LABORATOIRE
EISTI
LAπPI
EN PROCESSUS INTELLIGENTS

Chrysostome BASKIOTIS

LOGIQUE COMPUTATIONNELLE

EISTI, 2009-10

LOGIQUE COMPUTATIONNELLE

Site du cours

http ://sifoci.eisti.fr

Cours de 2e année, 1er semestre.

20h00 d'enseignement décomposées en 6h00 de cours et 14h00 de TD. Examen de 2h00.

Objectif :

La logique computationnelle malgré son aspect assez théorique constitue un formidable outil pour la pratique de l'informaticien. En effet l'industrie informatique se trouve confrontée à ce qu'on appelle *la crise du logiciel* c'est-à-dire à la difficulté de *vérifier* la conformité d'un système informatique par rapport à ses spécifications formelles et de *valider* que le comportement observé du système n'est pas différent du comportement attendu. Les méthodes de vérification sont fondées sur la logique computationnelle.

Ainsi, après un bref rappel de la logique propositionnelle, on étudiera la logique des prédicats qui est l'outil de base pour la logique. On examinera ensuite l'univers et le modèle d'Herbrand qui permet de déterminer la réponse correcte d'un programme. On terminera avec l'étude de la logique floue.

Prérequis :

Mathématiques pour l'ingénieur. Théorie des graphes. Algèbre de Boole.

Professeurs :

- CERGY : **Chrysostome Baskiotis** (Cours+TD), **Jean-Paul Forest** (TD)
- PAU : **Yannick Le Nir** (Cours+TD)

Supports du cours :

- Polycopié du cours.
- R. Cori & D. Lascar : *Logique mathématique*, 2 vol. Masson, 1993
- J. W. Lloyd : *Fondements de la programmation logique*, Eyrolles, 1988
- K. R. Apt & E.-R. Olderog : *Verification of sequential and concurrent programs*, Springer, 1991

Supports supplémentaires

- E. W. Beth : *The foundations of mathematics*, North-Holland, 1965

– G. S. Boolos & R. C. Jeffrey : *Computability and logic*, 3e Ed., Cambridge Univ. Press, 1989
– A. Church : *Introduction to mathematical logic*, Princeton Univ. Press, 1956
– R. Fraïssé : *Cours de logique mathématique*, 2 vol., Gauthiers-Villars, 1972
– M. R. Genesereth & N. J. Nilsson : *Logical foundations of artificial intelligence*, Morgan Kaufmann, 1987
– G. Kreisel & J. L. Krivine : *Éléments de logique mathématique*, Dunod, 1966
– U. Nilsson & J. Małuszyński : L*ogic, Programming and Prolog*, Wiley, 1990
– H. Rasiowa & R. Sikorski : *The mathematics of metamathematics*, PWN, 1963
– D. van Dalen : *Logic and structure*, Springer, 1997

Contrôle des connaissances :
– Examen.

Remarques : Sur le site du cours vous trouverez
– Le poly du cours, réactualisé chapitre par chapitre.
– Des indications pour le cours et le TD de chaque semaine.
– Des documents complémentaires et, en particulier, la forme électronique du livre de Nilsson et Małuszyński, cité en bibliographie.

INTRODUCTION

L'EISTI a toujours donné une large place dans son cursus à l'enseignement des fondements de l'Intelligence Artificielle (IA). Sous ce nom il y a plusieurs branches de Mathématiques et de l'Informatique qui se regroupent. Le cycle des cours consacré à l'IA est composé de Langages, Logique Computationnelle, Prolog, Décidabilité, Intelligence Artificielle et Systèmes Experts. Ce cycle est complété par l'option Génie Logiciel - Systèmes Intelligents et Complexes (GL-SICO) qui est, en partie, dédiée à l'IA. Il faut aussi signaler que la plupart des autres options font appel à diverses parties de l'IA.

Il est très difficile, voire impossible, d'avoir une définition de l'IA. La situation la plus acceptable serait d'utiliser une approche circulaire du type « l'intelligence artificielle est la résolution de problèmes par modélisation de connaissances ». En revanche il est très facile d'énumérer les disciplines qui se réclament, par elles-mêmes, comme faisant partie de cette science. Situation normale dans la mesure où l'IA se trouve au croisement des plusieurs disciplines et pour lesquelles constitue un des enjeux principaux. Mais en même temps situation embarrassante pour l'honnête homme de ce début de siècle dans sa tentative d'approcher son contenu.

Il n'est guère possible dans ce cas que de commencer par le début. Nous pouvons dire que l'objet de l'IA est la connaissance de la connaissance. Définition parallèle à celle donnée par Paul Valéry pour l'homme : « une tentative pour créer

ce que j'oserai nommer l'esprit de l'esprit »[1]. Bien que l'aventure soit récente, l'histoire est ancienne. Descartes[2] est certainement le premier qui a posé le problème de la connaissance dans sa généralité et qui a inventé une Méthode pour l'aborder. Et qui, de plus, a décrit son parcours : « Je me trouvais comme contraint d'entreprendre moi-même de me conduire. Comme un homme qui marche seul et dans les ténèbres ». En fait ce qui préoccupe Descartes au beau milieu du deuxième millénaire est l'éternel défi : régler, comme un mécanisme d'horlogerie, la démarche de l'esprit pour conquérir la connaissance, cette chimère sauvageonne et indomptable.

De ces 400 ans qui nous séparent de l'époque de Descartes, les plus importants sont les derniers 60 ans, pendant lesquels une véritable explosion a eu lieu à l'intérieur de l'IA. Création des nouvelles branches : réseaux neuronaux, algorithmes génétiques, machines à support vectoriel, boosting, systèmes experts, agents intelligents, apprentissage par renforcement, traitement du langage naturel, etc. Réorientation d'anciennes branches : classification, classement, reconnaissance de formes, traitement du signal, commande, régulation, etc. Le défi reste le même, peut-être légèrement déplacé : construire de façon artificielle de l'intelligence et utiliser l'intelligence ainsi construite.

Pendant ce parcours certains ont craint de perdre Descartes, d'autres l'auraient laissé volontiers au bas-côté de la chaussée. Mais lui, d'après Garcia Lorca, « petit comme une amande verte, lassé des cercles et des droites, s'enfuit par les canaux pour entendre chanter les matelots ivres ». Descartes nous faussant compagnie. Certes non ! Mais simple renouvellement du défi. Attrapons au vol ce chant « Quelqu'un m'a raconté que perdu dans les glaces, / Dans un chaos de monts, loin de tout océan, / Il vit passer sans heurt et sans fumée la masse / Immense et pavoisée d'un paquebot géant »[3] et essayons de doter les paquebots géants mais aussi les minuscules robots-jouets avec des systèmes embarqués. Nous nous trouverons ainsi face aux limites scientifiques et techniques actuelles. Car les systèmes embarqués – et l'informatique de demain sera essentiellement une informatique embarquée – il faut qu'ils puissent planifier des actions, qu'ils puissent surmonter des situations imprévisibles, donc être capables de modifier leur programme, et, encore, qu'ils puissent, en cas de panne, de s'auto-réparer. Tâches éminemment complexes qui requièrent de la connaissance.

Ainsi le traitement de la connaissance, qui est l'objet de l'étude de l'I.A., constitue en même temps la composante principale de la science informatique[4]. Comme d'habitude dans les sciences en pareilles circonstances, nous n'avons pas

1. P.Valéry Politique de l'esprit, Œuvres, Col. Pléiade.
2. Plus de 400 ans depuis sa naissance le 21 mars 1596.
3. R. Desnos déjà 60 ans depuis sa mort en 1945 au camp de concentration de Terezine.
4. C'est, d'ailleurs, grâce à cette composante que l'informatique a passé du statut de la technique à celui de la science

une idée claire de ce que c'est l'objet de la science, à savoir la connaissance. Déjà Gaston Berger écrivait en 1941 « Il faut dire (...), en toute rigueur, que la connaissance est indéfinissable » [5]. Il est difficile de savoir si cette pensée est la conséquence de la définition circulaire donnée par Henri Bergson « Si l'instinct et l'intelligence enveloppent, l'un et l'autre, des connaissances, la connaissance est plutôt jouée et inconsciente dans le cas de l'instinct, plutôt pensée et consciente dans le cas de l'intelligence » [6]. Bergson procède à une reduction de la connaissance à l'intelligence. Peut-être en 1907 ceci était-il licite. Mais en 1921 l'intelligence devient un paysage brumeux. Une revue de psychologie demande cette année à quatorze experts de lui fournir une définition de l'intelligence et elle en récolte neuf différentes [7] ! Si on posait la même question aujourd'hui, peut-être aurions-nous plus de définitions que d'experts. Une réponse, assez technique dans son essence, est donnée par Gaston Bachelard qui écrivit « Pour un esprit scientifique, toute connaissance est une réponse à une question. S'il n'y a pas de question, il ne peut y avoir de connaissance scientifique. Rien ne va de soi. Rien n'est donné. Tout est construit. » [8]. C'est donc grâce au sujet agissant et à son action qu'il y a création de la connaissance. Il est permis toutefois de douter que cette belle et très humaniste définition de Bachelard puisse être appliquée à des machines qui, au plus, peuvent espérer au statut des humanoïdes. Néanmoins nous pouvons, en utilisant cette définition et des éléments de la Théorie de l'Information, évaluer le coût et, donc, la valeur de la connaissance. C'est, comme nous le savons déjà, le nombre de questions élémentaires qu'il faut poser pour obtenir la réponse que l'on cherche. Poursuivant dans cette voie nous pouvons aussi citer une récente – et extrêmement intéressante – contribution de Hervé Zwirn. Dans un article [9] il distingue deux types de connaissances. Une qui est une connaissance forte et se produit en ramenant un phénomène inconnu à un phénomène familier qui nous est compréhensible. De cette façon nous connaissons quelque chose parce que nous le comprenons et nous pouvons ainsi dire que nous connaissons une parcelle du monde qui nous entoure. Et une autre connaissance, qui est une connaissance faible et qui ramène un phénomène inconnu à une loi générale qui l'explique. Il est évident que la loi générale nous ne la comprenons pas. Par exemple il est impossible de comprendre la loi de la gravitation. Dans ce cas nous disons que nous connaissons le phénomène parce que nous sommes capables, en utilisant la loi générale à laquelle il obéit, de prédire son évolution. En somme nous connaissons le phénomène qui va se passer avant qui se réalise. Nous effectuons ainsi une contraction du temps. Si nous traduisons

