise.

Ne permettre qu'une conjonction de faits en partie condition et un seul fait en partie conclusion allie à la fois simplicité et puissance. En effet cela permet de représenter toutes les règles utilisées habituellement dans les systèmes experts sans variables, que ce soient les conjonctions de faits en conclusion de règle ou encore une formule construite avec les opérateurs 'et', 'ou' et 'non' en condition. Il suffit pour cela d'utiliser les méthodes de transformation de formules bien connues (théorèmes de distributivité, lois de De Morgan, loi de la double négation) qui permettront de représenter cette connaissance moyennant un nombre de règles plus important. Se référer à tout livre de logique élémentaire comme [FG 87] pour plus de précisions.

2. Système de déduction.

Nous considérons que seul le chaînage avant permet d'inférer de nouveaux faits. C'est le système le plus couramment utilisé notamment parce qu'il permet de tirer pleinement parti de chaque fait ajouté dans la mémoire de travail (MT), mais aussi parce qu'il permet d'obtenir des réponses multiples [Del 87a] qui peuvent d'ailleurs être des conséquences que l'utilisateur n'attendait pas. Le chaînage arrière comme système de déduction n'est vraiment intéressant que pour les systèmes avec variables tels que le langage Prolog [Col 72] qui effectuent plus de la démonstration de théorèmes que de la déduction.

Dans le cadre que nous nous sommes fixés, la saturation par chaînage avant ne pose pas de problème. On montre en effet facilement qu'un chaînage avant monotone sur une base sans variables est fini, ainsi d'ailleurs que sur une base avec variables et sans symbole de fonction comme Datalog [MW 88].

Notre algorithme de chaînage avant travaille sur les littéraux (approche classique). Il travaille donc en logique trivaluée comme nous l'avons montré dans la première partie de cette thèse.

Nous nous intéresserons surtout au chaînage mixte très utilisé dans les systèmes experts car il permet de concentrer la recherche sur un but donné et offre la possibilité au système de poser des questions à l'utilisateur en cours de déduction. C'est la pertinence de cette question qui permet de qualifier un système d'intelligent, il est donc primordial que celle-ci soit calculée correctement. Ce calcul d'une question pertinente sera l'un des problèmes que nous tenterons de résoudre dans cet article.

Notre chaînage mixte utilisera donc un algorithme de chaînage avant pour effectuer les déductions et un algorithme de type chaînage arrière pour calculer la question à poser à l'utilisateur. Cette définition du chaînage mixte suit d'assez près le chaînage mixte proposé dans les générateurs de systèmes experts comme IS [IS 86] et les algorithmes présentés dans [Del 87a]. Nous ne reviendrons pas sur le fonctionnement des chaînages avant et arrière ni sur les arbres et/ou, les arbres de listes de buts, les graphes de dépendances ou encore l'art et la manière d'écrire une base de connaissances. Pour le lecteur qui ne serait pas familier avec ces notions se référer par exemple à [Cor 84], [Gan 85], [Del 87a] ou [FG 87].

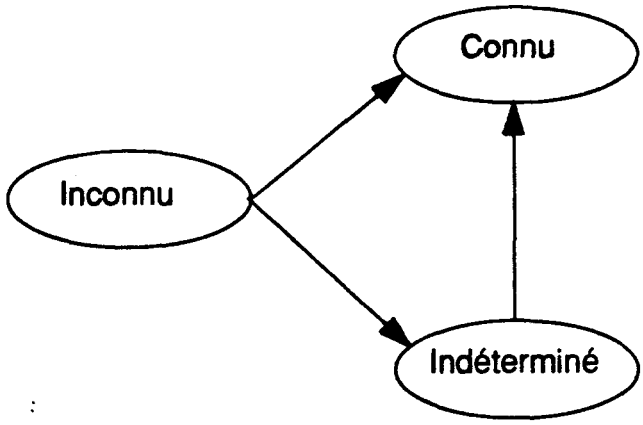
3. Représentation interne.

Dans le cas du chaînage mixte, le système a la possibilité de questionner l'utilisateur sur certains atomes. Or quand une question est posée, l'utilisateur peut très bien ignorer la réponse. On lui donne alors la possibilité de répondre 'Je ne sais pas'. Cette option couramment admise est très utile dans les systèmes experts. Il nous est donc nécessaire de différencier un atome qui a déjà été demandé à l'utilisateur d'un atome inconnu non encore demandé. Cela permet au système de ne pas poser plusieurs fois la même question à l'utilisateur. Dans le cas où l'utilisateur a répondu Vrai ou Faux cette différenciation se fait facilement puisqu'il suffit de tester la présence de cet atome dans la mémoire de travail, mais si l'utilisateur a répondu 'Je ne sais pas' aucune valeur n'est affectée à cet atome. Il nous faut donc un moyen de distinguer un fait inconnu déjà demandé à l'utilisateur d'un fait inconnu qui n'a pas été encore demandé. Nous avons choisi de représenter chaque littéral de la mémoire de travail sous forme d'un triplet: Nom, Méta-valeur, Valeur.

La valeur d'un atome est donc soit 'Vrai' soit 'Faux'. La valeur 'Inconnu' n'est pas représentée directement. C'est l'absence de l'atome dans la mémoire de travail qui fait office d'Inconnu.

Pour chaque atome présent dans la mémoire de travail on associe une Méta-valeur. Elle peut être soit 'Connu' si l'atome a été affecté à une valeur (donc Vrai ou Faux), soit 'Indéterminé' si l'utilisateur a répondu 'Je ne sais pas' à une question sur cet atome. Une troisième Méta-valeur 'Inconnu' est associée à chaque atome inconnu, elle n'est donc pas explicitement représentée puisque comme nous l'avons dit, c'est l'absence de l'atome dans la mémoire de travail qui fait office d'inconnu.

Les méta-valeurs sont liées par le schéma suivant :



Initialement chaque atome est inconnu. Il peut devenir connu (Vrai ou Faux) soit par précision de l'utilisateur soit par déduction du chaînage avant. Il peut aussi devenir Indéterminé si l'utilisateur répond 'je ne sais pas' à une question sur cet atome puis passer ensuite à Connu par déduction du chaînage avant.

La mémoire de travail (MT) vérifie bien évidemment le **principe de non-contradiction**: Deux littéraux opposés ne peuvent être présents simultanément dans MT. Toute tentative d'ajout dans MT d'un littéral dont l'opposé est déjà présent provoque une contradiction. Elle ne vérifie pas par contre le principe du tiers-exclus: Un atome n'est pas nécessairement vrai ou faux, il peut être inconnu. C'est d'ailleurs la raison pour laquelle nous nous basons sur la logique trivaluée présentée dans la première partie de cette thèse pour effectuer cette étude.

4. Chaînage avant par saturation.

Mémoire de travail (MT).

Pour ajouter un littéral d'atome associé Ato à MT deux cas se présentent: si le littéral est positif on ajoute le triplet Ato,Connu,Vrai à MT et s'il est négatif on ajoute Ato,connu,Faux. Que l'atome correspondant soit Inconnu ou Indéterminé, il devient alors Connu.

Ce sont donc des littéraux qui sont déduits par le chaînage avant, bien qu'ils soient représentés dans MT par un atome associé à une valeur (Ceci afin de faciliter la programmation).

Satisfiabilité des prémisses.

Dans un système à trois valeurs, un littéral n'est satisfait que s'il est présent dans MT. Un littéral positif est satisfait si l'atome associé est Connu Vrai dans MT, un littéral négatif est satisfait si l'atome associé est Connu Faux dans MT.

La partie condition d'une règle est satisfaite si tous les littéraux de la partie condition sont satisfaits dans MT.

Détection des contradictions.
Un littéral d'atome associé Ato peut contredire MT de deux manières: si le littéral est positif et que le triplet Ato,Connu,Faux est présent dans MT ou si le littéral est négatif et que le triplet Ato,Connu,Vrai est présent.

Après ces quelques précisions nous pouvons alors définir algorithmiquement notre chaînage avant par saturation. L'algorithme est décrit d'une manière assez générale pour pouvoir le récupérer par la suite en modifiant uniquement les trois rubriques précédentes.

Algorithme de chaînage avant par saturation.

```
Début
Tant que certaines règles dont la conclusion n'est pas déjà
     présente dans MT ont leur partie condition
     satisfaite
          déterminer la première de ces règles.
          Si la partie conclusion ne contredit aucun élément de
          MT Alors
               ajouter la conclusion à MT.
          Sinon
               retourner 'base inconsistante'
               Arrêt
          Finsi
     Fintq
Arrêt
```

Cet algorithme est très simple. Il pourrait être accéléré en ajoutant des techniques de désactivation de règles. Par exemple désactiver à chaque étape la règle venant d'être utilisée ou encore désactiver toutes les règles dont la conclusion est déjà dans la mémoire de travail (MT) ainsi que les règles dont les prémisses sont en contradiction avec MT puisque ces règles ne pourront plus être utilisées. Se référer à [Gha 88] ou [Frot 90] pour les optimisations de structures de données afin d'accélérer l'exécution des algorithmes de systèmes experts.

5. Chaînage mixte.

Le chaînage mixte est l'algorithme qui permet de relier le chaînage avant qui effectue les déductions et l'algorithme de calcul de la question qui établit une question supposée intéressante à poser à l'utilisateur. C'est grâce à lui que l'utilisateur peut lancer les différents algorithmes sur l'évaluation d'un but. Le système a alors la possibilité de poser des questions à l'utilisateur en cours de déduction pour tenter d'enrichir ses connaissances. A la fin de

l'exécution le but recherché est vrai, faux ou inconnu s'il n'a été prouvé ni vrai ni faux.

Il nous faut maintenant établir comment lancer le chaînage mixte. On pourrait imaginer un système où le but initial est un littéral et où le système essaie de dire si oui ou non ce littéral est satisfait. Si le littéral x n'est pas satisfait il faut alors relancer le système sur l'opposé de x pour connaître la valeur de l'atome x (nous appellerons cela plus tard un interpréteur à simple tentative). Ce type de système ne serait pas très utile dans le cadre des systèmes experts où l'on préfère obtenir directement la valeur de l'atome demandé (ce que nous appellerons plus tard un interpréteur à double tentative). De ce fait, notre chaînage mixte sera toujours lancé sur un atome. En considérant connu l'algorithme de calcul de la question, l'algorithme de chaînage mixte est alors le suivant.

```
Algorithme Chainage-Mixte(Atome)

Saturer la base par chaînage avant.

Tant que le but n'est pas connu (ni vrai, ni faux)
     qu'aucune contradiction n'a été détectée
     qu'une question utile peut être posée
          poser cette question.
          prendre en compte la réponse de l'utilisateur.
          saturer par chaînage avant.
Fintq

Si le but est vrai dans MT
     afficher 'Atome vrai'
Sinon
     Si le but est faux dans MT
          afficher 'Atome faux'
     Sinon
          afficher 'Atome non déductible'
     Fsi
Fsi
```

Par construction de l'algorithme de chaînage mixte, le calcul d'une question utile se fera toujours sur une base précédemment saturée par chaînage avant.

Il est à noter que certains systèmes se vantent de posséder un "véritable" chaînage mixte alors que celui-ci n'est en fait qu'un chaînage arrière modifié. C'est le cas notamment de Guru qui ne sait pas calculer toutes les conséquences d'un fait ajouté à la mémoire de travail. Il suffit pour s'en assurer de charger une base Br={c→b , b→a , b→d , d→e} avec comme but a, et de répondre vrai à la question c. On remarque alors que e n'est pas déduit !.

III. Calcul d'une question utile.

L'interface utilisateur

Précisons d'abord comment la question sera posée à l'utilisateur. Nos règles étant formées de littéraux, la question utile qui sera calculée sera donc un littéral. Mais il serait inutile et compliqué de demander à l'utilisateur la valeur d'un littéral, la réponse 'Vrai' à la question ¬x par exemple, donnant x faux !. Nous décidons alors que quel que soit le littéral choisi pour la question, c'est la valeur de l'atome correspondant qui sera demandée à l'utilisateur.

Trois réponses peuvent être fournies à la question sur l'atome x:

- L'utilisateur répond vrai.
On ajoute alors x,connu,vrai à la mémoire de travail.

- L'utilisateur répond faux.
On ajoute alors x,connu,faux à la mémoire de travail.

- L'utilisateur répond 'Je ne sais pas'.
On ajoute alors x,Indéterminé,_ à la mémoire de travail.

Attention à ne pas confondre une question intéressante (qui est un littéral) avec la manière dont la question est posée à l'utilisateur (ici c'est l'atome correspondant qui est demandé). Ce choix est uniquement pratique, on pourrait tout aussi bien demander à l'utilisateur si le littéral est satisfait.

Faits demandables

Nous l'avons dit précédemment, le chaînage mixte a la possibilité de poser des questions à l'utilisateur. Or il ne doit évidemment pas questionner sur tout littéral rencontré. Ceci présuppose donc que certains faits soient demandables à l'utilisateur et d'autres non. Couramment, les seuls faits demandables sont les littéraux qui ne figurent dans aucune conclusion de règle (ce que nous avons appelé littéraux de base). D'après notre syntaxe les faits de base sont des littéraux. Ce sont des littéraux qui devraient donc être demandables mais avoir un littéral A demandable tandis que ¬A ne l'est pas indique une base mal structurée, voir incorrecte. Nous considérons donc que dès qu'un littéral est demandable son opposé l'est aussi. Il ne nous est donc pas nécessaire de préciser l'option demandable pour chaque littéral mais simplement pour chaque atome. Par extension un littéral sera dit demandable si l'atome associé est demandable. Le cas de figure cité précédemment est donc interdit.

Par défaut notre système considère que dès qu'un littéral est de base, l'atome associé est demandable. Il nous paraît cependant nécessaire de pouvoir modifier cette option, ne serait-ce que pour pouvoir poser des questions sur des faits intermédiaires. Dans le cas de modifications de cette option par défaut dans nos exemples, nous préciserons alors la liste des atomes

demandables.

Le calcul d'une question utile

Poser une question utile à l'utilisateur est une tâche ardue. C'est l'un des critères qui permet de juger si le système est 'intelligent' ou non. En effet le système doit pouvoir déduire par chaînage avant le but recherché en un minimum de questions. Calculer cette question n'est pas facile. Il nous faut d'abord définir précisément ce qu'est une question utile. Pour cela nous introduisons deux définitions de questions: Les questions pertinentes et les questions intelligentes.

Définition.

Soit Br une base de règles, L une conjonction de littéraux inconnus demandables et x un littéral, L est une **liste de questions intelligentes** associée à x ssi :

- Br$\cup$L est consistant
- L$\rightarrow$x ou L$\rightarrow\neg$x est conséquence trivaluée de Br
- le fait d'enlever un des littéraux de L ne permet plus d'avoir ni L$\rightarrow$x ni L$\rightarrow\neg$x (critère de minimalité).

Ou de manière formelle: L=$\{lit_1,...,lit_n\}$ tel que Br$\cup$L consistant, Br$\cup$L $\models_T$ x $\vee \neg$x et que $\forall i \in \{1,...,n\}$ Br$\cup$(L-lit_i) $\not\models$ x $\vee \neg$x.

Nous avons montré dans la première partie que le chaînage avant classique est un calculateur de conséquences trivaluées et donc on peut de manière intuitive considérer qu'une liste de questions intelligentes associée à x est une liste minimale de littéraux inconnus demandables permettant au chaînage avant sur toute la base de déduire une valeur sur x.

Définition.

Soit But un littéral fixé, un littéral Q est une **question intelligente** associée à x ssi il existe une liste de questions intelligentes associée à But contenant Q.

Malheureusement le calcul d'une question intelligente est un problème algorithmiquement complexe (vraisemblablement de complexité exponentielle). C'est pourquoi nous introduisons une définition plus faible. Nous cherchons alors des algorithmes qui permettent d'obtenir une question la plus proche possible d'une question intelligente en utilisant des techniques sensiblement plus rapides. On propose les définitions suivantes:

Définition.

Soit Br une base de règles, L une conjonction de littéraux inconnus demandables et x un littéral, L est une **liste de questions pertinentes** associée à x ssi il existe un sous ensemble R de n règles de Br telles que

- R$\cup$L est consistant
- L$\rightarrow$x ou L$\rightarrow\neg$x est conséquence trivaluée de R
- le fait d'enlever un des littéraux de L ne permet plus d'obtenir ni L$\rightarrow$x ni L$\rightarrow\neg$x (critère de minimalité).