5. G. Berger :Recherches sur les conditions de la connaissance. Essai d'une théorétique pure, Paris, P. U. F., 1941

6. H. Bergson : L'Évolution créatrice, Paris, 1907

7. cf. R.L.Gregory (dir.) : Le cerveau un inconnu, Robert Laffont, Paris, 1993, pp.667-668

8. G. Bachelard : La formation de l'esprit scientifique : contribution à une psychanalyse de la connaissance objective, Vrin, Paris, 1938

9. H. Zwirn : Compréhension et compression, La Recherche, décembre 2002, p.111

cette contraction en termes informatiques, elle revient à être une compression des données. Nous approchons ainsi, par une autre voie, l'idée d'Andrei Kolmogorov concernant la relation entre information et complexité. Selon cette approche, qui est aussi celle de Ray Solomonoff, l'information apportée par un phénomène est fonction de la longueur du code du phénomène, si on transcrit ce phénomène en code binaire. La complexité d'un phénomène est considérée comme étant relative à la longueur du code du phénomène. Si donc nous sommes capables de compresser le code binaire pour le faire exécuter et avoir des résultats avant qu'ils se produisent, nous pouvons dire que nous connaissons le phénomène. Si par contre nous ne pouvons pas effectuer une compression du code, alors nous ne sommes pas capables de connaître le phénomène. C'est ce qui se passe avec la génération des nombres aléatoires et l'évolution des processus chaotiques où nous ne connaissons pas les résultats avant qu'ils se réalisent bien que la ou les lois de leur évolution sont connues. Bien sûr il ne faut pas aller vite en besogne et affirmer que plus le phénomène est moins complexe, moins nous le comprenons plus lentement.

Pendant ce cours nous verrons d'abord les raisons pour lesquelles la logique est un excellent langage pour le traitement de la connaissance. Ensuite nous travaillerons avec la logique du premier ordre avant d'examiner des algorithmes qui conduisent à l'automatisation de l'opération d'inférence logique, opération qui permet la production des connaissances nouvelles. Tout au long de ce cours afin de favoriser la compréhension et la consolidation d'un point important soit de la théorie soit de son application, nous le faisons suivre par des ascèses que les élèves doivent résoudre. Par ailleurs chaque chapitre se termine par des exercices qui permettent aux élèves de faire un tour récapitulatif du contenu du chapitre. De plus de cours du langage Prolog s'intercaleront avec ce cours, afin que l'élève puisse avoir des applications informatiques concrètes des concepts et des méthodes de la Logique computationnelle.

1

INTELLIGENCE, CONNAISSANCES ET LANGAGES

L'*intelligence*, au sens propre du mot, désigne l'ensemble de fonctions mentales qui permettent à un être vivant d'adapter son comportement à son environnement. Elle depend donc de la connaissance que cet être vivant a de son environnement. Elle suppose l'intégration au niveau du système nerveux des fonctions différentes, notamment des fonctions :

- de perception de l'environnement ;
- de reconnaissance des situations ;
- de décision d'actions ;
- de mémorisation.

La tâche fondamentale, du point de vue des applications, pour l'IA est de comprendre l'intelligence afin de la rendre plus productive. Son objectif est donc le suivant :

Étant donné une tâche précise, dont l'exécution par l'homme requiert de l'intelligence, il s'agit de construire un logiciel permettant à un ordinateur d'exécuter la même tâche avec des résultats comparables à ceux obtenus par l'homme.

Deux cas sont à considérer.

Un premier où il existe un algorithme pour la résolution du problème posé. Bien que nous pouvons, sous certains aspects, envisager que cette approche peut faire partie de l'IA, nous la considérons pas ici.

L'autre cas est constitué par des problèmes pour lesquels il n'existe pas d'algorithme de résolution. On cherche alors, à déterminer une stratégie de résolution, pas forcement optimale. On construit ainsi une méthode qui contient un ensemble de règles qui permettent d'effectuer, à différents stades et en fonction de la situation du moment, des choix qui favorisent, mais sans pour autant le garantir, l'aboutissement à une solution. On appelle cette démarche une *heuristique*. La raison d'être de l'IA est fondée sur le postulat suivant :

Toute tâche cognitive peut être accomplie par un ordinateur programmé heuristiquement.

Un des axes majeurs de la démarche heuristique est constitué par la *représentation de connaissances*, à savoir la méthodologie d'organisation d'un vaste répertoire de données, de manière à pouvoir en extraire ce qui est nécessaire et pertinent au traitement d'une situation qui émerge d'un ensemble infini des possibilités. Bien sûr les outils à mettre au point pour accomplir cette tâche, n'imiteront pas nécessairement l'intelligence humaine, mais il faut qu'ils aient des performances non inférieures en précision et rapidité à celles de l'intelligence humaine.

Nous avons vu que l'intelligence humaine se fonde sur l'existence de la connaissance de l'environnement et l'exploitation de cette connaissance. Ce processus se fait de deux façons distinctes :

- une première qui consiste en l'assimilation, mémorisation et structuration des connaissances, et
- une seconde qui consiste en la décomposition et en la particularisation de la connaissance.

Par conséquent pour résoudre, de façon artificielle, un problème il faut d'une part stocker la connaissance et, d'autre part, trouver une méthode qui exploite la connaissance. Du fait que la connaissance

- est volumineuse ;
- n'est pas facile à caractériser précisément ;
- change constamment ;

il faut que la méthode qui exploite la connaissance :

(1) Fasse surgir des généralisations. Dans ce cas il n'est pas nécessaire de représenter séparément chaque situation particulière, car les situations qui ont les mêmes propriétés importantes sont groupées ensemble. Si la connaissance n'a pas cette propriété, il faut plus de place mémoire pour la stocker et plus de temps pour la traiter.

(2) Soit facile à modifier afin de corriger les erreurs et/ou de prendre en compte des modifications.

(3) Puisse être utilisée dans la majorité de cas, même si elle n'est pas totalement précise ou complète.

Remarquons que ce que nous avons mis en évidence sur la connaissance humaine nous pouvons le retrouver sur un ordinateur. En effet pour qu'un programme tourne sur un ordinateur, il lui faut des informations. L'exploitation des informations se fait de deux façons :

- par stockage de l'information ;
- par recherche de l'information.

Ces deux façons d'exploitation des informations sont déjà utilisées par des logiciels algorithmiques opérant sur des données numériques (i.e. les logiciels classiques).

La seule différence, par rapport à ces logiciels, est que l'information relative à la connaissance a, comme nous le verrons au cours de l'Intelligence Artificielle, un caractère symbolique et non pas numérique. Par conséquent la méthode de résolution d'un problème de l'intelligence se traduira par la fabrication d'un logiciel non-algorithmique opérant sur des données symboliques. Le logiciel sera non-algorithmique car la résolution d'un problème d'intelligence ne peut pas être complètement explicitée et donc programmée au sens classique du terme compte tenu du nombre important de choix à effectuer et, de plus, tous les éléments ne sont pas connus a priori. On est donc amené à fournir à de tels logiciels un ensemble d'informations de type symbolique.

Il est bien évident qu'il faut aussi fournir à l'ordinateur les règles de gestion de cet ensemble d'informations symboliques.

La résolution donc, des problèmes dans le cadre de l'IA, s'effectue à l'aide de deux modules :

(1) Un premier qui concerne l'utilisation des informations (méthodes de recherche) : accès aux informations, combinaison des informations entre elles, etc.

(2) Un second module qui est la représentation des connaissances : regroupement des informations au sein d'une base de données, dont la structure tient compte des règles de recherche précédemment établies.

L'IA s'occupe essentiellement du deuxième module. Son objectif est de pouvoir exprimer la connaissance dans une forme qui puisse être utilisée par un ordinateur. Il faut, par conséquent, trouver un langage qui permet d'assimiler, par un ordinateur, la représentation des connaissances. Comme nous le savons, un langage est déterminé en fonction de deux aspects. Un aspect syntaxique qui permet de vérifier que les phrases du langage sont écrites correctement, en respectant les

règles établies de syntaxe du langage. Et un aspect sémantique qui permet de vérifier la signification des phrases du langage. En réalité ce qui nous interesse principalement est l'aspect sémantique, c'est-à-dire le sens d'une phrase. Mais pour y acceder au sens d'une phrase, il faut que cette phrase soit correcte, ce qui nous amène à examiner sa forme syntaxique. Mais il est évident qu'une phrase peut être du point de vue syntaxique correcte et, malgré cela, dépourvue de sens.

Les exigences que nous pouvons formuler à ce stade pour le langage de représentation des connaissances concernent sa capacité d'être concis et sans ambiguïtés. Il faut aussi qu'il soit indépendant du contexte, en ce sens qu'il puisse exprimer des connaissances de toute sorte. Il y a plusieurs langages, surtout informatiques, qui peuvent répondre à ses caractéristiques. Mais il y a une autre exigence, très importante, concernant la représentation des connaissances, c'est le formalisme. Nous ne pouvons pas espérer manipuler efficacement les connaissances et obtenir des nouvelles connaissances si nous ne faisons pas une approche extrêmement formelle. Dès lors il devient évident qu'un des langages les plus appropriés pour la représentation et le traitement des connaissances est la logique mathématique selon une triple démarche :

(1) Utilisation de la logique mathématique pour modéliser et stocker la connaissance.

(2) Utilisation de la logique mathématique comme un langage pour communiquer avec l'ordinateur.

(3) Utilisation de la logique mathématique comme une méthode pour la vérification des programmes.

Le cours de la Logique Computationnelle a comme objectif de fournir aux élèves les bases théoriques de la programmation logique, que nous verrons au cours de Prolog, et qui permet d'un part d'écrire des programmes pour la représentation et le traitement des connaissances et, d'autre part, de procéder à la vérification des programmes.

1.1 Références

Nous donnons ci-après la liste des livres qui nous ont servi pour la redaction de ces notes.

J.-M. ALLIOT, T. SCHIEX : *Intelligence Arificielle et Informatique Théorique*, Cépadués, 1993

K. R. APT & E.-R. OLDEROG : *Verification of sequential and concurrent programs*, Springer, 1991

E. W. BETH : *The Foundations of Mathematics*, North-holland, 1965

GEORGE S. BOOLOS, RICHARD J. JEFFREY : *Computability and logic*, Third edition, Cambridge U.P., 1989

STANLEY N. BURRIS : *Logic for mathematics and computer science*, Prentice Hall, 1998

JEAN CAVAILLÉS : *Méthode Axiomatique et Formalisme*, Hermann, 1981

ALONZO CHURCH : *Introduction to mathematical logic*, Princeton Univ. Press, 1956

RENÉ CORI, DANIEL LASCAR : *Logique mathématique*, 2 volumes, Masson, 1993

DIRK VAN DALEN : *Logic and Structure*, Third edition, Springer, 1994

MICHAEL R. GENESERETH, NILS J. NILSSON : *Logical Foundations of Artificial Intelligence*, M.Kaufmann, 1987

ROBERT LKOWALSKI : *Logic for Problem Solving, North-Holland, 1979*,

J. W. LLOYD : *Fondements de la programmation logique*, Eyrolles, 1988

PER MARTIN-LÖF : *Intuitioniostic Type Theory*, Bibliopolis, 1984

ANIL NERODE, RICHARD A. SHORE : *Logic for applications*, Second edition, Springer, 1997

ULF NILSSON, JAN MAŁUSZYŃSKI : *Logic, Programming and Prolog*, Wiley, 1990

JEAN PIAGET : *Essai de logique opératoire*, Dunod, 1972

HELENA RASIOWA, ROMAN SIKORSKI : *The MMathematics of Metamathematics*, Państwowe Wyd. Nauk., 1963

H. ROGERS, JR : *Theory of recursive functions and effective computability*, MIT Press, 1987

STUART J. RUSSEL, PETER NORVIG : *Artificial Intelligence. A Modern Approach*, Prentice-Hall, 1995

ROBERT I. SOARE : *Recursively enumerable sets and degrees*, Springer-Verlag, 1987

ALFRED TARSKI : *Introduction to Logic and to Methodology of Deductive Sciences*, Galaxy, 1965

R. TURNER : *Logics for artificial intelligence*, E.Horwood, 1984

JACQUES ZAHND : *Logique élémentaire*, Presses Polytechniques et Universitaires Romandes, 1998

2

OBJECTIFS ET MÉTHODES DE LA LOGIQUE

Dans ce premier chapitre nous présentons les objectifs généraux de la logique computationnelle. Comme cette logique est dérivée de la logique mathématique nous présentons d'abord les éléments fondamentaux de celle-ci et en particulier les systèmes de raisonnement qu'elle utilise.

2.1 La logique comme activité humaine

Comme nous venons de voir au chapitre précédent, le comportement intelligent d'un être vivant se resume à sa capacité de s'adapter aux changements de l'environnement. Pour ce faire il y a un prérequis, c'est la connaissance de cet environnement. Sa forme la plus élémentaire est la forme descriptive, c'est-à-dire celle qui présente la connaissance à l'aide des propositions (phrases) déclaratives.

L'objectif de la logique est l'analyse des propositions et l'évaluation de la valeur de vérité de ces propositions. C'est une tâche très difficile, car les langues naturelles ont évolué d'une façon pas toujours conforme avec la logique et ainsi il n'est pas rare d'avoir des phrases qui du point de vue de la forme sont similaires mais du point de vue de la logique sont différentes. Considérons par exemple les deux raisonnements suivants, empruntés à A. Church :

– J'ai vu un portrait de John Wilkes Booth. John Wilkes Booth a assassiné

Abraham Lincoln. J'ai donc vu le portrait de l'assassin d'Abraham Lincoln.

– J'ai vu le portrait de quelqu'un. Quelqu'un a assassiné Abraham Lincoln. J'ai donc vu le portrait d'un assassin d'Abraham Lincoln.

Malgré la similitude de deux propositions il y a une grande différence en ce qui concerne le sens de chacune de ces propositions : la première proposition peut être vraie tandis que la seconde est fausse. Pour éviter des telles situations il faut réduire de façon drastique la richesse de la langue tant du point de vue nombre de mots utilisés que du point de vue variétés de formes employées pour une phrase. L'objectif est de créer un langage artificiel aussi réduit que possible, qui permet d'analyser la validité d'un raisonnement sans réflexion, de façon automatique en utilisant des algorithmes appropriés. On obtient ainsi, pour exprimer les propositions, un langage formel par opposition à la langue naturelle qui, bien sûr, elle est informelle. De plus il faut, à l'aide de ce langage, pouvoir exprimer toute proposition de n'importe quelle situation de toute activité humaine. On doit donc, dans ce langage, remplacer les mots par des symboles qui expriment des objets, des concepts ou, encore, des opérateurs propres à une activité humaine particulière. Pour ce faire, il faut, étant donnée une phrase exprimée dans la langue courante, de séparer, par une démarche d'abstraction, le contenu de la phrase de sa forme. En ne gardant que la forme de la phrase, nous pouvons construire un langage formel qui, en se complétant, sera en mesure d'exprimer la totalité des phrases que nous pouvons formuler en utilisant une langue.

Pour réaliser cette abstraction qui conduit à la formalisation, nous avons besoin de déterminer les ingrédients d'une phrase et aussi leur représentation formelle. Nous distinguons ainsi les éléments suivants pour une phrase :

Les noms Nous avons d'abord les *noms propres* qui peuvent indiquer :

– soit n'importe quelle personne qui porte ce nom, par exemple *Toto* ;
– soit une personne particulière, par exemple *Berlioz*.

Remarquons que dans ce dernier cas on peut faire référence à un nom propre par périphrase, par exemple *le compositeur de la symphonie fantastique*. Cette remarque nous aide à comprendre qu'en logique un nom n'est pris en compte que par ce qu'il *dénote*, donc *Berlioz* et *le compositeur de la symphonie fantastique* dénotent la même personne. Malgré tout, la difficulté n'est pas complètement surmontée, car s'il est loisible (en suivant Bertrand Russel) de s'interroger si *Berlioz est le compositeur de la symphonie fantastique*, il est, par contre, sot de se poser la question si *Berlioz est Berlioz*. Il y a donc quelque chose de plus que la dénotation que véhicule un nom ; c'est son *sens* d'où découlent d'ailleurs, l'existence et l'unicité de la dénotation. [1]

1. Le premier à faire cette distinction fut G.Frege en 1892. Selon lui un terme dénote (fait ré-

Nous avons aussi les *noms communs* qui, à leur tour, possèdent un sens et, par conséquent, dénotent ainsi quelque chose, qu'il s'agisse d'un objet, d'un être vivant ou d'un concept. La langue commune fait que beaucoup des noms communs ont des sens additionnels qui dévient de leur signification originelle. Par exemple le nom commun *poirier* qui, dans sa signification originnelle, dénote un arbre fruitier, ne retrouve pas ce sens quand on dit qu'un enfant fait *le poirier*.

Pour un nom commun la dénotation (ou référence) est le concept qui désigne, tandis que son sens est donné par sa valeur de vérité. Carnap [2] en suivant Frege, propose de considérer comme dénotation d'un mot commun son *extension*, i.e. l'ensemble des objets qui sont désignés par ce mot et comme sens son *intension*, c'est-à-dire la fonction qui permet de savoir si un objet donné peut être référencé par ce mot.

En langage formel nous représenterons les noms propres, qui ont une seule dénotation, par des *constantes* qui seront notées $a, b, c, \ldots$. Nous réserverons les *variables* pour les noms communs ou les noms propres qui ont plusieurs dénotations, par exemple *Toto*. Les variables seront notées $X, Y, Z, \ldots$. Notez l'usage des lettres majuscules. La distinction entre constantes et variables se fait de cette façon : toute constante commence par une lettre minuscule et toute variable par une lettre majuscule. Remarquons que les valeurs numériques sont des constantes.