Ou de manière formelle: $L=\{lit_1,...,lit_n\}$ tel que $\exists R \subseteq Br$ avec $R \cup L$ consistant, $R \cup L \models_T x \vee \neg x$ et que $\forall i \in \{1,...,n\}$ $R \cup (L-lit_i) \not\models x \vee \neg x$

De manière intuitive on peut considérer qu'une liste de questions pertinentes associée à x est une liste minimale de littéraux inconnus demandables permettant au chaînage avant sur un ensemble restreint de règles de déduire une valeur sur x.

Définition.
Soit But un littéral fixé, un littéral Q est une **question pertinente** associée à But ssi il existe une liste de questions pertinentes associée à But contenant Q.

Remarque.
Ramener la base de connaissances à un sous-ensemble de règles est assez naturel. On part du principe que calculer une question intelligente sur toute la base est un problème trop complexe et l'on se restreint à une partie de la base que l'on pourrait appeler un **cône de déduction** du but recherché, pour pouvoir calculer une question par un algorithme de type chaînage arrière de manière rapide. La minimalité de la liste de questions intelligentes inclut donc le calcul de toutes les listes de questions pertinentes simultanément pour ne garder ensuite que les minimales.

Proposition.
Toute question intelligente est une question pertinente.

Preuve.
Il suffit de prendre toute la base comme cône de déduction (R=Br).♦

L'inverse est évidemment faux comme le montrent les exemples suivants:

Exemple 1

r1 b,c → a	MT=∅ But: a
r2 c → a	b est une question pertinente (avec R={r1}) mais pas une question intelligente. c est une question intelligente (donc pertinente).

La question b n'est pas intelligente car la conséquence trivaluée b,c→a n'est pas conséquence minimale puisque c→a est conséquence trivaluée de Br. Par contre b est une question pertinente car il existe un sous-ensemble de R={b, c → a} de Br et un ensemble de littéraux inconnus demandables L={b,c} telles que a ∈ Sat(R∪L) avec L minimale.

Exemple 2

r1 b → a	MT=∅ But: a
r2 c → a	b est une question pertinente (avec R={r1}) mais
r3 b → x	pas une question intelligente. c est une question
r4 b → ¬x	intelligente (donc pertinente).

La question b n'est pas intelligente car b → a n'est pas conséquence trivaluée de Br (à cause de la contradiction). Par contre b est pertinente car en prenant R={b → a} et L={b} on a bien a ∈ Sat(R∪L) avec L minimale.

D'une manière intuitive on peut dire qu'une question pertinente ne corrige pas une base mal écrite. Si l'utilisateur écrit des redondances ou des incohérences dans sa base, des questions superflues risquent d'être posées (mais pas des questions stupides !!). Une question intelligente par contre, "corrige" les défauts d'une base de connaissances.

Pour obtenir une question qui fait avancer le raisonnement il faut donc calculer une liste de questions pertinentes ou intelligentes et demander la valeur de l'un des atomes utilisés dans cette liste à l'utilisateur. C'est ce que nous appelons le niveau logique de calcul de la question. Mais il se peut qu'il existe plusieurs questions pertinentes possibles. Dans ce cas il faut choisir quelle liste de questions utiliser (la première, la plus courte ...) et quel atome de cette liste doit être demandé. C'est ce que nous appelons le niveau heuristique de calcul de la question.

On obtient donc un calcul de la question à deux niveaux:

- Le niveau logique, qui définit le niveau d'intelligence de la question.
- Le niveau heuristique, qui définit le niveau d'optimisation de la question posée.

Remarque.
Nous traitons ici le cas du calcul d'une question dans un système trivalué. Le calcul d'une question pertinente ou intelligente en logique bivaluée est un autre problème. Voir par exemple [Dem 90] et [DF 90] pour le traitement du cas bivalué.

Exemple 3

b → a	MT=∅ But: a.
c → a	e est une question intelligente en
non(b), non(c) → non(d)	bivalué.b ou c sont des questions
e → d	intelligentes en trivalué.

Une question intelligente ou pertinente en bivalué est intéressante pour un système de déduction bivalué. Nous avons choisi le cas trivalué car c'est celui utilisé classiquement dans les systèmes experts puisqu'il est attaché au chaînage avant. Nous avons montré de plus dans la première partie de cette thèse que l'on peut ramener le cas bivalué au cas trivalué par achèvement des bases de connaissances.

1. Le niveau logique.

1.1. Définition.

L'objectif du **niveau logique** est de calculer une liste de questions pertinentes ou intelligentes, c'est-à-dire une liste de littéraux inconnus demandables qui, s'ils étaient satisfaits, permettraient de déduire le but recherché par chaînage avant.

Il est important de noter que la plupart des générateurs de systèmes experts ne satisfont pas à ce niveau logique, comme on peut le voir dans les exemples présentés en introduction.

Dans nos algorithmes nous considérerons évidemment que le but à obtenir n'est pas déjà connu (ni vrai ni faux).

La première idée venant à l'esprit pour calculer une question est d'utiliser un algorithme de type chaînage arrière en profondeur d'abord avec retour arrière (backtracking) de la manière suivante: Si le but initial est l'atome x, on s'intéresse alors aux règles qui concluent sur x et sur $\neg$x. On part de l'atome à obtenir qui constitue le but initial. S'il existe (point de choix) une règle ayant comme conclusion un littéral constitué de cet atome, on prend comme nouveau but l'un des atomes utilisé dans les prémisses (point de choix) de cette règle et on recommence. Si on tombe sur un fait inconnu demandable on questionne sur ce fait sinon on retourne au précédent point de choix.

On obtient alors l'algorithme suivant, utilisé dans de très nombreux systèmes experts:

Algorithme Calcul-naïf

```
Soit Ato l'atome initial demandé par l'utilisateur

Calcul_question(Ato)
     Si Ato n'est pas (inconnu et demandable)
          Si il existe une règle ayant comme conclusion un
           littéral d'atome associé Ato
               Choisir une telle règle. (point de choix
               Sélectionner l'un des atomes inconnus X utilisé
               dans les prémisses de cette règle.

               Calcul_question(X).
          Sinon
               Revenir au précédent point de choix.
          Fsi
     Fsi
Fin_calcul_question
```

Exemple 4

$c \rightarrow \neg a$	MT=∅ But: a
$\neg e \rightarrow a$	a dépend des atomes c ou e, c dépend de l'atome
$\neg d \rightarrow c$	d, e dépend de f. comme d et f sont inconnus
$f \rightarrow \neg e$	demandables on questionne sur l'un des deux.

Cet algorithme consiste donc à parcourir le graphe de dépendances des atomes du programme. Ses dérivés sont souvent utilisés dans les systèmes de démonstration automatique mais nous verrons que cet algorithme est non seulement inefficace mais incorrect dans le cas des systèmes experts utilisant des bases avec négations (bien qu'il soit encore largement utilisé) car il amène des questions qui ne sont ni intelligentes ni pertinentes. Cet algorithme initial nous permettra cependant de montrer sur quelques exemples les erreurs courantes et les modifications à apporter pour obtenir un fonctionnement correct.

1.2. Cheminement sur les littéraux.

L'algorithme calcul_naïf défini précédemment, bien que séduisant par sa simplicité n'est pas du tout adapté aux bases avec négations. Il s'intéresse à une règle dès que l'atome recherché est utilisé en conclusion de cette règle (sous entendu, si une règle conclut sur l'atome

correspondant elle peut amener des questions intéressantes). Malheureusement dès que ce type de cheminement est utilisé sur une base contenant des négations, on obtient des questions qui ne sont pas pertinentes comme pour l'exemple suivant.

Exemple 5

¬b → a	MT=∅ But: a
c → b	La seule question intelligente est e.
d → ¬c	La question d, trouvée par Calcul_naïf n'est
e → ¬b	même pas pertinente.

Comme seul le chaînage avant permet d'enrichir la mémoire de travail et que ce chaînage avant déduit des littéraux, il est évident que le calcul de la question en chaînage arrière ne doit pas se faire en cheminant d'atomes en atomes mais bien de littéraux en littéraux. Le but initial doit alors être un littéral. Nous appellerons ce type d'algorithmes un interpréteur à simple tentative [MD 89a].

Définition.
On appelle **interpréteur à simple tentative** tout algorithme fonctionnant de la manière suivante:
L'algorithme de calcul d'une question est lancé sur un littéral et le système essaie de calculer une liste de questions permettant de prouver ce littéral en cheminant de littéraux en littéraux. S'il n'y arrive pas le littéral est non déductible (ce qui ne veut pas dire que sa négation soit vraie mais simplement que ce littéral n'est pas satisfait). Ce type d'interpréteur est en fait très naturel puisque c'est l'extension directe de l'interpréteur utilisé pour des bases sans négations.

Mais nous avons précisé à la définition du chaînage mixte qu'il était beaucoup plus pratique pour l'utilisateur de lancer le calcul de la question sur un atome, le système devant essayer de déduire la valeur de cet atome. L'interpréteur à simple tentative ne peut donc pas être utilisé de manière brute. On ajoute alors un niveau supplémentaire à cet interpréteur de manière à pouvoir lancer notre système sur un atome et à en déduire sa valeur. C'est ce que nous appellerons un interpréteur à double tentative [MD 89a].

On appelle **interpréteur à double tentative** tout algorithme fonctionnant de la manière suivante:
L'algorithme de calcul d'une question est lancé sur un atome et le système essaie de déduire la valeur de cet atome. S'il n'arrive ni à montrer que cet atome est vrai, ni à montrer qu'il est faux, l'atome est alors inconnu. Pour cela il peut s'y prendre de trois façons différentes:

.Essayer de prouver le vrai et s'il n'y parvient pas essayer de prouver le faux.

.Essayer de prouver le faux et s'il n'y parvient pas essayer de prouver le vrai.

.Essayer les deux possibilités simultanément. Dans ce cas les questions qui permettent de déduire le vrai et les questions qui permettent de déduire le faux peuvent être mélangées. Ceci permet alors d'optimiser efficacement la question à poser. Il ne serait en effet pas très intéressant d'optimiser le calcul du vrai puis ensuite seulement d'optimiser le calcul du faux. La meilleure question doit être choisie parmi celles qui permettent de déduire le vrai et le faux simultanément.

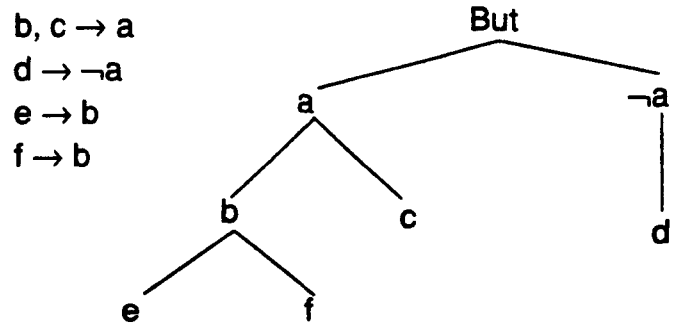
Plusieurs techniques peuvent être utilisées selon les heuristiques souhaitées, nous n'en présentons que deux:

Soit x l'atome demandé par l'utilisateur.

- Calculer les questions permettant de prouver x, calculer les questions permettant de prouver ¬x puis faire la réunion des deux listes obtenues. On a alors un niveau supplémentaire à l'appel de l'interpréteur à simple tentative.

- Ajouter deux règles de la forme x → but et ¬x → but où but est un but fictif sur lequel on lance le calcul des questions. On utilise alors directement un interpréteur à simple tentative lancé sur le but fictif.

Ces deux méthodes sont absolument équivalentes. Dans les implémentations nous conseillons la première méthode qui n'ajoute aucune règle à la base, mais sur les exemples traités nous utiliserons la deuxième méthode qui a l'avantage d'être facilement représentable par un arbre Et/Ou.



Le but étant a, la première question calculée est e.

Attention, même si l'on s'intéresse à la fois aux règles qui concluent sur x et ¬x, x étant le but demandé, il n'en est pas de même pour les autres littéraux. Ce processus ne doit donc pas être répété récursivement sous peine d'obtenir des questions qui ne sont pas pertinentes voir même stupides comme dans l'exemple 5. On obtient alors l'algorithme suivant:

```
Algorithme Calcul_de_base

Soit Ato l'atome initial demandé par l'utilisateur

Ajouter deux règles de la forme Ato→but et ¬Ato→but
Le but initial est alors but, et on lance Calcul_question(But).

Calcul_question(Litt)
     Si Litt n'est pas (inconnu et demandable)
          Si il existe une règle ayant Litt comme conclusion
               Choisir une telle règle. (point de choix)
               Sélectionner l'un des littéraux inconnus X
               utilisé dans les prémisses de cette règle.

               Calcul_question(X).
          Sinon
               Revenir au précédent point de choix.
          Fsi
     Fsi
Fin_calcul_question
```

Il faut remarquer que c'est très certainement à cause de la confusion entre un interpréteur à simple tentative et un interpréteur à double tentative que l'on trouve de nombreux systèmes cheminant d'atome en atome. Le but initial étant (pour des raisons de commodité) un atome il faut s'intéresser aux règles qui concluent sur cet atome ainsi qu'à celles qui concluent sur sa négation. Les concepteurs de systèmes ont souvent l'impression (à tort comme le montre l'exemple 5) que cette procédure doit être récursive, ce qui nous l'avons vu n'est pas le cas puisqu'on chemine ensuite de littéral en littéral.

1.3. Cohérence avec la mémoire de travail.

En cours de calcul de la question on développe un littéral dès qu'une règle possède ce littéral comme conclusion. Or ce littéral peut entrer en contradiction avec ce qui est déjà présent dans la mémoire de travail et dans ce cas, la liste de questions obtenue ne sera bien évidemment pas pertinente car si elle est satisfaite une contradiction se produira.

Exemple 6

	MT={¬d} But: a
b → a	e et f sont les seules questions intelligentes
c, d → b	à cet instant. La question c, trouvée par
e, f → b	Calcul_de_base n'est pas une question
	pertinente puisque d est faux.

Exemple 7

	MT={¬b} But: a
b → a	d est la seule question intelligente à cet
c → b	instant.La question c, trouvée par
d → a	Calcul_de_base n'est pas une question
	pertinente puisque b est déjà faux.

Dans l'exemple 6 la contradiction s'effectue sur l'une des questions à poser, tandis que dans l'exemple7 c'est un fait intermédiaire qui crée la contradiction. Le parcours de l'arbre Et/Ou n'est donc pas satisfaisant. Il est nécessaire de gérer des listes de littéraux (à la manière des listes de buts dans Prolog) et de vérifier à chaque développement que la liste est cohérente avec MT, c'est-à-dire qu'aucun littéral de la nouvelle liste n'entre en contradiction (opposé présent) avec MT. On cherche donc à calculer non plus une seule question comme dans les algorithmes précédents mais une liste de questions.

Algorithme Calcul_ldb (liste de base)

```
Soit Ato, l'atome initial demandé par l'utilisateur
Ajouter deux règles de la forme Ato→but et ¬Ato→but
La liste initiale 'Liste' est constituée de (but)

Calcul_liste(Liste)
     Tant qu'un des littéraux de la liste est non demandable

          Si il existe une règle ayant ce littéral en conclusion
               Choisir une telle règle. (point de choix)
               Remplacer dans la liste, le littéral par les
               prémisses de la règle choisie en supprimant les
               littéraux déjà présents dans la mémoire de
               travail.
```

```
                Si l'un des littéraux de la liste est en
                contradiction avec MT
                     Revenir au précédent point de choix ayant
                     des règles en suspens et choisir l'une de
                     ces règles.
                Fsi
           Sinon
                Revenir au précédent point de choix.
           Fsi
      Ftq

      Soit L la liste obtenue.
      Calcul_liste(L).
Fin_calcul_liste
```

Définitions.

Soit L un ensemble de littéraux, on appelle **dérivé d'ordre 1 de L** tout ensemble de littéraux obtenu en remplaçant chaque littéral de L qui apparaît au moins une fois en conclusion d'une règle par les prémisses de cette règle (pour un ensemble donné L il existe en général plusieurs dérivés d'ordre 1 de L).

Par récurrence pour i>1 on appelle **dérivé d'ordre i de L** tout ensemble obtenu en prenant un dérivé d'ordre 1 d'un dérivé d'ordre i-1.

On appelle **dérivé d'ordre $_\infty$** tout ensemble dérivé d'ordre i qui n'admet que lui-même comme dérivé d'ordre 1 (ou en d'autres termes quand plus aucun littéral de L n'apparaît en conclusion de règle.

Proposition 1.

Si L est une liste de questions pertinentes associée à x alors il existe un dérivé d'ordre ∞ de (x) contenant L.

Démonstration.

Si L est une liste de questions pertinentes associée à x c'est qu'il existe une déduction de x à partir de L. Il existe donc une dérivation de (x) contenant au moins tous les littéraux de L. On a donc L inclus dans un dérivé d'ordre $_\infty$ de (x).♦

Proposition 2.