Les qualités Ce qui est dénoté par un nom, propre ou commun, a des qualités diverses qui s'expriment en utilisant d'autres noms. Ceux-ci sont bien entendu, des valeurs des ces qualités, considérées comme des fonctions sur celui-là. Par exemple *rouge* est la valeur de qualité *couleur* appliquée, comme une fonction, au *petit livre* (de celui que vous savez). Cette fonction a, comme n'importe quelle fonction mathématique, un domaine de définition et un domaine des valeurs qui est son image. Pour des raisons qui s'éclairciront par la suite (cf. paragraphe suivant), nous l'appellerons *foncteur*.

Les propriétés Indépendamment de ses qualités, ce qui est dénoté par un nom, propre ou commun, a aussi des propriétés. Une propriété quelconque appliquée sur un objet soit elle se vérifie, soit non. Elle peut donc être considérée comme une fonction sur l'objet dont l'image se réduit à l'ensemble {vrai, faux}. Une telle fonction s'appelle un *prédicat* ou, encore, *fonction propositionnelle* et qu'il ne faut pas confondre avec le foncteur défini précédemment.
.

Les propositions Une proposition dans la langue courante est un assemblage des mots qui ont un sens et qui expriment une pensée. S'il s'agit d'une affirma-

férence) à un individu particulier. Le sens du terme est la condition qu'un individu particulier doit remplir pour être la dénotation (le référant).

2. R. Carnap : *Meaning and necessity*, Un. Chicago Press, 1956

tion, nous avons une proposition sous forme déclarative. En langage formel la proposition joue le même rôle et nous utilisons uniquement des propositions sous forme déclarative. Dans la mesure où une proposition a un sens, elle peut être vraie ou fausse. Nous pouvons donc lui associer une fonction de vérité qui prend deux valeurs – vrai ou 1 et faux ou 0.

Nous pouvons envisager que les propositions peuvent être représentées par des variables qui prennent deux valeurs, vrai ou faux. Ce sont des *variable propositionnelle* et seront notées par $p, q, \ldots$. Remarquons que la formalisation d'une proposition conduit à la définition d'une variable propositionnelle.

La proposition est la forme la plus achevée du langage que nous utiliserons et, comme nous l'avons déjà indiquer, l'objectif de la logique est d'analyser ces propositions et d'évaluer leur valeur de vérité. Depuis Aristote on sait que cette évaluation dépend plutôt de la forme de la proposition que de son contenu [3]. L'évaluation se fait en utilisant différents types de raisonnement que nous présentons brièvement ci-après :

Raisonnement déductif C'est la forme la plus connue et la seule utilisée par la logique classique. Il permet, en partant d'une loi générale, d'obtenir – de déduire – des faits particuliers. Le fameux exemple

Tous les hommes sont mortels
Socrate est un homme
Donc, Socrate est mortel

est l'archetype du raisonnement déductif et qui de façon formelle s'écrit comme suit :

Tous les A sont B.
C est A.
Donc, C est B.

Pour la compréhension de la suite du chapitre nous transformons l'exemple précédent comme suit :

Tous les êtres vivants sur Terre sont mortels.
Socrate est un être vivant sur Terre.
Donc Socrate est un mortel.

Il faut remarquer que le raisonnement déductif ne permet pas d'obtenir des nouvelles connaissances. Il permet seulement de rationaliser et de codifier une démarche qui consiste à prendre un élément d'une famille et à lui attribuer une propriété que tous les éléments de la famille possèdent.

3. Nous avons déjà rencontré quelque chose d'analogue en Théorie d'Information. Nous avons en effet vu que l'information d'un message ne dépend pas de son contenu mais de la complexité de sa forme. Vous pouvez, en faisant des observations à d'autres disciplines, vous apercevoir que souvent, en Mathématiques, pour pouvoir faire un calcul on doit abandonner la matérialité du contenu pour l'idéalité de la forme.

Raisonnement inductif Ce raisonnement est attribué à Bacon et il suit la démarche inverse du raisonnement déductif. À partir d'un ensemble des faits particuliers qui ont tous un élément en commun, on érige cet élément commun en loi générale. Par exemple :

Ces êtres vivants (Socrate, Héraclite, Parménide, ...) sont sur la Terre.

Ces êtres vivants (Socrate, Héraclite, Parménide, ...) sont mortels.

Donc, tous les êtres vivants qui sont sur la Terre sont mortels.

Sa traduction formelle est la suivante :

$A_1, A_2, \ldots, A_n$ *sont B.*

$A_1, A_2, \ldots, A_n$ *sont C.*

Donc, tout B est C.

Il est évident que pour appliquer ce raisonnement il faut disposer de plusieurs observations. Sa base scientifique est la loi de grands nombres. Néanmoins il faut se rappeler que l'induction permet d'établir des lois expérimentales mais non pas des lois théoriques. Ce raisonnement est donc à la base de la méthode experimentale, c'est-à-dire à la base des sciences (physique, chimie, ...) où une loi reste vraie tant qu'il n'y ait pas un contre-exemple la refutant [4]. En ce sens il permet la production des nouvelles connaissances, qui peuvent éventuellement être potentiellement fausses.

Raisonnement abductif Il relie les effets aux causes et suggère une hypothèse. Par exemple :

Tous les êtres vivants qui sont sur la Terre sont mortels.

Ces êtres vivants (Socrate, Héraclite, Parménide, ...) sont mortels.

Donc, ces êtres vivants (Socrate, Héraclite, Parménide, ...) sont sur la Terrre.

que nous pouvons exprimer de manière formelle :

Tous les A qui sont B sont aussi C.

D est C.

Donc, D est B.

Ce type de raisonnement a été introduit par Peirce qui l'a défini comme étant une adoption probatoire d'une hypothèse. Dans son esprit il s'agissait d'un raisonnement inductif allegé. En effet le raisonnement inductif, en se fondant sur un ensemble d'observations qui ont toutes un élément commun, forge une observation générale relative à cet élément commun et, par conséquent, cette observation générale revêtirait le statut de règle générale permettant ainsi de

4. C'est justement cette remarque qui permet à Karl Popper de distinguer les sciences des croyances, car les premières sont, selon lui, réfutables au contraire des secondes.

tirer des conclusions. Par contre la conclusion d'un raisonnement abductif est une hypothèse choisie dans un ensemble d'hypothèses, en raison de sa plausibilité. Ainsi le fait de vivre sur Terre apparaît comme une cause de la mortalité des êtres vivants. Mais elle est une cause parmi d'autres.

Le raisonnement abductif a été appliqué au calcul logique par Kowalski et Kakas. Ce raisonnement construit aussi des nouvelles connaissances qui ont en réalité un statut très précaire.

Raisonnement par analogie Il s'agit du plus faible type de raisonnement. En effet il ne fournit aucune certitude et il est surtout utilisé en justice pour établir des jugements en exploitant la jurisprudence. Un exemple de raisonnement par analogie est le suivant.

Socrate est un être vivant sur Terre et qui est mortel.

Héraclite est un être vivant sur Terre et qui est comme Socrate.

Donc Héraclite est mortel.

L'aspect formel de ce raisonnement est le suivant

A est P.
B est similaire à A.
Donc, B est P.

Une forme encore plus attenuée du raisonnement par analogie est la suivante :

A et B sont P.
A est S.
Donc, B est S.

Ce raisonnement peut conduire à des erreurs et à des fausses croyances solidement ancrées dans la mesure ou elles sont, à l'instar des conclusions du raisonnement lui-même, non démontrables.

De tous les raisonnements que nous venons de voir, seulement le raisonnement déductif fournit la garantie de la validité du résultat.

2.2 Logiques formelle et computationnelle

La logique est la branche des mathématiques qui étudie les lois de la pensée. Son objectif est de former de règles qui permettent de penser correctement. Transposée à l'informatique, la logique permettrait d'écrire des programmes corrects. Écrire un programme reviendrait ainsi à appliquer un ensemble de règles de la logique.

La *logique formelle* est une version decontextualisée de la logique. En tant que telle elle a :

– un langage formel ;
– une syntaxe sans ambiguïtés ;
– une sémantique précise, et
– des règles de formation des propositions.

Pour formaliser dans ce langage une phrase comme par exemple *Toto aime la logique* on doit utiliser différents types de symboles. Nous présenterons en detail ces symboles dans les deux chapitres suivants.

La représentation d'une phrase par un langage formel correspond à une représentation des connaissances. Si on veut faire un traitement des connaissances, il faut pouvoir faire un raisonnement en utilisant la représentation formelle de ces connaissances et appliquer les différents types de raisonnement. Pour que le traitement soit efficace il faut envisager son automatisation, c'est-à-dire avoir un procédé mécanique qui applique aux phrases de la logique, des règles de raisonnement d'une façon systématique.

La *logique computationnelle* est une branche qui se trouve au croisement de la logique mathématique et de l'informatique et dont son objectif est d'élaborer les bases théoriques pour la construction des tels procédés mécaniques qui, en réalité, sont ici des programmes informatiques. Elle recupère donc toutes les préoccupations de la logique formelle qui sont relatives aux approches syntaxique et sémantique des raisonnements, auxquelles elle rajoute sa propre préoccupation concernant l'efficacité du raisonnement.

Plus particulièrement la logique computationnelle joue un rôle important au *test des modèles* (model-checking) qui est actuellement une de techniques les plus utilisées pour la vérification et le déverminnage des programmes. Dans ce cadre la logique examine ce qui peut être écrit dans un langage formel - et qui fait partie du domaine de la syntaxique - et la façon dont ce qui est écrit, est interprété par un modèle concret - ce qui constitue le domaine de la sémantique.