Soit x un littéral. Si il existe un littéral y appartenant à la mémoire de travail tel que $\neg$y appartient à un dérivé d'ordre i de (x) alors cette dérivation n'amènera pas de listes de questions pertinentes.

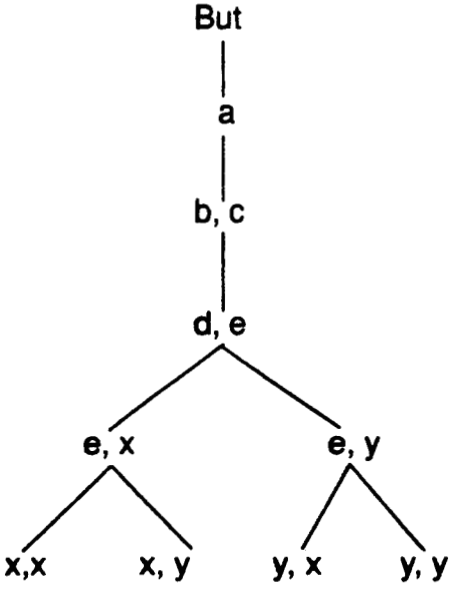
Démonstration.
Si il existe un littéral y appartenant à la mémoire de travail tel que ¬y appartient à un dérivé d'ordre i de (x) alors toute liste obtenue par dérivation de cette liste permettra de déduire ¬y. Donc une contradiction sera déclenchée et le but initial ne sera jamais prouvé.♦

1.4. Calcul de listes minimales de questions.

En développant des listes de questions en largeur d'abord, en cheminant de littéraux en littéraux et en vérifiant à chaque fois la cohérence avec la mémoire de travail, on est certain de n'obtenir que des questions qui font avancer la déduction vers le but recherché. Mais il se peut que dans la liste obtenue, certains littéraux permettent d'effectuer des déductions identiques. Dans ce cas, bien que chaque littéral de la liste fasse avancer le raisonnement, nous n'obtenons pas de liste minimale de questions.

Exemple 8

r1 b, c → a	MT=∅ But: a
r2 d → b	Les listes de questions (x,y) et (y,x) calculées
r3 e → d	par Calcul_ldb ne sont pas pertinentes (car elles
r4 e → c	ne sont pas minimales).Les seules listes de
r5 x → e	questions pertinentes sont (x) avec R={r1,r2,r3,
r6 y → e	r4,r5} et (y)avec R={r1,r2,r3,r4,r6}



Pour obtenir une liste minimale de questions il faut alors à chaque étape vérifier que chaque littéral de la liste en cours n'a pas déjà été développé précédemment. Il est donc nécessaire de garder à chaque instant la liste des ancêtres du but recherché. Si un littéral appartient à la liste

de ses ancêtres, on le supprime de la liste courante. Il en va de même s'il figure plusieurs fois dans la liste courante. En effet soit, comme dans l'exemple, e le littéral de la liste courante présent dans les ancêtres. On est certain que les littéraux qui permettent de déduire e sont présents dans la liste puisqu'il a déjà été développé. e peut donc être supprimé de la liste en cours de développement.

Remarque.
Le chaînage mixte effectuant un cycle question-saturation, le fait que la liste de questions obtenue ne soit pas minimale n'est pas toujours gênant. En effet si deux littéraux (x et y) de la liste permettent d'effectuer la même déduction e c'est que e a été développé deux fois. Comme les règles sont toutes selectionnées dans le même ordre, le 1er développement du 1er e a généré x de même que le 1er développement du 2nd e. Les derniers développements de chaque e ont généné deux fois y. Donc si x est vrai, e est déduit par la saturation et au prochain calcul y n'est plus calculé. Si x est faux, les listes intermédiaires contenant x ne sont plus calculées et seule la liste contenant deux y est calculée. C'est donc bien y qui est calculé.

Il est néanmoins intéressant d'obtenir une liste de questions minimales pour au moins deux raisons. La première vient de la nécessaire conformité à la définition des listes de questions pertinentes et intelligentes que nous nous sommes imposés, la seconde pour pouvoir ensuite dans le niveau heuristique comparer plusieurs listes de questions.

On obtient alors l'algorithme suivant:

Algorithme Calcul_lm (liste minimale)

```
Soit Ato, l'atome initial demandé par l'utilisateur
Ajouter deux règles de la forme Ato→but et ¬Ato→but
La liste initiale 'Liste'et la liste des ancêtres Anc sont cons-
tituées toutes deux de (but)

Calcul_liste(Liste,Anc)
     Tant qu'un des littéraux de la liste est non demandable

          Si il existe une règle ayant ce littéral en conclusion
               Choisir une telle règle. (point de choix)
               Remplacer dans la liste, le littéral par les
               prémisses de la règle choisie en supprimant les
               littéraux déjà présents dans la mémoire de
               travail.

               Si l'un des littéraux de la liste est en
               contradiction avec la mémoire de travail
                    Revenir au précédent point de choix.
```

```
                Fsi

                Si des littéraux de la liste sont en double ou
                sont déjà présents dans la liste Anc.
                     Supprimer ces littéraux
                Fsi
           Sinon
                Revenir au précédent point de choix.
           Fsi
      Ftq

      Soit L la liste obtenue.
      Ajouter L à Anc pour obtenir A.
      Calcul_question(L,A).
Fin_calcul_liste
```

Proposition 3.
Si une liste L est abandonnée alors cette liste n'est pas minimale.

Démonstration.
Si une coupure se produit c'est qu'il existe un littéral x appartenant à une liste calculée précedemment en n étapes et tel que x appartient à L. x a donc déjà été développé précédemment puisque tous les littéraux d'une liste sont développés en largeur. Soit D un dérivé d'ordre n de (x) on a alors D qui est inclus dans L puisque x a été dérivé. Comme nos bases sont sans cycle, x ne dépend pas de lui-même donc x n'appartient pas à D. Si on développe x à nouveau, on obtient alors une nouvelle fois les dérivés d'ordre n de (x), donc les listes obtenues ne seront pas minimales puisqu'elles permettront d'obtenir x de plusieurs manières différentes.♦

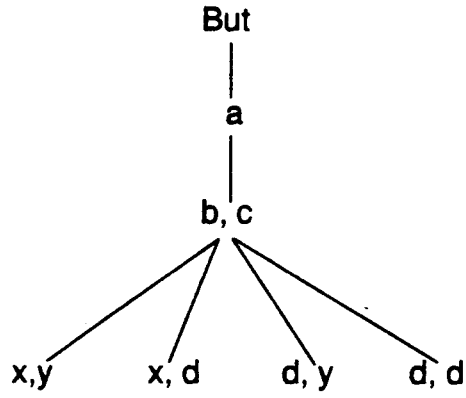
Proposition 4.
Si la liste en cours L est non minimale alors cette liste est abandonnée.

Démonstration.
Si la liste L en cours est non minimale, c'est qu'il existe deux sous ensembles R1 et R2 de L tels que R1 et R2 permettent tous deux de déduire un même littéral x. Comme nous cheminons en arrière x a donc été développé deux fois, la première pour obtenir R1 et la seconde pour obtenir R2. Donc soit x figurait deux fois dans une même liste et donc une coupure se produit, soit l'un des deux x a été développé avant l'autre et une coupure se produit aussi.♦

Attention c'est ici que la distinction entre une question pertinente et une question intelligente prend tout son sens.

Exemple 9

r1 b, c → a	MT=∅ But: a
r2 x → b	(x,d) est une liste de questions pertinentes car
r3 d → b	avec R={r1,r2,r5} notre définition est satisfai-
r4 y → c	te, mais pas une liste de questions intelligentes
r5 d → c	car (d) permet de déduire a tout seul.



1.5. Cohérence de la déduction future.

Après vérification des points précédents nous obtenons une liste de questions à poser à l'utilisateur qui semble pertinente mais qui risque encore d'amener une contradiction après utilisation de la saturation par chaînage avant. En effet durant l'étape de saturation, le chaînage avant peut ajouter un littéral donné puis lors de la même saturation tenter d'ajouter son opposé. Dans ce cas il est inutile d'utiliser cette liste de questions puisqu'elle déclenchera ensuite une contradiction par chaînage avant.

Les exemples suivants sont de difficulté croissante et montrent bien que ce problème est complexe et coûteux.

Exemple 10

b → a	MT=∅ But: a
c,¬c → b	La seule liste de questions intelligentes est
x,y,z → b	(x,y,z) et non pas (c,¬c) trouvée par Calcul_Im
	qui engendre une contradiction et qui n'est donc
	même pas pertinente.

Exemple 11

b,c → a	MT=∅ But: a
d → b	La seule liste de questions intelligentes est
¬b → c	(x,y,z) et non pas (d,e) trouvée par Calcul_lm
e → ¬b	qui engendre une contradiction sur b et qui n'est
x,y,z → a	donc même pas pertinente.

Exemple 12

r1 b → a	MT=∅ But: a
r2 c → b	La seule liste de questions intelligentes est
r3 c → d	(x,y,z)et non pas (c) qui engendre une
r4 c → ¬d	contradiction sur d.(c) est par contre une
r5 x,y,z → b	question pertinente (avec R={r1,r2}).

Dans l'exemple 10 la contradiction s'effectue sur la liste de questions, dans l'exemple 11 ce sont des faits intermédiaires à la déduction (qui ne figurent jamais simultanément dans la même liste) qui créent la contradiction tandis que dans l'exemple 12 la contradiction résulte de règles annexes à la déduction du but qui ne seront utilisées que par le chaînage avant.

On peut définir trois niveaux de vérification de la cohérence.

Vérification complète.

Le seul moyen d'être certain que la question que nous allons poser ne créera pas de contradiction est de lancer un chaînage avant fictif en considérant acquis les littéraux de la liste de questions calculée par l'algorithme puis vérifier que le but à obtenir sera bien déduit. Ce chaînage avant est donc semblable au précédent hormis le fait qu'il ne doit pas ajouter de nouveaux littéraux dans MT mais dans une mémoire temporaire propre à ce calcul. La mémoire temporaire est plus simple que MT principale puisqu'elle n'a à gérer que des faits connus, donc pas de méta-valeurs. Un fait est alors satisfait s'il est présent dans la mémoire principale ou s'il est présent dans la mémoire temporaire.

Cette vérification très coûteuse pourrait être utilisée par l'expert durant la phase de mise au point de la base de connaissances puis supprimée en cours d'expertise pour accélérer les déductions. **Elle n'est pas nécessaire pour le calcul d'une question pertinente mais elle l'est par contre pour le calcul d'une question intelligente.**

Vérification partielle du cône de déduction.

Le cône de déduction est l'ensemble des règles utilisées par le chaînage arrière lancé sur le but. On vérifie à chaque développement que les nouveaux buts obtenus ne sont pas en contradiction avec la liste de leurs ancêtres ni avec eux-mêmes. Cette vérification permet de résoudre la plupart des cas comme les exemples 10 et 11. Si le calcul de listes minimales présenté dans le chapitre précédent est effectué, cette vérification est alors très facile à implémenter puisque la liste des ancêtres est déjà conservée. Nous verrons par la suite que pour le traitement des faits symboliques cette vérification est particulièrement utile notamment parce qu'elle permet de couper très rapidement des branches inutiles dans l'arbre de calcul de la question. **Cette vérification est nécessaire pour le calcul d'une question pertinente.**

Aucune vérification.

On peut supposer que la plupart du temps, si la base est correctement écrite, la question n'amènera pas une contradiction après saturation. Dans ce cas aucune vérification n'est nécessaire. Il en va de même pour certaines bases particulières comme les bases sans aucun littéral négatif (base bivaluée). Cette option peut être utilisée dans le cas de bases très particulières où l'on est certain qu'aucune contradiction en cours de déduction ne risque d'arriver.

Il est évident que la vérification de la cohérence de la déduction est très coûteuse. Elle est néanmoins nécessaire puisque c'est la seule qui donne le niveau maximal de correction de la question. Une contradiction dans une base est un signe de mauvaise analyse de la connaissance et ne devrait jamais se produire en situation réelle. Nous suggérons donc d'avoir cette vérification désactivable par une option. Elle sera alors utilisée par le cogniticien qui met au point la base et sera désactivée au cours d'une utilisation réelle.

1.6. Prise en compte du non déductible.

Jusqu'à présent nous ne nous sommes intéressés qu'aux informations positives au sens où elles permettent de déduire une valeur pour le but recherché. Nous pourrions aussi nous intéresser aux informations du genre "a n'est plus déductible", c'est-à-dire que plus aucune question pertinente n'existe pour déduire a. Nous ne nous intéresserons pas ici à ce problème. Notons simplement que la prise en compte du non déductible permet d'avoir un calcul plus complet de toutes les listes de questions possibles. Les notions de questions intelligentes et pertinentes ne prennent pas en compte cet aspect. On pourrait imaginer une définition considérant le fait qu'un atome donné peut très bien devenir non-déductible (ni vrai ni faux ne peuvent être prouvés) en fonction de certaines réponses de l'utilisateur. Dans ce cas une question simple (et peut-être plus simple que les autres) permettrait de déduire immédiatement que le but n'est plus déductible.

Exemple 13

b → a	MT=∅ But: a
c → ¬b	(c) pourrait être prise en compte , mais selon
d,e → b	nos définitions seule (d,e) est pertinente.

Dans l'exemple précédent, nous voyons que si c est vrai a n'est plus déductible. Cette question est donc à prendre en compte puisqu'elle permet de déduire de la connaissance sur a.

Il est important de remarquer que les systèmes qui travaillent en cheminant d'atome en atome donnent l'impression de résoudre ce problème. Il n'en est évidemment rien. Ces systèmes sont incorrects du point de vue logique, nous l'avons vu sur l'exemple 5, tandis que les systèmes qui travaillent sur des littéraux sont corrects d'un point de vue logique: ils ne calculent peut-être pas les listes de questions prouvant la non-déductibilité comme dans l'exemple 13 mais les listes calculées sont toujours pertinentes et toutes les listes pertinentes sont calculées.

1.7. Algorithmes obtenus.

Le premier algorithme proposé permet de calculer une question pertinente conforme à la définition proposée. Il prend en compte toutes les évolutions de l'algorithme naïf présentées dans ce chapitre. Le second permet de calculer une question intelligente. Il est construit comme un sur-ensemble du précédent.

A. Calcul d'une question pertinente.

La liste initiale est constituée par le but à obtenir.

Tant qu'un des littéraux de la liste n'est pas inconnu et demandable et qu'il existe au moins une règle avec ce littéral comme conclusion. Choisir l'une de ces règles (règles candidates) (point de choix). Remplacer dans la liste ce littéral par les prémisses de la règle choisie en supprimant les littéraux déjà présents dans MT et les doublons. Si l'un des littéraux est en contradiction avec MT abandonner cette liste et revenir au précédent point de choix. Si l'un des littéraux est en contradiction avec ses ancêtres abandonner cette liste et revenir au précédent point de choix. Si l'un des littéraux de la liste en cours est déjà présent dans ses ancêtres supprimer ce littéral. Si aucune règle ne possède ce littéral comme conclusion abandonner cette liste et revenir au précédent point de choix.

Algorithme Calcul_complet_pertinent.

```
Soit x, l'atome initial demandé par l'utilisateur
Ajouter deux règles de la forme x→but et ¬x→but
La liste initiale 'Liste'et la liste des ancêtres Anc sont cons-
tituées toutes deux de (but)

Calcul_liste_pertinente(Liste,Anc)
     Tant qu'un des littéraux de la liste est non demandable

          Si il existe une règle ayant ce littéral en conclusion
               Choisir une telle règle. (point de choix)
               Remplacer dans la liste, le littéral par les
               prémisses de la règle choisie en supprimant les
               littéraux déjà présents dans la mémoire de
               travail.
               Si l'un des littéraux de la liste est en
               contradiction avec MT
                    Revenir au précédent point de choix.
               Fsi

               Si des littéraux de la liste sont en double ou
               sont déjà présents dans Anc.
                    Supprimer ces littéraux
               Fsi

               Si l'un des littéraux de la liste est en
               contradiction avec Anc
                    Revenir au précédent point de choix.
               Fsi

          Sinon
               Revenir au précédent point de choix.
          Fsi
     Ftq

     Soit L la liste obtenue.
     Ajouter L à Anc pour obtenir A.
     Calcul_liste_pertinente(L,A).
Fin_calcul_liste_pertinente
```

Il est à remarquer que comme tout autre chaînage arrière, il peut boucler si la base contient des cycles, mais nous avons déjà précisé que nos bases sont sans cycle. Si cela n'était pas le cas il serait nécessaire de prévoir un dispositif anti-bouclage dans les différents algorithmes ou encore un nombre maximum d'inférences à ne pas dépasser. Il est de plus facile dans le cas propositionnel de vérifier que la base ne contient pas de cycle dès la compilation ou encore de modifier cette base de manière à supprimer les cycles.

B. Calcul d'une question intelligente.

Pour obtenir une question intelligente il est nécessaire de calculer toutes les listes de questions pertinentes et de vérifier pour chacune d'elles qu'elle ne crée pas de contradiction sur la base totale. On vérifie donc sa cohérence par l'une des méthodes exposées précédemment. Puis on supprime les listes qui contiennent d'autres listes pour assurer le critère de minimalité.

Algorithme Calcul_complet_intelligent.

```
Soit x, l'atome initial demandé par l'utilisateur
Ajouter deux règles de la forme x→but et ¬x→but
La liste initiale 'Liste'et la liste des ancêtres Anc sont cons-
tituées toutes deux de (but)

     Calculer l'ensemble E de listes L telles que
          L=Calcul_liste_pertinente(Liste,Anc)
          et tel que But∈Chavt(Br∪L)
     Fin_calcul.

     Supprimer dans E les listes subsumées par d'autres listes.
```

Théorème 32.
Soit x un atome, l'algorithme Calcul_complet_pertinent calcule une liste de questions pertinentes pour x.

Démonstration.
Par la proposition 1 toutes les listes de questions sont calculées. La proposition 2 garantit qu'aucun littéral de la déduction ne sera en contradiction avec la mémoire de travail (MT).La vérification complète de la cohérence garantit qu'en cours de déduction aucun littéral et sa conclusion ne seront ajoutés à MT puisque la liste permet effectivement de déduire le but recherché. La proposition 3 quant à elle nous assure que la liste de questions sera minimale.♦

Théorème 33.
Soit x un atome, si L est une liste de questions pertinentes pour x alors L peut être calculée par Calcul_complet_pertinent.

Démonstration.
Toutes les listes de questions dérivées de x sont calculées par la proposition 1. La proposition 2 nous montre que si une liste est abandonnée c'est qu'elle n'est pas pertinente puisqu'elle créera une contradiction avec la mémoire de travail (MT) si elle est utilisée. La proposition 4 nous assure qu'aucune liste de questions minimales n'est supprimée. Enfin la vérification complète de la cohérence de la déduction ne supprime que les listes ne permettant pas de déduire le but recherché. Par la possibilité de retour aux différents points de choix (backtracking) à chaque abandon d'une liste on est alors assuré que si L est une liste e questions pertinentes alors L sera calculée par Calcul_complet.♦

Théorème 34.
Soit x un atome, l'algorithme Calcul_complet_intelligent calcule une liste de questions intelligentes pour x.

Démonstration.
Par les deux théorèmes précédents nous avons montré que calcul_complet_pertinent permet de calculer liste de questions pertinentes. Or une question pertinente n'est cohérente que dans son cône de déduction. Pour qu'elle soit intelligente il faut qu'elle soit cohérente avec le reste de la base. Ce qui est assuré par le chaînage avant temporaire utilisé après chaque calcul. La minimalité de la liste obtenue est assurée par la suppression des listes subsumées par d'autres en fin d'algorithme. Les listes de questions calculées sont donc bien des listes de questions intelligentes.♦

Théorème 35.
Soit x un atome, si L est une liste de questions intelligentes pour x alors L peut être calculée par Calcul_complet_intelligent.

Démonstration.
Evidente. Comme toute liste de questions intelligentes est une liste de questions pertinentes il en résulte immédiatement par le théorème 33 que toute liste de questions intelligentes est calculer par calcul_complet_intelligent♦

2. Le niveau heuristique

Après avoir calculé les listes de questions, il est maintenant nécessaire de savoir quelle question poser et pour cela quelle liste de questions choisir.

2.1. Calcul des Listes au fur et à mesure.

Les heuristiques présentées dans cette catégorie ne nécessitent pas le calcul de toutes les listes de questions avant de choisir la liste à utiliser, ce qui leur permet d'être très rapides. Elles ont l'avantage d'être facilement compréhensibles par l'utilisateur (les calculs de choix sont simples) et très facilement implémentables dans un algorithme de type chaînage arrière avec retour arrière (backtracking). Par convention c'est le premier littéral de la liste choisie qui sera demandé.

Si ces méthodes sont simples et rapides elles n'en sont pas moins intéressantes. Il est en effet facile pour l'utilisateur de comprendre comment la connaissance sera utilisée, la stratégie pouvant être calculée de tête. Pour des méthodes plus complexes le calcul ne pourra plus se faire aussi facilement.

2.1.1. Règles dans l'ordre de la base.

Dans certains cas la question la plus pertinente n'est pas forcément nécessaire ni souhaitable, n'importe quelle question pertinente pouvant suffire. On arrête alors l'algorithme utilisé dès qu'une question est trouvée. Les règles sont sélectionnées dans l'ordre d'écriture des règles dans la base. Un chaînage arrière en profondeur d'abord avec backtracking s'arrêtera alors à la première liste calculée ce qui permet d'obtenir une question très rapidement.

Exemple 14

b → a	MT=∅ But: a
e →b	parmi les listes (f,g,h) et (c,d)
c,d →b	(f,g,h) est choisie car dans l'ordre des règles
f,g,h → e	c'est e qui est évalué en premier.

2.1.2. Coefficients sur les règles.

Si plusieurs règles ont la même conclusion, on attribue initialement un numéro de préférence à chaque règle. On choisit alors celle dont le numéro est le plus faible (ou le plus fort). Cette méthode est très souvent utilisée car elle donne à l'expert le sentiment de garder facilement le contrôle sur la déduction.

Cette heuristique est en fait une généralisation de la méthode précédente. Pour les règles dans l'ordre cela correspond à affecter les coefficients à chaque règle selon leur ordre dans la base. Cette heuristique généralise aussi les algorithmes qui préfèrent d'abord les règles avec le moins de prémisses, puisque cela revient à associer pour chaque règle son nombre de prémisses

comme coefficient associé.

D'autres variantes existent encore, comme préférer les règles dont les informations sont les plus récemment acquises, les règles avec le plus de conclusions ou encore les règles avec le plus de prémisses car la conclusion est plus sûre (heuristique souvent utilisée avec les coefficients de vraissemblances) ... etc. Ces différents cas pouvant se ramener au cas précédent nous ne ferons que les citer.

Il est à noter que ces heuristiques peuvent être implémentées soit à la compilation par classement des règles, soit en modifiant le moteur d'inférences pour qu'il puisse choisir la règle à utiliser, soit en utilisant des méta-règles.

2.1.3. Règles avec le moins de prémisses inconnues.

Une autre heuristique très souvent utilisée consiste à préférer la règle qui possède le moins de prémisses inconnues dès que plusieurs règles sont candidates. On suppose alors que moins une règle a de prémisses plus on a de chances d'avoir une déduction rapide. Ce n'est évidemment qu'une heuristique mais elle donne généralement de bons résultats et est de ce fait souvent employée.

Exemple 15

	MT={f} But: a
b → a	parmi les listes (c,d) et (e)
c,d → b	c'est (e) qui est choisie car des deux règles qui
e,f → b	concluent sur b, la seconde a moins de prémisses
	inconnues

2.2. Calcul complet des listes de questions.

Dans les précédentes heuristiques aucune comparaison entre les différentes listes de questions possibles n'était effectuée. Or il peut s'avérer intéressant de choisir une liste plutôt qu'une autre en fonction de caractéristiques précises. Ceci nous amènera à proposer une méthode qui non seulement ordonne les listes de questions mais de plus ordonne les atomes de la liste choisie.

2.2.1. Choix de la liste la plus courte.

On peut supposer que tous les faits demandables ont la même probabilité d'être vrais ou faux. Dans ce cas plus la liste est courte et plus la chance d'obtenir le but recherché est grande. On préférera alors la liste qui contient le moins de littéraux et la question portera sur l'un des

littéraux de cette liste.

Exemple 16

b,c → a	MT=∅ But: a
d,e,f → a	a peut être déduit en une seule question: f
e → d	alors que b et c déduisent a en deux questions.
f → e	

Cet exemple montre de plus la nécessité de bien gérer des listes de questions et non pas seulement le nombre de littéraux qui la constitue. On serait alors amené à considérer que a nécessite trois questions si on utilise la deuxième règle, ce qui est faux.

2.2.2. Choix de l'atome le plus utilisé.

Quand plusieurs listes de questions sont possibles, il se peut qu'un atome soit présent dans plusieurs de ces listes. Il apporte alors plus d'informations que les autres puisque en particulier s'il n'est pas satisfait il invalide plusieurs listes à la fois.

Exemple 17

b → a	MT=∅ But: a
c → a	Les listes de questions obtenues sont (d,x) et
d,x → b	(e,x). La question x apporte plus d'informations
e,x → c	que d ou e.

Attention. C'est un atome que l'on demande à l'utilisateur, c'est donc le nombre de fois que l'atome apparaît (et non le littéral) qui doit être pris en compte.

Cette heuristique peut évidemment être couplée avec la précédente. Si un littéral est présent plusieurs fois on questionne sur ce littéral, sinon on utilise la liste la plus courte.

2.2.3. Coefficients de satisfaisabilité.

Jusqu'à maintenant nous faisions l'hypothèse que tous les faits demandables étaient équiprobables, c'est-à-dire qu'ils avaient autant de chance d'être "Vrai" ou "Faux". Or en réalité ceci est souvent inexact.

Prenons le cas d'un système d'expertise médicale. Des faits demandables du type symptômes

peuvent être 'exposition prolongée au soleil' ou encore 'nez qui coule'. Or si nous sommes en été les chances d'avoir une réponse affirmative à la question 'nez qui coule' sont très faibles. Si ces deux faits sont en concurrence, le système a tout intérêt à demander 'exposition prolongée au soleil'.

On associe alors à chaque littéral un coefficient de satisfaisabilité correspondant à la chance de satisfaire ce littéral par une question à l'utilisateur. Plus ce coefficient est élevé, plus la chance est grande. Pour des raisons de commodité nous considérerons que nos coefficients sont des réels compris entre 0 et 1 (Ce qui correspond à un pourcentage). Si l'on considère comme négligeable la possibilité d'avoir la réponse 'Je ne sais pas' à la question posée, le coefficient d'un littéral est alors égal au complémentaire à 1 du coefficient de son opposé. Par exemple si A a pour coefficient associé 0.2 (20%) $\neg$A a pour coefficient associé 0.8 (80%). Si le pourcentage de chances d'obtenir 'Je ne sais pas n'est pas négligeable il faut alors s'assurer que la somme des coefficients de A et de $\neg$A est inférieure à 1. Le coefficient de 'Je ne sais pas' est alors le complément à 1 de la somme des coefficients pour A et pour $\neg$A.

La note associée à une liste de questions est alors le produit des coefficients de chaque littéral de cette liste.

On choisit donc la liste ayant la plus grande note et dans cette liste on questionne sur le littéral ayant le plus fort coefficient

Remarque.
Si nous considérons que la chance d'obtenir 'je ne sais pas' à une question est infime nous pouvons alors dans le cadre d'une réalisation, ne préciser que les coefficients des littéraux positifs à la définition des faits demandables. A chaque atome demandable est associé un coefficient de demandabilité correspondant au littéral positif, le coefficient du littéral négatif étant calculé automatiquement par complément à 1. C'est ce que nous utiliserons ici pour ne pas surcharger nos exemples.

Exemple 18

	MT=∅ But: a
b → a	coefficients: c 50%, d 20%, e 70%
c → a	La liste (¬d,e) a un coefficient de 56% (80*70) et (c)
¬d,e → b	de 50%.On choisit donc (¬d,e) comme liste puis e qui a le coefficient le plus petit de cette liste (car ¬d a un coeff de 80%).

Cette méthode permet non seulement d'ordonner les listes de questions mais aussi d'ordonner les littéraux dans chaque liste. Elle permet de plus d'obtenir automatiquement la liste de questions qui possède le moins de littéraux dans le cas où ils ont tous le même coefficient puisque l'on effectue le produit de coefficients compris entre 0 et 1. Moins il y a de littéraux

plus le coefficient de la liste est grand et donc plus elle a de chances d'être utilisée.

2.2.4. Cumul des méthodes précédentes.

On le comprend après ces exemples, la liste la plus petite, le littéral le plus représenté et les coefficients de demandabilité sont trois options particulièrement intéressantes qui nécessitent d'être prises en compte simultanément.

Nous présentons ici une méthode permettant de vérifier ces trois options simultanément par un calcul arithmétique relativement simple.

Poids d'un littéral.
On détermine alors pour chaque littéral X figurant dans les listes de questions un poids compris entre 0 et 1 noté p(X) prenant en compte:

- la longueur de la liste minimale le contenant, notée l(X).

- le nombre de listes le contenant, noté nb(X).

- le coefficient de demandabilité de ce littéral, noté c(X).

En partant du principe que plus le poids est élevé plus la question sur ce littéral est intéressante nous obtenons alors la formule suivante pour calculer le poids d'un littéral X.

$$p(X) = c(X)^{l(X) / nb(X)}$$

Conformité de la formule.
Par hypothèse plus le poids est grand plus le littéral a de chances d'être demandé. Comme le coefficient est compris entre 0 et 1, plus l'exposant est petit plus le poids obtenu est grand. Or plus la liste est courte, plus l'exposant est petit, donc plus le littéral a de chances d'être demandé. De même plus le littéral apparaît de fois, plus l'exposant est petit et donc plus le littéral a de chances d'être demandé.

La formule est donc bien conforme à nos exigences (Il est néanmoins très facile de modifier la formule pour avantager l'un des trois paramètres).

Exemple 19

b,c,d → a	MT=∅ But: a
b,e,f → ¬a	Tous les coefficients sont à 50%
g,h → a	C'est b qui est demandé.

Dans cet exemple il y a trois listes de questions possibles: (b,c,d) (b,e,f) et (g,h).

b apparaît deux fois et la liste la plus courte est constituée de 3 éléments, le poids associé à b est alors $(1/2)^{(3/2)}$ soit 35,35%

g (comme h) par contre n'apparaît qu'une fois mais dans une liste de deux éléments, le poids associé est alors $(1/2)^{(2)}$, Soit 25%.

Quant à c (comme d,e,et f) il apparaît une fois dans une liste de trois éléments, le poids associé est alors $(1/2)^{(3)}$ soit 12,5%. C'est donc b qui a le poids le plus important.

Note associée à un atome.

Après avoir calculé le poids associé à chaque littéral il nous faut maintenant tenir compte du fait qu'un littéral et sa négation peuvent êtres tous deux dans deux listes de buts distinctes.

Exemple 20

b,d → ¬a	MT=∅ But: a
¬c,¬b → a	Tous les coefficients sont à 50%
	C'est b qui est demandé.

Dans cet exemple si les coefficients sont tous les mêmes, la question la plus intéressante porte évidemment sur b. Pour prendre en compte cette nouvelle possibilité on calcule la note associée à chaque atome, ce qui déterminera la question à poser. Deux cas se présentent:

- Un littéral est présent dans les listes sans que son opposé le soit (c'est le cas de d et ¬c). La note de l'atome correspondant est égale au poids du littéral.

- Un littéral et son opposé sont présents dans les listes (c'est le cas de b). L'atome correspondant doit avoir une note supérieure au poids du littéral et au poids de son opposé tout en restant comprise entre 0 et 1. La formule que nous proposons, basée sur la somme probabiliste vérifie cette propriété:

$$n(X) = p(X) + p(\neg X) - p(X).p(\neg X)$$

Cette formule peut encore s'écrire $n(X) = 1 - (1-p(X)).(1-p(\neg X))$. C'est sous cette seconde forme que nous l'utiliserons par la suite pour les faits symboliques.

Dans l'exemple 20, si les coefficients sont tous à 50% on obtient:

$p(d)=p(\neg c)=p(b)=p(\neg b)=0.5^2=0.25$

$n(d)=n(c)=0.25$

n(b)= (0.25+0.25) - 0.0625 = 0.4375

Exemple 21

---

b,c,e → ¬a
b,d,¬e → a

Dans cet exemple, deux listes sont possibles: (b,c,e) et (b,c,¬e).

Tous les coefficients sont à 50%

$p(c)=p(d)=p(e)=p(\neg e)=0.5^3=0.125$
$p(b)=0.5^{(3/2)}=0.353$

n(c)=n(d)=0.125 n(b)=0.353
n(e)=0.125+0.353-(0.125*0.353)=0.234

C'est b qui est demandé.

Tous les coefficients sont à 50% sauf e qui est à 25%.