L'utilisation de la logique est devenu indispensable dès qu'on a introduit l'ordinateur à l'automatisation des diverses tâches. Prenons l'exemple de la conduite d'une rame de métro. Si le conducteur est un être humain, il dispose d'un certain nombre d'actionneurs - frein, vitesse, ouverture/fermeture des portes, ... - sur lesquels agit directement. Si par exemple il rencontre un feu rouge, il appuiera sur le frein. Cette action est conséquence d'une démarche logique qui est issue d'un modèle interne propre au conducteur et qui représente, sous forme symbolique, le monde réel dans lequel se trouve le conducteur et la rame de métro. Si ce modèle interne est faux, le conducteur fera des bêtises, ce qui peut arriver, par exemple, à quelqu'un qui n'a jamais conduit une rame. Et celui qui apprend à conduire, est en train en fait de construire un modèle interne de plus en plus exact. Si on remplace le conducteur par une conduite automatique, on a, en réalité, remplacé le conducteur par un ordinateur qui, muni d'un logiciel, peut, à l'aide d'un certain nombre

de dispositifs, agir sur les actionneurs. Le logiciel doit être la replique exacte du modèle interne du conducteur. Le logiciel est écrit à l'aide d'un langage formel et en construisant des systèmes mathématiques avec des propriétés bien définies. Nous avons ainsi un *modèle logique* de la réalité qui n'est pas le même avec celui du conducteur. Mais, malgré cette différence, il faut, devant une situation donnée, que les deux modèles réagissent de la même façon. Il faut donc savoir si le programme informatique se comporte chaque fois comme il faut qu'il se comporte. L'examen de la conformité du comportement des programmes avec les exigences de son utilisation constitue le *test des modèles*.

3

CALCUL PROPOSITIONNEL

Pour décrire un environnement donné, que nous appellerons par la suite *univers du discours* (noté $\mathcal{U}$), la logique utilise des phrases que nous appellerons *propositions logiques* ou encore *formules*. Le but du calcul logique est de donner un fondement à ces propositions c'est-à-dire examiner leur validité. Cet examen de la validité d'une formule se fera indépendamment de sa structure et le résultat sera en rapport avec sa propriété d'être vraie ou fausse. Nous commençons l'étude de la logique par l'étude des propositions. Nous allons d'abord définir un langage qui permettra la construction des propositions, i.e. un alphabet et des mécanismes qui permettent de créer des propositions. Viendra ensuite l'étude sémantique et syntaxique des propositions créées, c'est-à-dire l'étude du sens et de la preuve des propositions.

3.1 Éléments du langage

Les propositions seront représentées par des symboles qui auront une valeur de vérité : *vraie, fausse*. Pour leur étude logique nous allons définir un *langage formel*

$\mathcal{L}_0$ à l'aide des éléments suivants :

- Un ensemble V_p, au plus dénombrable, des *propositions* qui seront notées par $p, q, \ldots$. Notons que l'« appellation contrôlée » des propositions est *variables propositionnelles* terme que nous n'utiliserons pas, de peur d'introduire une confusion en ce qui concerne le sens de la variable. On pourra par contre utiliser le nom – moins usité – de *propositions atomiques*.
- Un ensemble Ξ, au plus dénombrable, des *constantes*.
- Un ensemble L des *connecteurs* qui sont les suivants :
  - *Connecteur logique* unaire : la négation $\neg$
  - *Connecteurs propositionnels* binaires :
    - Disjonction : $\vee$
    - Conjonction : $\wedge$
    - Implication : $\rightarrow$
    - Équivalence (ou double implication) : $\leftrightarrow$
- Les séparateurs : parenthèses gauche "(" et droite ")", crochets gauche "[" et droit "]".

Les séparateurs ne font pas partie, à proprement parler, du langage. Leur présence permet de faciliter la lecture des propositions.

Les éléments du langage déterminent un *alphabet*

$$\Sigma_0 = \{V_p, \Xi, L\}$$

La brique élémentaire du calcul propositionnel est l'*atome*. Un proposition atomique est un atome. Une constante aussi. Plus généralement

DÉFINITION 3.1.1 *Un* atome *est une proposition dont la structure interne ne nous préoccupe pas.*

Par abus de notation, les atomes seront notés par la suite avec les lettres $p, q, r, \ldots$, c'est-à-dire de la même manière que les propositions bien qu'ils puissent être aussi des constantes. Implicitement on accepte donc qu'une constate puisse être vue comme une proposition, ce qui est d'ailleurs la réalité. Si nous maintenons la distinction entre constates et propositions, que les puristes pourraient nous reprocher, c'est uniquement pour des raisons de compatibilité avec le contenu du chapitre suivant.

À partir des atomes on peut construire des *formules bien formées* (fbf) dont la définition est la suivante :

DÉFINITION 3.1.2 *Une* formule bien formée *est*

- *soit un atome*
- *soit une proposition obtenue à partir des fbf A et B, selon les constructions suivantes :*

- $\neg A$
- $A \vee B$, $A \wedge B$
- $A \rightarrow B$, $A \leftrightarrow B$

Selon cette définition, les propositions atomiques sont des fbf. Les formules bien formées seront notées par les lettres $A, B, C, \ldots$.

Si on note par F_0 le plus petit ensemble de fbf que nous pouvons construire selon la définition 3.1.2, alors la paire

$$\mathcal{L}_0 = \{\Sigma_0, F_0\}$$

est le *langage d'ordre zéro* ou le *langage du calcul propositionnel*. Par construction F_0 est est un ensemble dénombrable, défini récursivement.

Les principales propriétés des connecteurs sont données ci-après :

(1) Double négation ou involution

$$p \leftrightarrow \neg\neg p$$

(2) Loi de de Morgan

$$\neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$$
$$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$$

(3) Définition de l'implication

$$(p \rightarrow q) \leftrightarrow (\neg p \vee q)$$

(4) Introduction de l'implication

$$p \rightarrow (q \rightarrow p)$$

(5) Distributivité de l'implication

$$(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$$

(6) Contradiction

$$(p \rightarrow q) \rightarrow ((p \rightarrow \neg q) \rightarrow \neg p)$$

Étant donnée une proposition, nous pouvons en construire trois autres qui dérivent de celle-là comme suit :

DÉFINITION 3.1.3 *Considérons la proposition $P_1 : p \rightarrow q$.*
La proposition $P_2 : \neg q \rightarrow \neg p$ est la proposition contrapositive *de P_1.*
La proposition $P_3 : q \rightarrow p$ est la proposition inverse *(ou* réciproque*) de P_1.*
La proposition $P_4 : \neg p \rightarrow \neg q$ est la négation *de P_1.*

ASCÈSE 3.1 *Soient les deux propositions :*

p_1 *Aujourd'hui on est le 13 du mois.*

p_2 *Demain on sera le 14 du mois.*

Exprimer en logique des propositions et en français les quatre propositions : directe, contra-positive, inverse et négation.

3.2 Interprétation sémantique – Modèles

La première tâche du calcul propositionnel est de donner un sens aux fbf, c'est-à-dire d'établir une correspondance entre les fbf et les faits de l'univers du discours. Ces faits sont connus par celui qui fournit la fbf. En effet, il faut admettre qu'en logique une fbf ne signifie, par elle-même, rien. Elle a un sens – si elle en a un – en fonction de l'*interprétation* faite par celui qui émet cette fbf.

Nous devons donc, pour évaluer le sens d'une fbf, établir seulement la valeur de vérité de cette fbf étant donnée l'interprétation adéquate. Pour ce faire, nous adoptons l'hypothèse fondamentale suivante :

Tout atome peut prendre deux valeurs : vrai – 1 et faux – 0 et la valeur de vérité d'une fbf est complètement déterminée par la valeur de chacun de ses atomes.

Concrètement à chaque atome d'une fbf on associe une valeur de vérité, selon l'interprétation que l'on donne à cet atome, et à chaque connecteur on associe une table de vérité fixe qui, en fonction de la valeur de vérité de ses arguments, fournit la valeur de vérité de l'opération effectuée par le connecteur. Dans la table ci-après nous donnons les valeurs de vérité pour tous les connecteurs du langage.

p	q	$\neg p$	$p \vee q$	$p \wedge q$	$p \rightarrow q$	$p \leftrightarrow q$
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

L'utilisation des tables de vérité pour le calcul de la valeur d'une fbf est une méthode sémantique. On verra par la suite des méthodes syntaxiques.

ASCÈSE 3.2 *Vérifier si la formule*

$$p \rightarrow (q \vee (r \leftrightarrow (r \rightarrow \neg p)))$$

est une fbf et établir sa valeur.

ASCÈSE 3.3 *Établir une propsition $p(a,b,c)$ qui a la même valeur de vérité que la majorité de ses arguments.*

La valeur de vérité associée à un atome peut changer en fonction de la signification de cet atome. Une interprétation d'une fbf correspond à une spécification de la signification de chaque atome de la fbf. Formellement

DÉFINITION 3.2.1 *Considérons un ensemble d'atomes propositionnels noté Δ. Nous dirons que Δ forme une* base de données (BdD). *On appelle* valuation *de l'ensemble des variables de la BdD Δ, une fonction $\varphi : \Delta \to \{0,1\}$*
Si A est une fbf, on note par Δ_A l'ensemble de variables propositionnelles de A et si $card\, A = n$, alors une valuation de A, qui sera notée $\varphi(\Delta_A)$ est un élément de $\{0,1\}^n$.

La notion de la valuation permet de définir celle de l'interprétation :

DÉFINITION 3.2.2 *Considérons une fbf A et soit Δ_A la BdD de ses atomes propositionnels. Une* interprétation *I de la fbf A est une valuation de Δ_A, c'es-à-dire $\varphi_I : \Delta_A \to \{0,1\}$.*
Cette valuation sera notée $\varphi_I(\Delta_A)$ ou, encore, plus brièvement, $\varphi_I(A)$ et elle est un élément de $\{0,1\}^{card\, A}$.

Si Δ contient n variables propositionnelles, alors on a au plus 2^n valuations possibles. Si la valuation d'une interprétation particulière φ_I a la valeur 1, on dit que la valuation satisfait A et l'on note $\varphi_I(A) = 1$. Il est évident que toute valuation d'une fbf A n'est pas susceptible de satisfaire à A, c'est-à-dire de conférer à A la valeur de vérité « vrai ». Nous avons ainsi :

DÉFINITION 3.2.3 *Soit A une fbf et I une interprétation. On dit que A est* satisfiable *par I ou que A est une* conséquence sémantique *de I, et l'on note $I \models A$, si l'une de situations suivantes est vérifiée :*

- *si $A \in V_p$, alors $I \models A$ ssi $\varphi_I(A) = 1$*
- *si A est de la forme $(\neg B)$, alors $I \models A$ ssi $I \not\models B$* [1]
- *si A est de la forme $(B \wedge C)$, alors $I \models A$ ssi $I \models B$ et $I \models C$*
- *si A est de la forme $(B \vee C)$, alors $I \models A$ ssi $I \models B$ ou $I \models C$*
- *si A est de la forme $(B \to C)$, alors $I \models A$ ssi soit $(I \not\models B)$, soit $(I \models C)$*
- *si A est de la forme $(B \leftrightarrow C)$, alors $I \models A$ ssi soit $(I \not\models B)$ et $(I \not\models C)$, soit $(I \models B)$ et $(I \models C)$*

ASCÈSE 3.4 *Considérons les fbf suivantes :*

(1) $p \wedge q$

(2) $p \wedge (\neg r \vee q)$

(3) $p \to q$

1. $\not\models$ signifie non satisfiable et il est un symbole extra-logique.

(4) $q \rightarrow p$

(5) $(p \vee q) \wedge (q \vee r)$

et l'interprétation I suivante :

$$\phi_I(p) = 1,\ \phi_I(q) = 0,\ \phi_I(r) = 1$$

Donner la valeur de vérité des formules précédentes selon l'interprétation I.
Existe-t-il une interprétation qui rende toutes les formules vraies ?

Nous introduisons maintenant la notion du modèle pour une fbf.

DÉFINITION 3.2.4 *Soit A une fbf et I une interprétation. Si I rend A satisfiable, c'est-à-dire si $I \models A$, alors I est un* modèle *pour A que l'on note par $M(A)$. On dit aussi que A est vraie dans I.*
On note par $\mathcal{M}$ un ensemble de modèles pour A. Alors $\forall I \in \mathcal{M}$ on a $I \models A$ ce qui permet de noter $\mathcal{M} \models A$.

De cette définition on peut en conclure que si I est un modèle pour A, alors dans la table de vérité pour A, la ligne qui correspond à la valuation φ_I aura la valeur 1 à la colonne correspondante à A.

La notion du modèle donnée avec la définition précédente pour une fbf, peut être étendue à un ensemble des fbf F. On dit que l'interprétation I est un modèle pour F si I est un modèle pour chaque fbf de F.

ASCÈSE 3.5 *Considérons une situation selon laquelle une alarme d'une maison se met en fonctionnement, si elle n'est pas en panne, quand il y a un tremblement de terre ou un cambriolage ou les deux à la fois.*

(1) Exprimer le déclenchement de l'alarme à l'aide de la logique propositionnelle.

(2) Trouver, s'il en existe, un modèle pour la formule établie précédemment.

(3) Pour chacune des situations suivantes, trouver, s'il en existe, en modèle :

- *(a) Tremblement de terre.*
- *(b) Cambriolage $\longrightarrow \neg$Cambriolage*
- *(c) (Cambriolage $\rightarrow \neg$Cambriolage) $\wedge$ Cambriolage*

Nous introduisons maintenant quatre notions qui découlent de l'interprétation.

DÉFINITION 3.2.5 *Une fbf A qui est vraie pour toute interprétation est appelée* tautologie *(ou* formule valide*) et sera notée par $\models A$.*
Une fbf A pour laquelle il y a au moins une interprétation I qui la satisfait (c'est-à-dire que I est un modèle pour A) est appelée satisfiable *ou* (sémantiquement) consistante.
Une fbf A pour laquelle il existe une interprétation I telle que $I \not\models A$ est appelée falsifiable.

Une fbf qui est fausse dans toute interprétation est appelée insatisfiable *ou* sémantiquement inconsistante [2].

Les algorithmes qui à partir d'un ensemble des fbf engendrent des tautologies s'appellent *systèmes de preuve* ou *prouveurs*. Un prouveur est *cohérent* s'il engendre seulement des tautologies. Il est *complet* s'il engendre toutes les tautologies.

Pour un ensemble de fbf, nous avons la définition suivante :

DÉFINITION 3.2.6 *Un ensemble des fbf est* mutuellement exclusif *si et seulement si chaque interprétation satisfait au plus à une fbf.*
Un ensemble des fbf est exhaustif *si et seulement si chaque interprétation satisfait au moins à une fbf.*

ASCÈSE 3.6 *(1) Montrer que l'ensemble de fbf $\mathcal{F} = \{p \wedge q,\ q \vee r\}$ est satisfiable.*

(2) Montrer que l'ensemble de fbf $\mathcal{F} = \{p \wedge q,\ q \wedge r,\ \neg r\}$ est insatisfiable.

Dans le cas où A est une tautologie, dans la table de vérité de A, la colonne qui correspond à A contient seulement de 1.

Remarquons que si A est une tautologie, alors $\neg A$ est sémantiquement inconsistante et vice-versa.

ASCÈSE 3.7 *Pour les fbf suivantes préciser leur nature (tautologie, satisfiable ou insatisfiable).*

(1) $p \rightarrow (q \rightarrow p)$

(2) $q \wedge (q \rightarrow p) \rightarrow p$

(3) $\neg p \rightarrow (p \vee q)$

(4) $p \wedge \neg(p \vee q)$

Deux fbf sont équivalentes si pour toute interprétation elles prennent la même valeur de vérité. La tautologie permet de formaliser la relation d'équivalence entre deux fbf :

DÉFINITION 3.2.7 *Deux fbf A et B sont* équivalentes, *et l'on note par $A \equiv B$, si et seulement si la fbf $A \leftrightarrow B$ est une tautologie.*

L'équivalence, $A \equiv B$, entre deux fbf est une relation d'équivalence en ce sens que : $A \equiv B \rightarrow B \equiv A$ et $A \equiv B \wedge B \equiv C \rightarrow A \equiv C$.

Les équivalences permettent des traitements sémantiques qui préservent la sémantique.

ASCÈSE 3.8 *Soit une fbf P et soit P' la proposition contrapositive de P. Montrer que P' est équivalente à P.*

2. Certains auteurs préfèrent le terme *antilogie* pour ce type de fbf.

On trouve dans la littérature une liste impressionnante des tautologies.Voici un extrait des tautologies célèbres :

(1) Loi du syllogisme

$$(p \to q) \to ((q \to r) \to (p \to r))$$

(2)

$$(p \to (q \to r)) \to ((p \to q) \to (p \to r))$$

(3) Introduction de *ou*

$$p \to (p \vee q) \ , \ q \to (p \vee q)$$

(4)

$$(p \to r) \to ((q \to r) \to ((p \vee q) \to r))$$

(5) Élimination de *et*

$$(p \wedge q) \to p \ , \ (p \wedge q) \to q$$

(6)

$$(r \to p) \to ((r \to q) \to (r \to (p \wedge q)))$$

(7) Loi de l'importation

$$(p \to (q \to r)) \to ((p \wedge q) \to r)$$

(8) Loi de l'exportation

$$((p \wedge q) \to r) \to (p \to (q \to r))$$

(9) Loi de Duns Scotus

$$(p \wedge \neg p) \to q$$

(10)

$$p \to (p \wedge \neg p) \to \neg p$$

(11) Loi du tiers exclu (tertium non datur)

$$p \vee \neg p$$

(12) Introduction de l'absurde $\perp$

$$(p \wedge \neg p) \to \perp$$

(13) Réduction à l'absurde

$$(\neg p \to \perp) \to p$$

(14)

$$(p \to q) \to ((p \to \neg b) \to \neg p)$$

L'exemple classique de fbf insatisfiable est le suivant :

$$p \leftrightarrow \neg p$$

ASCÈSE 3.9 *Vérifier que*

$$\neg (q \rightarrow p) \equiv q \wedge \neg p$$

Nous donnons dans la suite une liste numérotée avec des formules équivalentes. Toutes les formules du même numéro sont équivalentes.