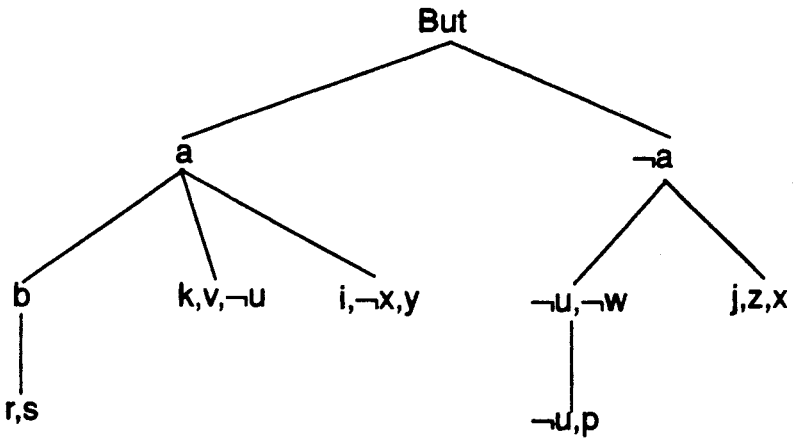
$p(c)=p(d)=0.5^3=0.125$
$p(b)=0.5^{(3/2)}=0.353$
$p(e)=0.25^3=0.015$
$p(\neg e)=0.75^3=0.421$

n(c)=n(d)=0.125
n(b)=0.353
n(e)=0.015+0.421-(0.015*0.421)=04296

C'est e qui est demandé.

Exemple Général.

$b \rightarrow a$
$t \rightarrow \neg b$
$r, s \rightarrow b$
$s \rightarrow c$
$c \rightarrow \neg b$
$k, v, \neg u \rightarrow a$
$\neg u, \neg w \rightarrow \neg a$
$p \rightarrow \neg w$
$i, \neg x, y \rightarrow a$
$j, z, x \rightarrow \neg a$



Le but demandé par l'utilisateur est l'atome u

Nous considérons que tous les faits demandables sont équiprobables (coefficient de 50%).

Si le système demande t, c'est qu'il fonctionne en cheminant sur les atomes, ce qui est une grave erreur. Si il demande r c'est qu'il ne vérifie pas la cohérence de la déduction car il lui faudrait ensuite s qui apporte une contradiction.

En choisissant la première question trouvée il doit demander k mais s'il optimise son calcul, la question la plus intéressante est alors u qui intervient deux fois et de plus dans la liste la plus courte.

u faux. Il essaie donc de déduire ¬a et demande alors p.

p faux. L'utilisation de ¬u pour prouver ¬a n'ayant pas fonctionné il essaie alors de prouver a et pose alors k.

k faux. Il reste deux listes de questions possibles qui utilisent toutes deux l'atome x et qui permettent de prouver a ou ¬a.

x vrai. On ne peut alors plus prouver a. On tente alors de prouver ¬a par les questions z et j. Si on répond vrai à ces deux questions, ¬a est prouvé et le système répond 'a est faux'.

IV. Extension à l'ordre 0+

1. Manipulations symboliques positives.

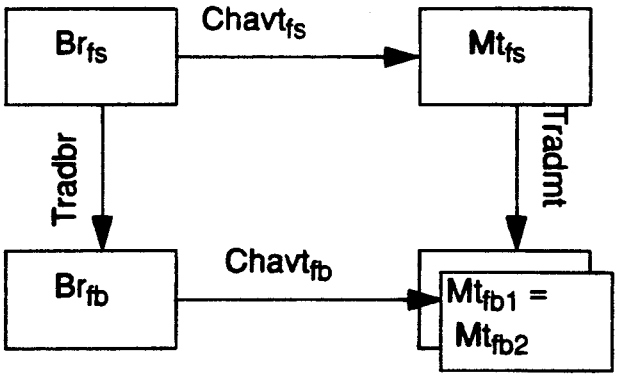
Il a été montré dans de nombreux cas l'utilité des faits symboliques [Del 87a] [IS 86]: concision dans l'écriture des règles, faits utilisés de différentes manières, factorisation de questions.

Des essais d'utilisation de ces faits symboliques ont notamment été effectués de manières très variées dans de nombreux systèmes experts, mais jusqu'ici aucune étude formelle n'a véritablement été effectuée.

La plus répandue de ces utilisations est d'autoriser uniquement les manipulations symboliques positives. D'autres systèmes permettent les manipulations symboliques négatives uniquement en condition de règle. Il est aussi possible d'autoriser les faits symboliques à la fois en condition et en conclusion de règle. Malheureusement ces différentes utilisations n'engendrent pas les mêmes contraintes de réalisation. Ce sont ces différences que nous allons tenter expliciter dans ce chapitre.

Il est nécessaire pour chaque cas de définir précisément ce que l'on calcule. Après avoir défini notre chaînage avant à trois valeurs sur des faits booléens, il nous paraît intéressant d'essayer de transformer les bases avec manipulations symboliques construites dans ce chapitre pour obtenir des bases avec uniquement des faits booléens. Un chaînage avant avec faits symboliques sera correct pour une base avec faits symboliques s'il calcule la même chose que le chaînage avant avec faits booléens sur la base transformée.

On peut illustrer ceci par la figure suivante:



Br_{fs}	Base de règles avec faits symboliques.
Br_{fb}	Base de règles avec uniquement des faits booléens.
Mt_{fs}	Mémoire de travail obtenue par $Chavt_{fs}$.
Mt_{fb1}	Mémoire de travail obtenue par $Chavt_{fb}$.
Mt_{fb2}	Mémoire de travail obtenue par $Trad_{mt}$.
$Chavt_{fs}$	Chaînage avant avec faits symboliques.
$Chavt_{fb}$	Chaînage avant avec uniquement des faits booléens.
$Trad_{br}$	Opération de transformation de Br_{fs} en Br_{fb}.
$Trad_{mt}$	Opération de transformation de Mt_{fs} en Mt_{fb}.

Définition.
Un chaînage avant $Chavt_{fs}$ sur une base avec des faits symboliques Br_{fs} est **correct** s'il existe une transformation $Trad_{br}$ de Br_{fs} en une base avec uniquement des faits booléens Br_{fb} telle que $Tradbr(Chavt_{fs}(Br_{fs}))=Chavt_{fb}(Trad_{br}(Br_{fb}))$

Le cas le plus simple d'utilisation de manipulations symboliques est sans doute de n'accepter que des manipulations symboliques positives dans une base de connaissances. La syntaxe s'étend alors de la manière suivante.

1.1. Syntaxe.

On reprend les définitions classiques d'ensembles d'atomes et de littéraux données pour les bases avec faits booléens et on étend la définition de bases de règles aux bases de règles avec manipulations symboliques.

On définit l'ensemble des **manipulations symboliques** que l'on note Ms comme étant l'ensemble des formules de la forme X='v' (manipulation symbolique positive) ou X<>'v' (manipulation symbolique négative. Equivalent à ¬ X='v').

La partie gauche d'une manipulation symbolique est appelée **fait symbolique**, tandis que la partie droite est appelée **valeur symbolique**.

On appelle **domaine de définition** d'un fait symbolique F, l'ensemble des valeurs symboliques v telles que F='v' ou F<>'v' soit présent dans la base de règles.

Nous partons du principe qu'un fait symbolique n'est affectable que dans son domaine de définition. Son domaine de définition peut être calculé par défaut de la manière exposée précédemment mais il est évident que l'utilisateur doit avoir la possibilité de le modifier de manière à rajouter des valeurs possibles s'il le faut. Ceci nous permet alors d'établir une sémantique logique aux différentes bases, ce qui ne serait pas possible avec des domaines infinis. On notera Dom(F) le domaine de définition d'un fait symbolique F. Il ne faut pas voir la définition du domaine comme une contrainte. Le définir permet d'établir des algorithmes plus simples mais il serait équivalent d'en faire le calcul dynamiquement comme dans certains

systèmes. Il ne restreint donc pas les possibilités du système. Après ces considérations la syntaxe autorisée est alors décrite par la grammaire ci dessous:

<base> ::= <regle> | <regle> <base>
<regle> ::= <condition> → <conclusion>
<condition> ::= <fait> , <condition> | <fait>
<conclusion> ::= <fait>
<fait> ::= <littéral> | <manip_symbolique>
<litteral> ::= <atome> | ¬ <atome>
<manip_symbolique>::= <atome> = '<valeur>'

1.2. Chaînage avant.

1.2.1. Sémantique.

On définit l'application de Br_{fs} en Br_{fb} de la manière suivante:

On remplace dans Br_{fs} toute manipulation symbolique positive de la forme X='v' par le littéral positif X_v.

Puis pour chaque fait symbolique Fi de $Fs(Br_{fs})$ et pour tous les couples de valeurs d'affectations Vj,Vk de Dom(Fi) tels que j≠k ajouter les règles Fi_Vj → ¬Fi_Vk et Fi_Vk → ¬Fi_Vj.

Exemple 22

b='1'→ a	sera transformée en	b_1 → a
c → b='2'		c → b_2
d → b='1'		d → b_1
		b_1 → ¬b_2
		b_2 → ¬b_1

Extension de la notion de cycle.

D'une manière rigoureuse il y aura cycle si une manipulation symbolique positive dépend d'elle-même, mais il est clair que si elle dépend d'une autre manipulation symbolique positive utilisant le même fait symbolique, une contradiction sera déclenchée. Nous considérerons donc qu'une base possède un cycle si il existe un fait symbolique dépendant de lui-même. Nous rappelons que les bases que nous traitons sont sans cycle.

1.2.2. Algorithme.

Appelons $Chavt_{fs1}$ le chaînage avant prenant en compte ce formalisme. Les modifications à apporter au chaînage avant classique pour les bases avec uniquement des faits booléens sont alors les suivantes:

Mémoire de travail.
Comme seules les manipulations symboliques positives peuvent être représentées, les modifications à apporter à la précédente mémoire de travail sont très minces. En plus du triplet (atome,meta_valeur,valeur) qui permet de représenter des littéraux on accepte les triplets de la forme (fait symbolique, méta_valeur, valeur symbolique) pour représenter les manipulations symboliques, donc le troisième élément du triplet peut contenir non seulement Vrai ou Faux mais aussi une valeur symbolique.

Satisfiabilité des prémisses.
Une manipulation symbolique positive du type F='v' sera satisfaite si F a pour valeur v dans la mémoire de travail.

Détection des contradictions.
Suivant le principe d'affectation unique, un fait symbolique affecté à une valeur ne peut plus prendre d'autre valeur. Il en résulte que toute tentative d'affectation d'une valeur à un fait symbolique déjà affecté à une autre valeur déclenchera une contradiction. Il en va de même pour toute tentative d'affectation d'une valeur n'appartenant pas au domaine de définition du fait concerné.

1.2.3. Correction de la déduction.

Théorème 36.
$Chavt_{fs1}$ est un chaînage avant correct pour les bases avec faits symboliques positifs.

Démonstration.
Montrons que tout littéral de Mt_{bf1} appartient à Mt_{bf2}.

Pour qu'une manipulation symbolique du type F='v' soit ajoutée à Mt_{bf2} il faut qu'elle figure en conclusion d'une règle dont les prémisses sont satisfaites. La règle correspondante obtenue par l'application définie précédemment peut donc elle aussi être déclenchée.

a) Le fait n'a pas encore été affecté.

Il est alors ajouté à Mt_{fs} puis sera traduit dans Mt_{fb2}, donc tout littéral de Mt_{fb1} est présent dans Mt_{fb2}

b) Le fait symbolique a déjà été affecté à une autre valeur.

Par définition de $Chavt_{fs}$ une contradiction est détectée. Or si une autre valeur a déjà été affectée c'est qu'une règle concluant sur cette manipulation symbolique s'est déclenchée et donc que la règle de Br_{fb} correspondante par l'application s'est déclenchée elle aussi. Or par définition de $Trad_{br}$, après avoir ajouté ce littéral, toutes les règles de la forme F_v → ¬F_v' se sont déclenchées. $Chavt_{fb}$ tente alors d'ajouter à la base un littéral dont l'opposé est déjà présent, une contradiction est alors détectée. Donc si $Chavt_{fs}$ détecte une contradiction, $Chavt_{fb}$ en détecte une aussi.♦

1.3. Chaînage mixte.

1.3.1. Interface utilisateur.

A. Faits demandables.

D'après la syntaxe utilisée, les faits de base sont maintenant des manipulations symboliques. C'est donc une manipulation symbolique et non simplement un fait qui devrait être demandable mais comme nous l'avons déjà dit pour les faits booléens il est plus pratique de déclarer demandable un fait symbolique. Il n'est donc pas possible d'avoir a='x' demandable et a='y' non demandable. Un fait symbolique sera demandable par défaut si il existe une manipulation symbolique de base qui l'utilise.

B. Lancement du chaînage mixte.

Pour un interpréteur à simple tentative il est clair qu'il sera lancé sur un but correspondant à une manipulation symbolique. Pour un interpréteur à double tentative on préférera lancer sur un fait symbolique de manière à pouvoir déduire la valeur qu'il peut prendre. Il est alors nécessaire de lancer le chaînage arrière sur chaque manipulation symbolique utilisant le fait demandé. Comme pour les faits booléens plusieurs méthodes sont possibles. Nous en proposons deux:

- Concaténer toutes les listes de listes de questions permettant de déduire chaque manipulation symbolique utilisant le but recherché.

- Ajouter pour chaque manipulation symbolique utilisant le fait demandé, une règle concluant sur un but fictif sur lequel sera lancé le chaînage arrière.

C. Question à l'utilisateur.

Comme pour les faits booléens où il n'était pas intéressant de questionner sur un littéral, il n'est ici pas très intéressant de questionner sur une manipulation symbolique. On préférera questionner l'utilisateur sur la valeur du fait utilisé dans cette manipulation. Il est alors nécessaire de vérifier que la valeur entrée par l'utilisateur appartient bien au domaine de définition du fait considéré.

1.3.2. Calcul de la question.

Après les considérations précédentes, le calcul de la question à poser se fait exactement de la même manière que pour les faits booléens. Le chaînage arrière qui calcule la question doit alors cheminer de manipulation symbolique en manipulation symbolique. On essaie alors d'appliquer toutes les règles qui ont le fait symbolique demandé en conclusion de règle.

Exemple 23

```
b='1' → a
c='1' → b='2'  MT=∅       But: a
d → c='2'      La question à poser est e mais surtout pas d
e → b='1'
```

On trouve très souvent des systèmes qui cheminent de fait symbolique en fait symbolique. La raison vient du fait qu'il est clair que la question doit être posée sur un fait symbolique et qu'il faut s'intéresser aux règles qui concluent sur toutes les manipulations symboliques utilisant ce fait. Les concepteurs de systèmes ont souvent l'impression (à tort comme le montre l'exempleprécédent) que cette procédure doit être récursive, ce qui nous l'avons vu n'est pas le cas.

Coefficients de satisfaisabilité.

Cette fois-ci les coefficients de satisfaisabilité sont associés à chaque manipulation symbolique. La somme de ces coefficients ne doit évidemment pas dépasser 1. Le coefficient de la valeur 'Je ne sais pas' est toujours calculé par complément à 1 de la somme des coefficients de chaque manipulation symbolique associée à ce fait. La note de l'atome associé est alors calculée par extension de la formule proposée pour les faits booléens de la manière suivante:

$$n(x) = 1 - (1-p(x='v_1')).(1-p(x='v_2'))...(1-p(x='v_n'))$$

Les v_n étant les différentes valeurs possibles que peut prendre x.

Remarque.
Un système ne traitant que des faits symboliques peut très bien traiter des faits booléens. Il suffit de remplacer chaque littéral positif x par x='v' et chaque littéral négatif ¬x par x='f'. C'est ce que nous ferons dans nos implémentations, l'interface utilisateur faisant en sorte que ce traitement soit transparent pour l'utilisateur.

2. Manipulations symboliques négatives uniquement en prémisses.

Il est parfois intéressant de pouvoir utiliser des manipulations symboliques négatives dans les bases de connaissances. L'étape intermédiaire étant de n'autoriser ces manipulations symboliques qu'en prémisses de règles. Il se peut en effet que l'utilisateur ne connaisse pas la valeur d'un fait mais qu'il connaisse certaines valeurs qu'il ne peut pas prendre ('Je ne connais pas sa profession mais je suis sûr qu'il n'est ni boucher ni boulanger'). Dans ce cas certaines règles peuvent être appliquées puisque la négation de manipulations symboliques est autorisée en prémisses.

La syntaxe obtenue est alors la suivante:

2.1. Syntaxe.

```
<base>::= <regle> | <regle> <base>
<regle>::= <condition> → <conclusion>
<condition>::= <fait> , <condition> | <fait>
<conclusion>::= <litteral> | <manip_symb_pos>
<fait>::= <littéral> | <manip_symbolique>
<litteral>::= <atome> | ¬ <atome>
<manip_symbolique>::= <manip_symb_pos> | <manip_symb_neg>
<manip_symb_pos>::= <atome> = '<valeur>'
<manip_symb_neg>::= <atome> <> '<valeur>'
```

Une manipulation symbolique négative du type F<>'v' est équivalente à ¬F='v'.