(1) $p \rightarrow q$, $\neg p \vee q$, $\neg q \rightarrow \neg p$, $p \wedge q \leftrightarrow p$, $p \wedge q \leftrightarrow q$

(2) $\neg (p \rightarrow q)$, $p \wedge \neg q$

(3) $p \leftrightarrow q$, $(p \wedge q) \vee (\neg p \wedge \neg q)$, $(\neg p \vee q) \wedge (p \vee \neg q)$, $(p \rightarrow q) \wedge (q \rightarrow p)$, $\neg p \leftrightarrow \neg q$, $(p \vee q) \rightarrow (p \vee q)$

(4) $\neg (p \leftrightarrow q)$, $p \leftrightarrow \neg q$, $\neg p \leftrightarrow q$

(5) p , $\neg \neg p$, $p \wedge p$, $p \vee p$, $p \vee (p \wedge q)$, $p \wedge (p \vee q)$, $\neg p \rightarrow p$, $(p \rightarrow q) \rightarrow p$, $(q \rightarrow p) \wedge (\neg q \rightarrow p)$

(6) $\neg p$, $p \rightarrow \neg p$, $(p \rightarrow q) \wedge (p \rightarrow \neg q)$

(7) $p \wedge q$, $q \wedge p$, $p \wedge (\neg p \vee q)$, $\neg (p \wedge \neg q)$

(8) $p \vee q$, $q \vee p$, $p \vee (\neg p \wedge q)$, $\neg a \rightarrow q$, $(p \rightarrow q) \rightarrow q$

(9) $p \rightarrow (q \rightarrow r)$, $(p \wedge q) \rightarrow r$, $q \rightarrow (p \rightarrow r)$, $(p \rightarrow q) \rightarrow (p \rightarrow r)$

(10) $p \rightarrow (q \wedge r)$, $(p \rightarrow q) \wedge (p \rightarrow r)$

(11) $p \rightarrow (q \vee r)$, $(p \rightarrow q) \vee (p \rightarrow r)$

(12) $(p \wedge q) \rightarrow r$, $(p \rightarrow r) \vee (q \rightarrow r)$

(13) $(p \vee q) \rightarrow r$, $(p \rightarrow r) \wedge (q \rightarrow r)$

(14) $p \leftrightarrow (q \leftrightarrow r)$, $(p \leftrightarrow q) \leftrightarrow r$

Les lignes numérotées 10 et 11 montrent qu'il y ait distributivité à gauche de l'implication par rapport à la conjonction et la disjonction et les deux lignes suivantes montrent que cette distributivité disparaît lorsque l'implication passe à droite.

On termine ce paragraphe par la notion de conséquence valide.

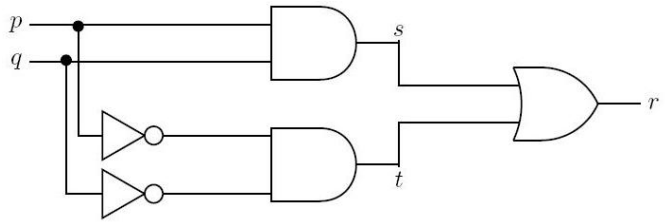
DÉFINITION 3.2.8 *Soit la fbf B dont les atomes font partie des atomes des fbf $A_1, A_2, \ldots, A_n$. B est une* conséquence valide *des $A_1, A_2, \ldots, A_n$, que l'on note par $A_1, A_2, \ldots, A_n \models B$, si B prend la valeur* 1 *quand tous les A_i, qui contiennent des atomes de B, sont simultanément à* 1.

On dit aussi que $A_1, A_2, \ldots, A_n$ impliquent logiquement B. De manière plus simple on peut dire que $A \models B$ ssi tout modèle de A est aussi modèle de B. Si, de plus, on a que tout modèle de A est aussi un modèle de B et tout modèle de B est un modèle de A, alors A et B sont équivalents : $A \equiv B$.

Dans la suite il faut faire attention au fait que le même symbole, à savoir $\models$, est utilisé pour indiquer deux relations différentes :

- la relation de satisfiabilité qui relie une interprétation à une fbf ;
- la relation de conséquence valide qui relie deux fbf entre elles.

ASCÈSE 3.10 *Considérons le circuit suivant*



Montrer que ce circuit vérifie la proposition

$$[(p \vee q) \rightarrow \neg(\neg p \vee \neg q)] \rightarrow r$$

Nous avons le

THÉORÈME 3.2.1 *$A \models B$ est équivalent à $\models (A \rightarrow B)$.*

ASCÈSE 3.11 *Traduire et évaluer la phrase suivante :*

Tu ne dis pas où tu te trouves et si Toto ne le dit pas, alors Koko le dira si et seulement si on lui promet de lui acheter une glace.

L'ascèse suivant introduit l'analyse logique d'un programme.

ASCÈSE 3.12 *Soient les deux programmes suivants :*

Programme 1

```
Si (x > 0) et (y > 0) alors
     afficher ('Produit x*y positif')
Finsi
```

Programme 2

```
Si (x > 0) alors
    si (y > 0) alors
        afficher ('Produit x*y positif')
    Finsi
Finsi
```

Montrer que ces deux programmes sont sémantiquement équivalents.

3.3 Modèles et connaissances

Nous pouvons utiliser les modèles $\mathcal{M}(A)$ d'une fbf A pour obtenir des connaissances sur l'univers du discours. Nous avons les résultats suivants :

- $\mathcal{M}(A \wedge B) = \mathcal{M}(A) \cap \mathcal{M}(B)$
- $\mathcal{M}(A \vee B) = \mathcal{M}(A) \cup \mathcal{M}(B)$
- $\mathcal{M}(\neg A) = \mathcal{M}(A)^C$

Nous avons aussi que

- La fbf A est satisfiable ssi $\mathcal{M}(A) \neq \emptyset$.
- Soient deux fbf A et B. Nous avons $A \rightarrow B$ ssi $\mathcal{M}(A) \subset \mathcal{M}(B)$.
- A est équivalente à B ssi $\mathcal{M}(A) = \mathcal{M}(B)$
- Les fbf $A_1, \ldots, A_n$ sont mutuellement exclusives ssi $\mathcal{M}(A_i) \cap \mathcal{M}(A_j) = \emptyset$; $\forall i \neq j$.

De plus nous pouvons utiliser les modèles pour examiner la sémantique d'une fbf. Supposons par exemple que nous avons une fbf A et que nous connaissons les modèles $\mathcal{M}(A)$ pour A. Si notre connaissance est enrichie d'une nouvelle fbf B, alors nous connaissons les modèles $\mathcal{M}(A \wedge B) = \mathcal{M}(A) \cap \mathcal{M}(B)$ qui est une connaissance plus précise que la précédente. Nous arrivons à un état complet de connaissance si toutes les interprétations sont fausses, sauf une qui constitue le seul modèle correspondant aux fbf utilisées.

ASCÈSE 3.13 *Considérons la situation de l'ascèse 3.5. Notre état de connaissances sont les huit interprétations possibles. Si on nous donne la fbf*

$$A \ : \ (\textit{Tremblement de terre} \ \wedge \ \textit{Cambriolage}) \ \rightarrow \ \textit{Alarme}$$

que devient notre état de connaissances ?

Supposons qu'une nouvelle fbf

$$B \ : \ \textit{Tremblement de terre} \ \rightarrow \ \textit{Cambriolage}$$

De quelle manière évolue notre état de connaissances ?

Nous avons le

THÉORÈME 3.3.1 *Soient $\mathcal{M}$ et $\mathcal{M}'$ deux ensembles de modèles tels que $\mathcal{M}' \subseteq \mathcal{M}$. Si $\mathcal{M} \models A$, alors $\mathcal{M}' \models A$.*

ASCÈSE 3.14 *Appliquer le théorème précedent à l'ensemble des modèles :*

	p	q
v_1	0	0
v_2	0	1
v_3	1	0
v_4	1	1

et vérifier qu'en restreignant l'ensemble de modèles on devient capable d'inférer des propositions plus fortes (plus restrictives).

3.4 Évaluation syntaxique – Démonstration

L'évaluation syntaxique est un procédé mécanique qui permet de déduire le bien fondé d'une formule indépendamment du sens de ses composantes. Pour ce faire on s'appuie sur un ensemble d'axiomes et des règles d'inférence. Considérée de cette façon, l'évaluation syntaxique est une théorie de la démonstration. La première notion de la théorie de démonstration est le théorème dont voici sa définition.

DÉFINITION 3.4.1 *Une fbf A est un* théorème*, et l'on note* $\vdash A$ *, si A est un axiome ou si A est obtenue par application des règles d'inférence sur d'autres théorèmes.*

Les axiomes de la logique sont les principes fondamentaux de la logique, c'est-à-dire des propositions du langage qui sont vraies en vertu uniquement de leur syntaxe. Le calcul propositionnel possède trois axiomes :

A1.- Introduction de l'implication

$$A \rightarrow (B \rightarrow A)$$

A2.- Distributivité de l'implication

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

A3.- Négation

$$(\neg B \rightarrow \neg A) \rightarrow (A \rightarrow B)$$

où, encore

$$(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$$

Remarquons que ces axiomes sont en fait des schémas d'axiomes, en ce sens qu'à chaque tel schéma correspond une infinité d'axiomes. Par exemple pour le schéma A1, en remplaçant A par $A \rightarrow C$ et B par $B \rightarrow C$, on peut avoir l'axiome : $(A \rightarrow C) \rightarrow ((B \rightarrow C)(A \rightarrow C))$.