2.2. Chaînage avant.

2.2.1. Sémantique.

On définit l'application de Br_{fs} en Br_{fb} de la manière suivante:

On remplace dans Br_{fs} toute manipulation symbolique positive de la forme X='v' par le littéral positif X_v et toute manipulation symbolique négative X<>'v' par le littéral négatif ¬X_v.

Puis pour chaque fait symbolique Fi de $Fs(Br_{fs})$ et pour tous les couples de valeurs d'affectations Vj,Vk de Dom(Fi) tels que j≠k ajouter les règles Fi_Vj → ¬Fi_Vk et Fi_Vk → ¬Fi_Vj.

Il n'y a donc aucun changement par rapport au cas sans négation.

Exemple 24

b → e='1'	sera transformée en	b → e_1
c → e='2'		c → e_2
d → e='3'		d → e_3
e<>'1' , e<>'2' → a		¬e_1, ¬e_2 → a
		e_1 → ¬e_2
		e_1 → ¬e_3
		e_2 → ¬e_1
		e_2 → ¬e_3
		e_3 → ¬e_1
		e_3 → ¬e_2

Extension de la notion de cycle.

D'une manière rigoureuse, il y aura cycle si une manipulation symbolique positive dépend d'elle-même ou d'une manipulation symbolique négative utilisant le même fait symbolique, mais comme précédemment il est clair que si elle dépend d'une autre manipulation symbolique positive utilisant le même fait symbolique, une contradiction sera déclenchée. Nous considérerons donc qu'une base possède un cycle si il existe un fait symbolique dépendant de lui-même. Nous rappelons que les bases que nous traitons sont sans cycle.

Il est à remarquer que si la base est sans cycle, l'ajout de ces règles n'en crée pas puisqu'elles ont des conclusions négatives alors que la base initiale n'en a pas.

2.2.2. Algorithme.

Appelons $Chavt_{fs2}$ le chaînage avant pour des bases avec faits symboliques négatifs uniquement en prémisses défini de la manière suivante (nous ne préciserons que ce qui est ajouté à $Chavt_{fs1}$):

Mémoire de travail.
Maintenant que deux types de manipulations symboliques sont permises il est nécessaire d'effectuer une modification de la mémoire de travail pour les représenter. S'il ne peut y avoir qu'une affectation possible pour un fait symbolique, l'utilisation de manipulations symboliques négatives permet d'avoir plusieurs valeurs interdites pour ce fait. Cette liste doit être évidemment comprise dans le domaine de définition du fait symbolique correspondant. Un fait symbolique est donc toujours représenté par trois paramètres mais le dernier ne contient plus seulement une valeur mais une **liste de contraintes** sur le fait. Cette liste contient donc soit la valeur affectée au fait, soit les valeurs qu'il ne peut pas prendre (valeurs interdites). En effet dès qu'une valeur est connue, les contraintes négatives peuvent alors être supprimées.

Satisfiabilité des prémisses.
Il est maintenant nécessaire de définir très précisément quand une manipulation symbolique sera satisfaite.

- Une manipulation symbolique positive du type F='v' sera satisfaite si F a pour valeur v dans la mémoire de travail.

- Une manipulation symbolique négative du type F<>'v' sera satisfaite si F='y' avec v≠y est présent dans la mémoire de travail ou si v appartient aux valeurs interdites de F.

Détection des contradictions.
- Une manipulation symbolique positive de la forme F='v' apporte une contradiction si F='y' avec v≠y est présent dans la mémoire de travail ou si v appartient aux valeurs interdites de F.

- Une manipulation symbolique négative de la forme F<>'v' apporte une contradiction si F a pour valeur v dans la mémoire de travail.

- Une affectation de la valeur permise ou des valeurs interdites par des valeurs non présentes dans le domaine de définition du fait considéré apporte une contradiction.

2.2.3. Correction de la déduction.

Théorème 37.
$Chavt_{fs2}$ est un chaînage avant correct pour des bases avec faits symboliques négatifs uniquement en prémisses.

Démonstration.
Montrons que tout littéral de Mt_{bf1} appartient à Mt_{bf2}.

Pour qu'une manipulation symbolique du type F='v' soit ajoutée à Mt_{bf2} il faut qu'elle figure en conclusion d'une règle dont les prémisses sont satisfaites. La règle correspondante obtenue par l'application définie précédemment peut donc elle aussi être déclenchée.

a) Le fait n'a pas encore été affecté.

Il est alors ajouté à Mt_{fs} puis sera traduit dans Mt_{fb2}, donc tout littéral de Mt_{fb1} est présent dans Mt_{fb2}

b) Le fait symbolique a déjà été affecté à une autre valeur.

Par définition de $Chavt_{fs}$ une contradiction est détectée. Or si une autre valeur a déjà été affectée c'est qu'une règle concluant sur cette manipulation symbolique s'est déclenchée et donc que la règle de Br_{fb} correspondante par l'application s'est déclenchée elle aussi. Or par définition de $Trad_{br}$, après avoir ajouté ce littéral, toutes les règles de la forme $F\_v \rightarrow \neg F\_v'$ se sont déclenchées. $Chavt_{fb}$ tente donc d'ajouter à la base un littéral dont l'opposé est déjà présent, une contradiction est alors détectée. Donc si $Chavt_{fs}$ détecte une contradiction $Chavt_{fb}$ en détecte une aussi.♦

2.3. Chaînage mixte.

2.3.1. Interface utilisateur.

Pour les faits demandables et le lancement du système rien ne change par rapport au chapitre précédent. Les faits demandables sont des faits symboliques et le système à double tentative est lancé sur un fait symbolique.

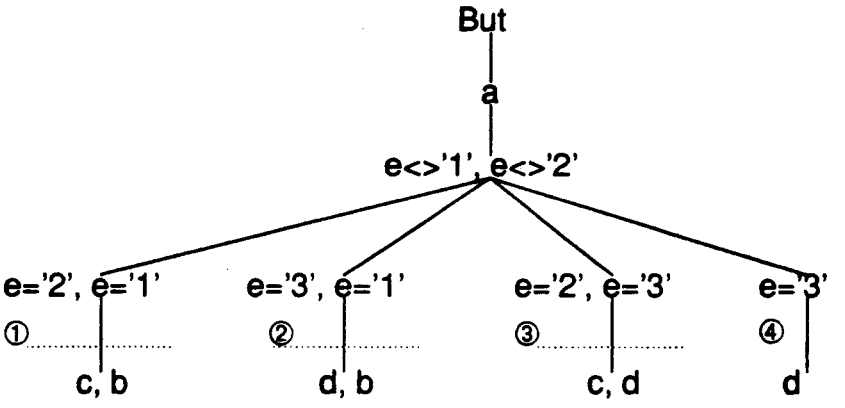
Le traitement de la question à l'utilisateur doit par contre être modifié puisqu'il doit permettre la saisie d'informations négatives concernant un fait s'il y a lieu. La gestion de la question à l'utilisateur devient alors plus complexe. Si l'utilisateur répond 'Je ne sais pas' à une question donnée le fait concerné devient indéterminé mais il faut alors lui demander s'il ne connaît pas des valeurs interdites pour ce fait (il peut y en avoir 0, 1 ou plusieurs). Les valeurs données sont alors rangées dans la liste des contraintes pour ce fait.

2.3.2. Calcul de la question.

En prenant en compte les considérations précédentes, le calcul de la question devient plus complexe que dans le chapitre précédent. Il est en effet nécessaire de simuler la présence de

règles qui concluent sur les manipulations symboliques négatives. Pour cela dès que le chaînage arrière s'intéresse à une manipulation symbolique négative de la forme F<>'v', toutes les règles dont la conclusion est de la forme F='w' avec w≠v doivent être candidates. On remplace donc dans chaque liste de buts en cours de développement toute manipulation symbolique négative de la forme F<>'v' par une manipulation symbolique positive de la forme F='w' pour tout w∈ Dom(F) tel que w≠v. Dans le cas de manipulations symboliques positives rien n'est changé, les règles candidates sont les règles qui possèdent cette manipulation comme conclusion.

Dans l'exemple 24, le but étant a, les listes calculées sont de la forme:



Avec l'utilisation des manipulations symboliques négatives, de nombreuses contradictions sont engendrées dans les listes de questions (Ex: e='1',e='2') notamment quand des conjonctions de manipulations symboliques négatives sont développées comme dans l'exemple précédent. Il devient alors nécessaire de détecter ces contradictions le plus tôt possible afin de couper les branches concernées.

La procédure de vérification partielle du cône de déduction établie pour les faits booléens se révèle ici très adaptée. On interdit qu'une manipulation symbolique de la liste en cours de développement soit en contradiction avec ses ancêtres (les ancêtres allant du but initial à la liste en cours comprise). Pour cela dès que l'on développe une manipulation symbolique positive de la forme F='v', on vérifie qu'il n'existe pas de manipulation symbolique de la forme F='w' avec v≠w ni de manipulation symbolique de la forme F<>'v' dans la liste des ancêtres. Dans le cas d'une manipulation symbolique négative de la forme F<>'v' on vérifie qu'il n'existe pas de manipulation symbolique de la forme F='v' dans la liste des ancêtres.

Dans l'exemple précédent les branches 1,2 et 3 sont alors coupées, seule la 4 est développée et donne comme question: d.

Remarque.

Comme pour la liste de contraintes associée à un fait, la liste des ancêtres peut être simplifiée en supprimant toutes les manipulations symboliques négatives concernant un fait, dès qu'une manipulation positive concernant ce fait est ajoutée. On interdit alors uniquement de dériver

une manipulation symbolique positive de la forme F='v' s'il existe une manipulation symbolique de la forme F='w' avec v≠w dans les ancêtres. Les cas F='v' avec F<>'v' dans les ancêtres ou F<>'v' avec F='v' dans les ancêtres se ramènent au cas précédent après une dérivation supplémentaire.

Vérification de la cohérence de la déduction.
Dans le cas d'une vérification complète de la cohérence de la déduction future, la mémoire temporaire n'a à gérer que des manipulations symboliques positives. En effet comme toutes les manipulations symboliques négatives du type F<>'v' sont remplacées par des manipulations positives du type F='w' avec w≠v il n'y a donc que des manipulations symboliques positives dans les listes de questions. La modification à apporter à la mémoire temporaire est alors très mince. Il suffit de simuler la présence d'une manipulation symbolique négative et de détecter une contradiction en cas d'ajout d'une manipulation symbolique positive de la même manière que pour la mémoire de travail principale.

3. Extension globale aux faits symboliques.

Dans ce chapitre l'utilisation des manipulations symboliques est complète. On permet les manipulations symboliques négatives aussi bien en prémisses qu'en conclusion de règles. Peu de changements interviennent dans les différents algorithmes étudiés précédemment.

3.1. Syntaxe.

```
<regle>          ::= <condition> → <conclusion>
<condition>      ::= <fait> , <condition> | <fait>
<conclusion>     ::= <fait>
<fait>           ::= <littéral> | <manip_symbolique>
<litteral>       ::= <atome> | ¬ <atome>
<manip_symbolique>  ::= <manip_symb_pos> | <manip_symb_neg>
<manip_symb_pos>    ::= <atome> = '<valeur>'
<manip_symb_neg>    ::= <atome> <> '<valeur>'
```

3.2. Chaînage avant.

Aucune modification particulière du chaînage avant n'est à effectuer que ce soit au niveau de la sémantique logique ou au niveau des algorithmes. Le seul point modifié étant le fait qu'une règle déclenchée peut maintenant ajouter une valeur à la liste des valeurs interdites d'un fait donné (Précédemment seul l'utilisateur pouvait le faire).

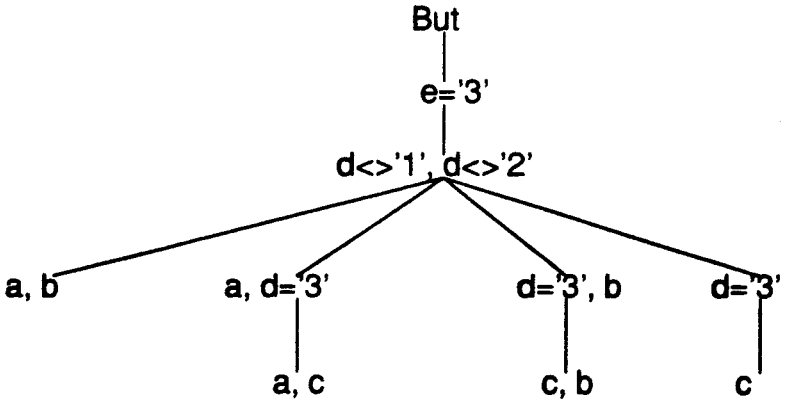
Le pouvoir d'expressivité du langage est maintenant grandement amélioré. Nous pouvons alors écrire des bases telles que celle présentée ici qui fait intervenir des manipulations symboliques négatives aussi bien en prémisses qu'en conclusions.

Exemple 25

a → d<>'1'	MT=∅ But: e
b → d<>'2'	dom(d)=dom(e)=1,2,3
c → d='3'	Les listes intelligentes pour prouver
d<>'1', d<>'2' → e='3'	e sont (a,b)et (c).

Remarque.
Pour un fait X défini sur trois valeurs 1,2 et 3 les faits X<>'1' et X<>'2' n'entraînent pas forcément le fait X='3'. Nous rappelons qu'en logique partielle, un fait n'est établi que s'il est prouvé. Pour avoir la caractéristique précédente il faudrait alors explicitement rajouter une règle de la forme X<>'1, X<>'2' → X='3' mais ce genre de règle établit un cycle. On peut alors contourner le problème en introduisant un fait symbolique secondaire XX qui nous permet d'écrire X<>'1', X<>'2' → XX='3'.

Calcul de la question.
Bien que le chaînage avant ne pose aucun problème, le calcul d'une liste minimale de questions devient maintenant très difficile. Voyons l'arbre de calcul de la question pour l'exemple 25.



Comme une manipulation symbolique négative n'est plus automatiquement remplacée par une manipulation symbolique positive, il devient très difficile de s'assurer que les listes de questions calculées sont minimales (voir les listes a,c et c,b). A ce jour nous n'avons pas encore trouvé d'algorithme permettant d'effectuer cette vérification efficacement. Nous rappelons (voir le chapitre 'Calcul d'une liste minimale de questions') que comme le chaînage mixte effectue un cycle question-saturation, si les règles sont toujours parcourues dans le même ordre, les questions redondantes d'une liste donnée ne seront pas posées, mais il n'est alors plus

possible d'utiliser les comparaisons de listes de questions évoquées dans la partie heuristique.

4. Variables symboliques

Après avoir traité des manipulations symboliques simples, on peut alors définir sur le même schéma l'utilisation de manipulations de variables symboliques. En prémisses cela correspond alors à un test d'égalité ou de différence de deux variables symboliques et en conclusion cela correspond à l'affectation d'une variable par la valeur d'une autre variable.

La syntaxe obtenue est alors la suivante:

4.1. Syntaxe.

```
<base>           ::= <regle> | <regle> <base>
<regle>          ::= <condition> → <conclusion>
<condition>      ::= <fait> , <condition> | <fait>
<conclusion>     ::= <litteral> | <manip_symb_pos>
<fait>           ::= <littéral> | <manip_symbolique>
<litteral>       ::= <atome> | ¬ <atome>
<manip_symbolique>::= <manip_symb_pos> | <manip_symb_neg>
<manip_symb_pos> ::= <atome> = '<valeur>' | <atome> = <atome>
<manip_symb_neg> ::= <atome> <> '<valeur>' | <atome> <> <atome>
```

4.2. Chaînage avant.

Un fait en prémisse du type x=y sera satisfait si x et y sont connus et sont affectés à la même valeur (les valeurs interdites ne sont pas prises en compte).

Un fait en prémisse du type x<>y sera satisfait si x et y sont connus et sont affectés à deux valeurs différentes.

Quand les prémisses sont satisfaites, une conclusion du type x=y sera traitée si y est connu et que x ne l'est pas. Dans ce cas x sera affecté avec la même valeur que y.

Les contradictions seront détectées de la même manière que précédemment dans le cas d'une tentative d'affectation d'une variable avec une valeur différente de celle déjà connue, dans le cas d'une tentative d'affectation d'une variable avec une valeur qui n'est pas dans son domaine de définition ou encore dans le cas d'une tentative d'affectation d'une variable par une valeur appartenant à la liste de ses affectations interdites.

4.3. Chaînage mixte.

Aucune modification de l'interface utilisateur n'est à effectuer. Seul le calcul de la question à poser est à modifier.

- Tout d'abord lors de l'évaluation d'une manipulation symbolique du type a='y' on s'intéresse non seulement aux règles qui possèdent cette manipulation en conclusion mais aussi à toutes les règles dont la conclusion est de la forme a=b. Dans ce cas la recherche est alors lancée sur les prémisses de cette règle ainsi que sur la manipulation b='y'.

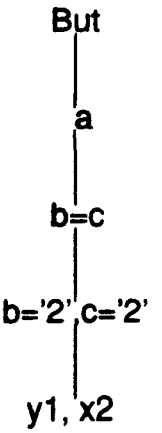
- Lors de l'évaluation d'une manipulation symbolique du type a<>'y' on s'intéresse non seulement aux règles qui possèdent une manipulation symbolique du type a='x' avec x<>y mais aussi à toutes les règles dont la conclusion est de la forme a=b. Dans ce cas la recherche est alors lancée sur toutes les listes formées par concaténation des prémisses de cette règle et d'une manipulation du type b='x' avec x<>y.

- Lors de l'évaluation d'une manipulation de variables de la forme a=b on s'intéresse aux règles qui permettent de montrer a='x' et b='x' pour tout x appartenant à l'intersection des domaines de définition de a et de b.

- Lors de l'évaluation d'une manipulation de variables de la forme a<>b on s'intéresse aux règles qui permettent de montrer a='x' et b='y' avec x<>y, x appartenant au domaine de définition de a et y appartenant au domaine de définition de b.

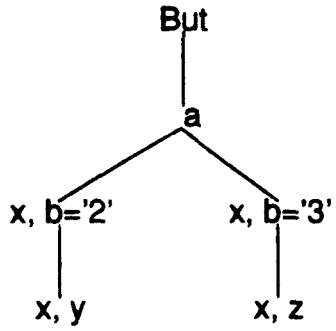
Exemple 26

```
b=c → a          Dom(b,[1,2]),dom(c,[2,3])
x1 → b='1'       MT=∅      But: a.
y1 → b='2'       la liste de questions intelligentes est (y1,x2)
x2 → c='2'       En aucun cas x1 ne doit être demandé.
y2 → c='3'
```



Exemple 27

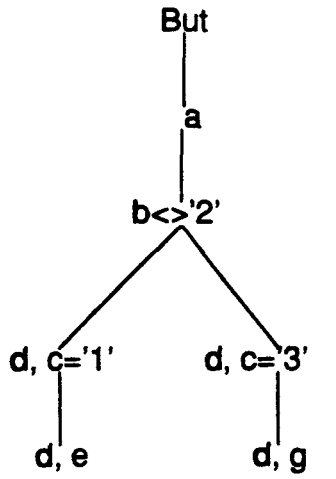
x → a=b	Dom(a,[2,3]),Dom(b,[1,2,3]),Dom(c,[1,2])
t → b='1'	MT=∅ But: a
y → b='2'	La liste de questions intelligentes est (x, y)
z → b='3'	En aucun cas t et z ne doivent être demandés
z → c='3'	



Contradiction sur z

Exemple 28

b<>'2' → a	Dom(b,[1,2,3]) Dom(c,[1,2,3])
d → b=c	MT=∅ But: a
e → c='1'	Les listes de questions intelligentes sont (d,e)
f → c='2'	ou (d,g).En aucun cas f ne doit être demandé.
g → c='3'	



5. Méta-valeurs

Nous avons vu dans le chapitre 'Représentation interne' que le système utilise des méta-valeurs pour chaque atome notamment pour ne pas poser plusieurs fois la même question. Il est alors très intéressant d'avoir la possibilité de tester ces méta-valeurs dans les règles de la base. Cette solution a déjà été envisagée dans des systèmes comme Intelligence service [IS 86], Guru [Guru 87] ainsi que dans [Del 87a]. Se référer à ceux-ci pour trouver une justification complète de l'utilisation des méta-valeurs. Ces méta-valeurs se réfèrent toutes à des atomes ou des faits symboliques et non pas à des littéraux ou des manipulations symboliques. Elles ne peuvent être utilisées que dans le corps d'une règle et ne peuvent pas être utilisées dans une question. Nous appellerons méta-test tout test utilisant une méta-valeur. Nous présentons ici les trois méta-valeurs les plus couramment utilisées (connu, inconnu et indéterminé).

Connu

Exemple d'utilisation: permet de forcer l'évaluation complète du but concerné. En d'autres termes lancer un interpréteur à double tentative sur ce but en cours d'évaluation. Cette méta-valeur est particulièrement utilisée pour les faits symboliques.

Exemple:

b==connu → a	avec c on déduit une valeur sur b, donc b devient connu
c → ¬b	et a est déduit.

Inconnu

Exemple d'utilisation: permet d'affecter une valeur par défaut aux faits qui ne figurent pas initialement dans la base (fichier d'initialisation dynamique).

Exemple:

b==inconnu → a	Si b n'est pas initialement affecté à une valeur alors a devient vrai.

Indéterminé

Exemple d'utilisation: permet d'affecter une valeur par défaut à un fait quand l'utilisateur ne connaît pas sa valeur (uniquement pour les faits demandables) ou de gérer la réponse 'Je ne sais pas' de la part de l'utilisateur.

Exemple:

b → d	Si l'utilisateur ne connaît pas la valeur de b
¬b → c	a devient vrai.
b==indéterminé → a	

5.1. Syntaxe.

La grammaire obtenue en utilisant des manipulations symboliques négatives uniquement en prémisses est alors la suivante:

```
<base>           ::= <regle> | <regle> <base>
<regle>          ::= <condition> → <conclusion>
<condition>      ::= <fait> , <condition> | <fait>
<conclusion>     ::= <litteral> | <manip_symb_pos>
<fait>     ::= <littéral> | <manip_symbolique> | <meta-test>
<litteral>           ::= <atome> | ¬ <atome>
<manip_symbolique>   ::= <manip_symb_pos> | <manip_symb_neg>
<manip_symb_pos>     ::= <atome> = '<valeur>'
<manip_symb_neg>     ::= <atome> <> '<valeur>'
<meta-test>          ::= <atome> == <meta-valeur>
<meta-valeur>        ::= connu | inconnu | indetermine
```

5.2. Chaînage avant.

Comme aucun nouveau type de fait ne peut être ajouté à la mémoire de travail, aucune modification de celle-ci n'est à effectuer. Il est néanmoins nécessaire de définir quand un méta-test sera satisfait.

Il est évident qu'il n'y a pas à proprement parler de sens logique à ces méta-tests (sauf peut-être pour connu qui peut être justifié par un très grand nombre de règles). Seul un sens opérationnel existe, c'est celui que nous avons détaillé dans le paragraphe précédent.

- Un méta-test du type x==inconnu sera satisfait si rien n'est connu sur le fait x dans la mémoire de travail.

- Un méta-test du type x==connu sera satisfait si une valeur a été affectée au fait x dans la mémoire de travail.

- Un méta-test du type x==indéterminé sera satisfait si la question x a déjà été posée à l'utilisateur et que celui-ci a répondu 'je ne sais pas'.

5.3. Chaînage mixte.

5.3.1. Interface utilisateur.

Dans l'interface utilisateur seuls quelques points sont à préciser:

- Le lancement du chaînage mixte ne demande en effet aucune modification car on n'autorise pas de question sur un méta-test.

- La question à l'utilisateur par contre doit être modifiée. Une question peut maintenant contenir une méta-valeur (ex: a==indeterminé ?). Dans ce cas seul l'atome associé (ici a) sera demandé.

Par extension quand un méta-test est de base, l'atome associé est considéré par défaut comme demandable.

5.3.2. Calcul de la question

Le calcul de la question devient ici plus délicat.

Dans le cas où un méta-test se trouve dans une liste de buts sans être satisfait, d'autres considérations doivent être effectuées.

- Pour un fait qui n'est pas inconnu aucune question ne peut être posée à l'utilisateur pour qu'il le devienne. S'il n'est pas satisfait, la liste de buts peut donc être abandonnée.

- Pour un fait qui n'est pas indéterminé, s'il est inconnu et demandable on peut encore le rendre indéterminé en posant la question à l'utilisateur sur l'atome associé. Il est donc nécessaire de garder ce méta-test dans la liste de buts. Ce méta-test peut donc se retrouver dans la liste finale de questions si l'atome associé est demandable. Dans ce cas nous rappelons que le calcul de son coefficient de demandabilité est le complément à 1 des différents coefficients attachés à cet atome.

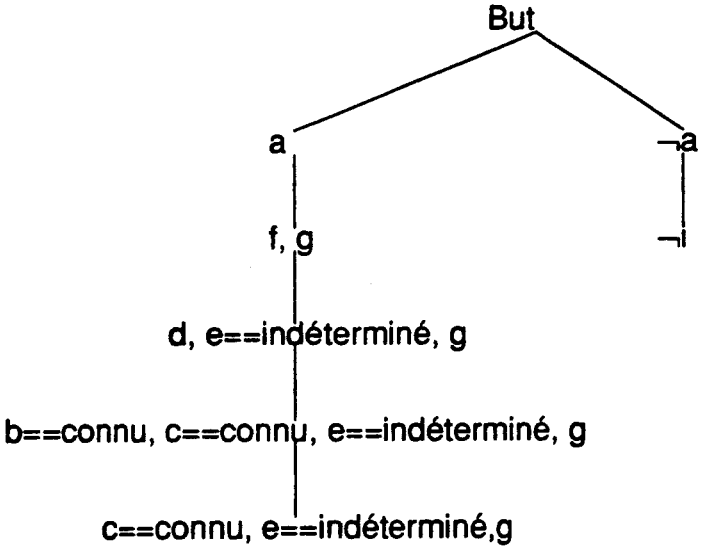
- Un fait qui n'est pas connu peut le devenir par déduction ou par question sur le fait considéré. Il faut donc s'intéresser à toutes les manipulations symboliques qui utilisent ce fait. Pour tout méta-test de la forme x==connu remplacer dans la liste de buts en cours de développement ce méta-test par x='v' $\forall v \in$ dom(x). Ce méta-test ne se retrouvera donc jamais dans une liste finale de questions puisqu'il sera remplacé à chaque fois par une manipulation symbolique positive.

Il est à remarquer que les méta-valeurs inconnu et indéterminé sont non monotones. Le méta-test peut être satisfait à un instant donné et ne plus l'être par la suite. L'ordre des règles pour ces deux méta-valeurs est donc capital. Avec l'utilisation de ces méta-valeurs le chaînage avant n'est plus monotone.

Dans l'hypothèse d'une vérification complète de la liste de questions, l'algorithme de chaînage avant temporaire doit maintenant prendre en compte le fait qu'un atome peut être initialement considéré comme indéterminé. En effet comme un méta-test de la forme x==indéterminé peut se trouver dans une liste de questions il faut pouvoir l'ajouter à la mémoire temporaire pour vérifier si le but initial peut être déduit.

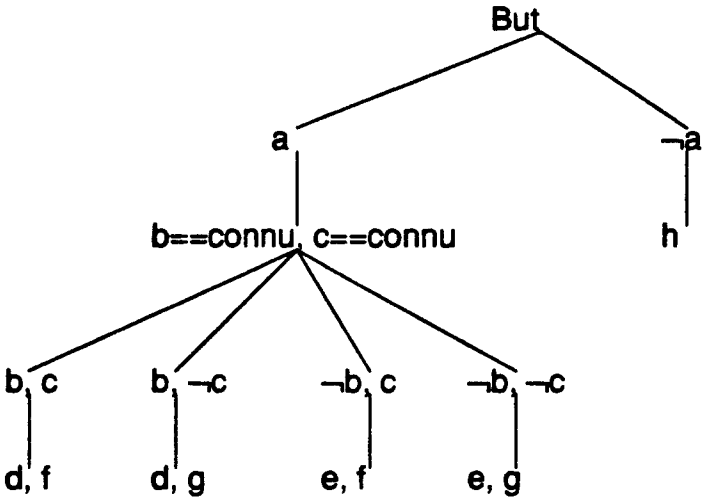
Exemple 29

k==inconnu → ¬b	MT=∅	But: a
b==connu et c==connu → d	Avec	c=faux
d, e==indéterminé → f		e=indéterminé
¬i → ¬a		g=faux
f, g → a		i=faux
	On obtient a faux	



Exemple 30

b==connu et c==connu → a	MT=∅	But: a
h → ¬a	Avec	d faux
d → b		e vrai
e → ¬b		f vrai
f → c	On obtient a vrai	
g → ¬c		



Extension possible.

Il peut être envisagé d'ajouter d'autres méta-valeurs au schéma précédent. Par exemple, une méta-valeur du type non_déductible peut être très utile pour activer des règles dès qu'un fait ne peut plus être déduit.

Exemple:

$c \rightarrow b$	Si c et d sont faux, b ne peut plus être déduit
$d \rightarrow \neg b$	et donc a devient vrai.
b==non_deductible $\rightarrow$ a.	

Un méta-test du type x==non_deductible sera satisfait si le fait x n'est pas connu et qu'aucune question ne peut être posée pour déduire ce fait.

Si le chaînage avant ne pose aucun problème, le calcul de la question intéressante devient générateur d'explosion combinatoire. En effet un fait qui n'est pas non_déductible peut le devenir si plus rien ne peut déduire une valeur sur ce fait. Il faut donc calculer toutes les listes de questions qui permettent de déduire une valeur sur ce fait, ce qui correspond à une formule sous forme normale disjonctive, puis calculer la négation de cette formule que l'on met ensuite sous forme normale conjonctive. Si la base est conséquente, cette opération génère très vite de très nombreux calculs qui fournissent d'énormes listes de buts, ce qui n'est pas satisfaisant.

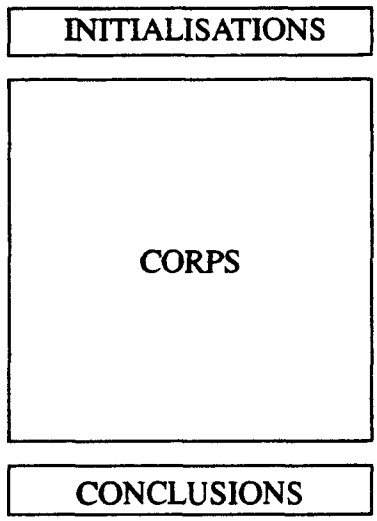
Exemple:

h==non_deductible $\rightarrow$ a
b,c $\rightarrow$ h
d,e $\rightarrow \neg$h

Les listes de questions intéressantes sont: (¬b ¬d) (¬b ¬e) (¬c ¬d) (¬c ¬e)

En d'autres termes calculer les listes de questions pour h et ¬h et calculer la négation de la formule associée.

Une autre manière de voir les choses est de considérer que cette méta-valeur ne peut être déclenchée qu'en chaînage avant. Ce qui revient à dire que si la base a été saturée et que le fait concerné est non_déduit alors le méta-test est satisfait. On a alors une base de connaissances segmentée en trois parties. Une partie initialisations contenant les opérations à effectuer avant toute déduction, une partie corps sur laquelle les déductions sont effectuées et une partie conclusions contenant les opérations à effectuer après arrêt de toute déduction, dans laquelle on accepte l'utilisation de la méta-valeur non_déduit. Les inférences ne se font alors que dans la partie corps et seul un parcours séquenciel des règles est effectué dans les parties Initialisations et Conclusions.



Utilisation de la méta-valeur Inconnu pour les initialisations par défaut.

Corps de la base sur lequel les inférences sont effectuées. Utilisation possible des méta-valeurs Connu et Indéterminé.

Gestion des conclusions avec possibilité d'utiliser la méta-valeur Non_déduit.

Réalisation.

Le système BIVOUAC a été réalisé à l'Université de Lille 1 pour implémenter toutes les techniques décrites dans cette thèse. Il se présente comme une boîte à outils logiques contenant notamment:

- un précompilateur effectuant la fermeture des bases de connaissances.
- un vérificateur d'achèvement relativement aux faits de base.
- un compilateur de bases de connaissances.
- un simplificateur de bases de règles redondantes.
- un vérificateur de clauses conséquences bivaluées.
- un chaînage avant rapide sur les règles.
- un chaînage avant rapide sur les clauses.
- un calcul de questions pertinentes.
- un calcul de questions intelligentes.
- et enfin la possibilité d'utiliser pour les deux derniers algorithmes la notion de coefficient de satisfaisabilité pour une heuristique efficace.

Ce système est le plus ouvert possible de manière à pouvoir lui adjoindre le plus aisément possible d'autres outils ou de tester facilement d'éventuelles modifications. Il serait par exemple intéressant d'y ajouter les champs de production de Pierre Siegel.

Le premier prototype de BIVOUAC a été écrit en Prolog (Quintus-Prolog [QP 88]) ce qui a permis une mise au point rapide et un code court, lisible et facilement modifiable. Actuellement une traduction complète de ces outils en langage C est en cours pour permettre des traitements plus rapides. Nous incluons dans BIVOUAC le traitement complet des faits booléens, les manipulations symboliques positives aussi bien en prémisses qu'en conclusion, les manipulations symboliques négatives uniquement en prémisses, les méta-valeurs Connu, Inconnu et Indéterminé, l'égalité et la différence de variables symboliques en prémisses et enfin l'affectation d'une variable par une autre en conclusion de règle. Actuellement une extension aux faits réels est en cours pour les questions intelligentes, bien que l'achèvement dans ce cas ne soit pas encore possible.

BIBLIOGRAPHIE

[ABW 88] Apt, Blair et Walker. *Towards a theory of declarative knowledge. Foundations of deductive databases and logic programming*. Minker(ed), M. Kaufmann, Los Altos California 1988.

[AC 86] J.L. Alty et M.J. Coombs. *Systèmes experts, Concepts et exemples*. Masson 1986.

[Bla 86] S. Blamey. *Partial logic*. Handbook of Philosophical Logic (D. Gabbay, F. Guenthner eds) Vol III, Reidel Publishing Company, 1986.

[Bra 86] I. Bratko. *Prolog programming for artificial intelligence*. Addison-Wesley publishing company inc. 1988.

[Bry 86] R.E. Bryant. *Graph-Based Algorithms for Boolean Function Manipulation*. IEEE Transactions on computers, Vol C-35, n°8, Aout 86.

[BS 87] W. Buttner et H. Simonis. *Embedding Boolean Expressions into Logic Programming*. J,. of Symbolic Computation, n°4, p191-205, Octobre 1987.

[BS 85] G. Bossu et P. Siegel. *Saturation, Nonmonotonic Reasoning ans the Closed-Word Assumption*. Artificial Intelligence, n°25, Janvier 1985.

[Car 66] L. Caroll, *La Logique sans Peine*, Hermann, Paris, 1966.

[Cha 86] K. H. Chan. *Représentation of negative and incomplete information in Prolog*. Sixth Canadian Conference on Artificial Intelligence, Montreal, 21-23 may 1986.

[CL 73] C. L. Chang et R. T. Lee. *Symbolic logic et mechanical Theorem proving*. Academic press, New York, 1973

[Col 72] A. Colmerauer. *Prolog : un système de communication homme-machine en français*. Rapport interne GIA Marseille Luminy 1972.

[Col 87] A. Colmerauer. *Opening the Prolog-III Universe*. Byte magazine, 12(9), Aout 1987.

[Cor 84] M.O. Cordier. *Les systèmes experts*. La Recherche n°151 Janvier 1984. P60-70.

[Cor 86] M.O. Cordier.*Informations incomplètes et contraintes d'intégrité: le moteur d'inférences SHERLOCK*. Thèse d'état. Université d'Orsay. Novembre 1986.

[Dem 90] R. Demolombre. *A Strategy for the Computation of Conditional Answers.* Rapport interne ONERA/CERT Toulouse 1990.

[DeK 86] J. De Kleer. *An Assumption-Based Truth-Maintenance System.* Artificial Intelligence, vol 28,1,1986

[Dinc 88] M. Dincbas, P.Van Hentenryck, H, Simonis, A. Aggoun, T. Graf, F. Bertier, *The Constraint Logic Programming Language CHIP.* Proceedings of the Int. conf. on fifth generation computer systems, ICOT, 1988.

[Del 86] J.P. Delahaye. *Outils logiques pour l'intelligence artificielle.* Editions Eyrolles 1986.

[Del 87a] J.P. Delahaye. *Systèmes experts : Organisation et programmation des bases de connaissances en calcul propositionnel.* Ed Eyrolles, Paris 1987.

[Del 87b] J.P. Delahaye. *Chaînage avant et calculs de modèles en logique bivaluée et trivaluée.* 7èmes Journées Internationales sur les Systèmes Experts et leurs Applications. Avignon 1987.

[Del 87c] J.P. Delahaye. *Programmation en Logique Trivaluée.* Publication I.T. n°115, du Laboratoire d'Informatique Fondamentale de Lille (U.A. CNRS 369), Université des Sciences et Techniques de Lille. Octobre 1987.

[DF 90] R. Demolombre, L. Farinas Del Cerro. *An Inference Rule for Hypothesis Generation.* Rapport interne ONERA/CERT Toulouse 1990.

[DG 84] W. Dowling et J. Gallier. *Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae.* J. of Logic programming, n°3, p267-284, 1984.

[DP 60] M.Davis, H. Putnam. *A Computing Procedure for Quantification Theory.* JACM 7, p201-215, 1960.

[DT 90a] J.P. Delahaye, V. Thibau. *Optimal Fixed Point Semantics.* JELIA 90, Amsterdam.

[DT 90b] J.P. Delahaye, V. Thibau. *Programming in Three-Valued Logic*, TCS (to appear), 1990.

[Fit 85] M. Fitting. *A Kripke-Kleene semantics for logic programs.* J. of Logic Programming, vol 3, p. 93-114, 1985.

[Fit 86] M. Fitting. *Partial Models and Logic Programming.* TCS, vol 48 p229-255,1986.

[Fit 88] M. Fitting, M. Ben-Jacob. *Stratified and Tree-valued Logic Programming Semantics*. Proc. 5th International Conference and Symposium in Logic Programming (Edited by Kowalski and Bowen), p.1054-1069,1988.

[FG 87] H. Farreny et M. Ghallab. *Eléments d'intelligence artificielle*. Ed Hermes, Paris.

[Frot 90] P. Frot. *Trois systèmes experts en Turbo-Pascal utile*. Sybex 1990.

[Gan 85] J.G. Ganascia. *La conception des systèmes experts*. La Recherche n°170, Octobre 85. P1142-1151.

[Gha 88] M. Ghallab. *Compilation de bases de connaissances*. LAAS, CNRS, Toulouse, 1988.

[Guru 87] *Guru 1.1* MDBS Inc. Distributeur France Ise-Cegos. Manuel de référence, 1987.

[GU 89] G. Gallo et G. Urbani. *Algorithms for Testing the Satisfiability of Propositional Formulae*. J. of Logic Programming, n°7, p45-61, 1989.

[HWL 83] F. Hayes-Roth, D. Waterman, D. Lenat, *Building Expert Systems*, Addison-Welsey, New-York, 1983

[IS 86] *Intelligence service II*. GSI-TECSI. Manuel utilisateur, 1986.

[KG 85] Y. Kodratoff, J.G. Ganascia. *Démonstration automatique de théorèmes et systèmes experts*. 5èmes journées int. "Les systèmes experts et leurs applications", Avignon 85

[KK 71] R.A. Kowalski, D. Kuehner. *Linear Resolution with Selection Fonction*. Artificial Intelligence 2, p227-260, 1971

[Kle 67] S.C. Kleene. *Introduction to metamathematics*. North-Holland Amsterdam, 1967.

[Kow 79] R. Kowalski. *Logic for problem solving*. Elsevier North-Holland inc, New-York, 1979.

[Kun 87] K. Kunen. *Negation in Logic Programming*. Journal of Logic Programming, vol 4, 1987, p289-308.

[Kun 89] K. Kunen. *Signed Data Dependencies in Logic Programs*. Journal of Logic Programming, vol 7, 1989, p231-245.

[LM 85] J.L. Lassez and M.J. Maher. *Optimal fixedpoints of logic programs* TCS, 39, p.15-25, 1985.

[Lee 67] C.T. Lee. *A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms*. Ph.D. Thesis, University of California, Berkeley, 1967.

[Llo 84] J. W. Lloyd. *Foundations of logic programming*, Springer Verlag, 1984.

[Lov 78] D. W. Loveland. *Automated Theorem Proving*. A logical basis. North Holland Publishing Company, Amsterdam, 1978.

[Luk 63] J. Lukasiewicz. *Elements of Mathematical Logic*. Pergamon Press, 1963.

[LS 88] Léa Sombé. *Inférences non classiques en Intelligence Artificielle*. Réunion du PRC IA les 14-15 mars 1988, Toulouse.

[MD 89a] P. Mathieu et J.P. Delahaye. *Logique partielle et Prolog*. Séminaire de Programmation en Logique de Trégastel (SPLT 89), 1989.

[MD 89b] P. Mathieu et J.P. Delahaye. *L'achèvement et ses applications aux interpréteurs de règles*. Publication IT n°172 LIFL, Septembre 1989.

[MD 90a] P. Mathieu et J.P. Delahaye. *The Logical Compilation of Knowledge Bases*. JELIA 90, Amsterdam.

[MD 90b] P. Mathieu et J.P. Delahaye. *For which Bases Forward Chaining is Sufficient*. COGNITIVA 90, Madrid.

[MD 90c] P. Mathieu et J.P. Delahaye. *How to Make a Complete Computation with Forward Chaining*. IASTED 90, Honolulu.

[Mat 90] P. Mathieu. *Le calcul des questions intelligentes dans les systèmes experts*. Publication IT n°185 LIFL, 1990.

[MW 88] D. Maier et D.S. Warren. *Computing with Logic*. Benjamin Cummings Publishing Company inc. 1988.

[Myc 84] A. Mycroft. *Logic Program and many-valued logic*. STACS 84 (M. Fontet, K. Mehlhorn). LNCS 166, p. 274-286, 1984.

[Nil 80] N. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Co, Palo Alto

[Oke 90] R. O'Keefe. *The Craft of Prolog*. The MIT press, 1990.

[OR 89] L. Oxusoff & A. Rausy. *L'Evaluation Semantique en Calcul Propositionnel*. These nouveau régime, Universite d'Aix-Marseille II, 1989.

[Pof 89] L. Pofelski. Rapport de DEA 1989. LIFL.

[Pra 70] D. Prawitz. *A Proof Procedure with Matrix Reduction.* Lecture Notes in Mathematics, 125, Springer Berlin, p207-213,1970.

[Prz 89] T. Przymusinski. *Non-monotonic formalisms and logic programming.* Proc. 6th international conference on logic programming, Levi and Martelli eds.

[Quin 55] W.V. Quine. *A Proof Procedure for Quantification Theory.* J. Symbolic Logic 20, no 2, p141-149, 1955.

[QP 88] *Quintus Prolog.* Quintus computer system inc. Distributeur France Elsa Software. Reference manual.

[RT 90] O. Ridoux, H. Tonneau. *Une mise en œuvre de l'Unification d'Expressions Booléennes.* Séminaire de Programmation en Logique de Trégastel (SPLT) 1990.

[Rob 65] J.A. Robinson. *A machine oriented Logic based on the resolution principle.* Journal of the association for Computing Machinery vol 12, 1965

[Rou 83] M.C. Rousset. TANGO: *Moteur d'Inférences pour une Classe de Systèmes Experts avec Variables.* Thèse de 3ième cycle, Université d'Orsay, Octobre 1983.

[Rou 88] M.C. Rousset. *Sur la Cohérence et la Validation des Bases de Connaissances: le Système COVADIS.* Thèse d'état, Universite d'Orsay, Septembre 1988.

[She 88] J.C. Shepherdson. *Negation in Logic Programming.* Foundations of deductive databases and logic programming (J. Minker ed), Morgan Kaufmann, p.19-88, 1988.

[Sie 87] P. Siegel. *Représentation et utilisation de la connaissance en Calcul Propositionnel.* Thèse d'état, GIA-Marseille-Luminy, 1987

[SS 86] L. Sterling et E. Shapiro. *The art of prolog.* The MIT Press 1986

[Urq 86] A. Urquhart. *Many-valued Logic.* Handbook of Philosophical Logic (D. Gabbay, F. Guenthner eds) Vol III, Reidel Publishing Company, 1986.

[TDB 86] Terrier, Decamps et Bonnemay. *Système Expert pour Diagnostic rapide: Procedure de Compilation de Règles à Conclusions Dubitatives.* 6èmes journées int. "Les systèmes experts et leurs applications", Avignon 86

[Ter 87] F. Terrier. *Représentation de la Déduction par une logique trivalente: Réalisation de moteurs d'inférences rapides.* Thèse de Doctorat, Université Paris-sud centre d'Orsay, 1987.

[Thi 90] V. Thibau. *Une logique trivaluée appliquée à la programmation logique.* Thèse nouveau régime, Université de Lille 1, 1990.

[Tur 84] R. Turner. *Logics for Artificial Intelligence*. Ellis Horwood, 1984.

[VEnt 89] P. Van Entenryck. *Constraint Satisfaction in Logic Programming*. The MIT Press, 1989.

Abstract of the thesis in English.

The aim of this work is to study how to use a particular thee-valued logic in expert systems. This logic first defined by J.P. Delahaye is able to characterize exactly the set of literals computed by forward chaining. This thesis is in two parts. The first one study the incompleteness of forward chaining and solve this problem by using a Robinson's resolution based method that we call logical compilation. Specially we define in the chapter III the achievement notion and we give sufficient properties to imply that a knowledge base is achieved for basic facts. This defines a methodology to construct knowledge bases. We present in the chapter IV the logical compilation and we give many algorithms to make this compilation. They allow not only to compute soundly with a forward chaining, but also to solve the problem the 'or' connective in rule conclusions problem. The second part study the problem of computing useful queries to ask to the user in a three-valued expert system based in mixed chaining. We give in the chapter III two formal definitions of queries based on our three-valued logic: the 'intelligent queries' and the 'pertinent' queries. We propose then a two-level structutation of query algorithms: the logical level and the heuristic level. First of all we describe different logical levels that we can obtain according to the method used. Then we propose two algorithms which compute intelligent queries and pertinent queries. We present too different heuristic levels and specially one of them based on the use of satisfiability coefficients applied to the queries computed by the logical level. Finally we extend these results to knowledge bases with symbolic facts, meta-values and symbolic variables. All the algorithms of this thesis have been realised in the BIVOUAC soft which is a logical toolbox for knowledge bases.

Key-words.

ARTIFICIAL INTELLIGENCE
EXPERT SYSTEMS
FORWARD CHAINING
KNOWLEDGE BASE
LOGIC
MIXED CHAINING
QUERIES
RESOLUTION
SOUNDNESS
THREE-VALUED

Résumé de la thèse en Français.

L'objet de ce travail est d'étudier l'utilisation possible d'une logique trivaluée particulière dans les systèmes experts. Cette logique définie par J.P. Delahaye permet de caractériser parfaitement l'ensemble de littéraux calculés par un chaînage avant. Cette thèse se divise en deux parties distinctes. La première partie traite de l'incomplétude du chaînage avant et résout ce problème par l'utilisation d'une méthode basée sur la résolution de Robinson que nous appelons compilation logique. Notamment dans le chapitre III nous établissons le principe d'achèvement et nous donnons des propriétés pour s'assurer qu'une base est achevée relativement aux faits de base, ce qui permet de définir une méthodologie de construction de bases de connaissances. Puis dans le chapitre IV nous présentons la compilation logique proprement dite et nous donnons un certain nombre d'algorithmes pour effectuer cette compilation qui permet non seulement d'effectuer un calcul complet avec un chaînage avant mais aussi de résoudre le problème du connecteur 'ou' en conclusion de règles. La seconde partie pose le problème du calcul de questions utiles à poser à l'utilisateur dans un système expert trivalué fonctionnant en chaînage mixte. Nous donnons dans le chapitre III deux définitions formelles de questions basées sur notre logique trivaluée: les questions 'intelligentes' et les questions 'pertinentes', puis nous proposons une structuration des systèmes de calcul d'une question en deux niveaux: le niveau logique et le niveau heuristique. Nous détaillons tout d'abord les différents niveaux logiques pouvant être obtenus selon les méthodes utilisées et nous présentons deux algorithmes permettant de calculer des questions pertinentes et des questions intelligentes. Nous présentons ensuite les différents niveaux heuristiques qui nous ont amené à définir une méthode basée sur l'attribution de coefficients de satisfaisabilité aux différentes questions pouvant être posées. Enfin nous étendons ces résultats aux bases avec faits symboliques, méta-valeurs et variables symboliques. Les différents algorithmes présentés dans cette thèse ont été implémentés dans le logiciel BIVOUAC qui se présente comme une boîte à outils logiques pour bases de connaissances.

Mots clés.

BASE DE CONNAISSANCES — COMPLETUDE

CHAINAGE AVANT — CHAINAGE MIXTE

INTELLIGENCE ARTIFICIELLE — LOGIQUE

QUESTIONS — RESOLUTION

SYSTEMES EXPERTS — TRIVALUE