Les règles d'inférence sont les règles utilisées par le mécanisme de démonstration. Le calcul propositionnel, et en particulier Prolog, utilise la règle d'inférence suivante :

R1.- Modus ponens (règle du détachement)

$$\frac{\vdash A, \vdash A \rightarrow B}{\vdash B}\ {}^{3}$$

3. La notation $\frac{\vdash A}{\vdash B}$ est utilisée à la place de $(\vdash A) \vdash (\vdash B)$ pour indiquer qu'il s'agit d'une règle d'inférence et non pas d'une proposition.

Pour faciliter les calculs on peut aussi utiliser les règles suivantes :

R2.- Modus tollens (l'inverse de modus ponens) ou, encore, résolutio, unitaire

$$\frac{\vdash A \rightarrow B, \vdash \neg B}{\vdash \neg A}$$

R3.- Élimination de « ET »

$$\frac{\vdash A \wedge B}{\vdash A, \vdash B}$$

R4.- Introduction de « ET »

$$\frac{\vdash A, \vdash B}{\vdash A \wedge B}$$

R5.- Introduction de « OU »

$$\frac{\vdash B}{\vdash A \vee B \vee C \cdots}$$

R6.- Loi de l'involution (Double négation)

$$\frac{\vdash \neg\neg A}{\vdash A}$$

R7.- Double résolution

$$\frac{\vdash A \vee \vdash B,\ \vdash \neg B \vee C}{\vdash A \vee C} \text{ ou, de façon équivalente } \frac{\vdash \neg A \rightarrow B,\ \vdash B \rightarrow C}{\vdash \neg A \rightarrow C}$$

Nous pouvons déterminer un *système d'inférence* ou *système de déduction* comme étant constitué d'un ensemble d'axiomes et d'un ensemble de règles d'inférence, c'est-à-dire par le couple $\mathcal{S} = (\mathcal{A}, \mathcal{R})$.

Si on se limite aux trois axiomes (A1)-(A3) et la règle (R1) du modus ponens, nous avons le système d'inférence de Hilbert, noté $\mathcal{H}$. Dans ce système, on peut établir le théorème :

THÉORÈME 3.4.1 *On a*

$$\vdash A \rightarrow A$$

Dans ce système nous avons aussi la règle (controversée par les intuitionnistes) de *reductio ad absurdum* qui s'enonce comme suit :

$$Si\ A \vdash \neg B \rightarrow \bot,\ alors\ A \vdash B$$

Cette règle est à la base du théorème de réfutation (cf. infra) qui est la pierre angulaire du mécanisme de démonstration utilisé par Prolog.

Une *théorie* est constituée par la donnée d'un système d'inférence et de tous les théorèmes qui peuvent être obtenus par le système d'inférence.

Nous sommes maintenant en mesure de définir la démonstration et la déduction.

DÉFINITION 3.4.2 *Soit un théorème A. Une* démonstration *de A est une suite finie* $(A_1, A_2, \ldots, A_n, A)$ *où chaque* A_i *est soit un axiome, soit le résultat d'une règle d'inférence appliquée sur des éléments* A_j *précédemment obtenus (c'est-à-dire* $j < i$*).*

ASCÈSE 3.15 *Démontrer la fbf*

$$\vdash (A \rightarrow A)$$

DÉFINITION 3.4.3 *Une fbf A est une* déduction *de l'ensemble de fbf* $B_1, B_2, \ldots, B_n$*, que l'on note* $B_1, B_2, \ldots, B_n \vdash A$*, s'il existe une suite finie* $(A_1, A_2, \ldots, A_n, A)$ *où chaque* A_i *est soit un axiome, soit un des* B_i *soit il est obtenu par application d'une règle d'inférence sur des éléments* A_j *précédemment obtenus.*
Les fbf B_i *sont appelées des* hypothèses.

3.5 Équivalence entre modèles et théorie de démonstration

Nous allons examiner la relation qu'il existe entre l'interprétation sémantique d'une fbf et sa démonstration syntaxique. Nous avons vu que pour établir la validité d'une formule l'interprétation sémantique utilise des éléments qui ne font pas nécessairement partie du langage de la logique. Par contre la démonstration syntaxique utilise de méthodes de la logique pour élaborer la démonstration d'une formule.

Le problème que nous avons ici est que les résultats de l'interprétation sémantique ne doivent pas être en contadiction avec les résultats de la démonstration syntaxique. Il faut donc que, pour une formule donnée, $\vdash f$ entraîne $\models f$ (adéquation de la logique) et aussi que $\models f$ entraîne $\vdash f$ (complétude de la logique). Plus formellement, nous avons :

DÉFINITION 3.5.1 *Une logique est* adéquate *si tout théorème* $\vdash A$ *est une formule valide* $\models A$*.*

Une logique est syntaxiquement consistante *s'il n'existe aucune formule du langage telle que* $\vdash A$ *et* $\vdash \neg A$*.*

Une logique est (faiblement) complète *si toute formule valide est un théorème, c'est-à-dire* $(\models A) \rightarrow (\vdash A)$*.*

L'équivalence cherchée est obtenue à l'aide des trois théorèmes suivants :

THÉORÈME 3.5.1 *Le calcul propositionnel est adéquat :* $(\vdash A) \rightarrow (\models A)$*.*

THÉORÈME 3.5.2 *Le calcul propositionnel est syntaxiquement consistant.*

THÉORÈME 3.5.3 *Le calcul propositionnel est (faiblement) complet :* $(\models A) \rightarrow (\vdash A)$*.*

Les théorèmes 3.5.1 et 3.5.3 illustrent le fait qu'en calcul propositionnel, toute tautologie est un théorème et vice-versa, ce qui exprime le théorème suivant :

THÉORÈME 3.5.4 *Si $\vdash p$, alors p est une tautologie et vice-versa.*

3.6 Quelques méta-théorèmes

Dans les paragraphes précédents nous avons introduit un certain nombre de concepts concernant les fbf considérées individuellement. Nous allons étendre ces notions à un ensemble E des fbf. Nous avons ainsi :

DÉFINITION 3.6.1 *Un ensemble E de fbf est* insatisfiable *ou* sémantiquement inconsistant *si et seulement s'il n'existe aucune interprétation I telle que chaque fbf A de E soit satisfiable par I.*

Cette définition permet d'établir le théorème de la réfutation :

THÉORÈME 3.6.1 (Th. de la réfutation) *Une fbf A est* conséquence valide *d'un ensemble E de fbf, c'est-à-dire $E \models A$, si et seulement si $E \cup (\neg A)$ est insatisfiable.* [4]

Si on note par $\perp$ la fbf qui est toujours fausse, on déduit qu'un ensemble de fbf est insatisfiable si et seulement s'il a comme conséquence la formule $\perp$. Il s'ensuit donc, que toute vérification de la validité d'un ensemble de fbf peut se réduire à la preuve de son inconsistance sémantique.

Cette remarque permet d'établir une autre méthode pour la vérification d'une fbf. Jusqu'ici la seule méthode que nous avons examiné était fondée sur les tables de vérité. La preuve par réfutation est une autre méthode qui a permis d'obtenir des algorithmes très éfficaces concernant la preuve des fbf.

Nous donnons ci-après quelques métathéorèmes [5] supplémentaires du calcul propositionnel en commençant par la partie sémantique. Nous avons :

THÉORÈME 3.6.2 (Th. de la déduction) $A_1, A_2, \ldots, A_n \models B$ *ssi* $A_1, A_2, \ldots, A_{n-1} \models (A_n \rightarrow B)$.

THÉORÈME 3.6.3 (Th. de finitude) *Soit F un ensemble fini ou dénombrable de fbf et A une fbf. Si $F \models A$, alors il existe un ensemble fini $F' \subset F$ tel que $F' \models A$.*

THÉORÈME 3.6.4 (Th. de monotonie) *Soient F_1, F_2 des ensembles finis de fbf. Si $F \models A$, alors $\{F_1, F_2\} \models A$.*

Du point de vue syntaxique nous avons :

4. Une autre forme équivalente (et utile) de ce théoreme est la suivante : Si $F \vee A$ est inconsistante, alors $F \vdash \neg A$

5. Les métathéorèmes sont des théorèmes sur les théorèmes

THÉORÈME 3.6.5 (Th. de la déduction) $A_1, A_2, \ldots, A_n \vdash B$ *ssi* $A_1, A_2, \ldots, A_{n-1} \vdash (A_n \to B)$.

THÉORÈME 3.6.6 (Th. de transitivité) *Si* $C \vdash A_1, \ldots, C \vdash A_n$ *et si* $(A_1, \ldots, A_n) \vdash B$, *alors* $C \vdash B$.

THÉORÈME 3.6.7 (Th. d'échange) *Soit A une sous-formule de la fbf F. Si F est un théorème et si* $A \leftrightarrow B$ *est un théorème, alors la formule obtenue en remplaçant dans F une occurrence de A par B est un théorème.*

THÉORÈME 3.6.8 (Th. de substitution) *Soit S un théorème, x une proposition ayant au moins une occurrence dans S et A une fbf. Alors* $S[x/A]$ *est un théorème.*

THÉORÈME 3.6.9 (Th. de monotonie) *Soient* F_1, F_2 *deux fbf. Si* $F \vdash A$ *,alors* $F_1, F_2 \vdash A$.

THÉORÈME 3.6.10 (Contraposition) $F \wedge A \vdash \neg B$ *ssi* $F \wedge B \vdash \neg A$

Nous donnons maintenant un méta-théorème qui part de la sémantique pour aboutir à la syntaxe.

THÉORÈME 3.6.11 (Règle T) *Si* $F \models A_1, \cdots, F \models A_n$ *et* $A_1, \ldots, A_n \models A$ *, alors* $F \vdash A$.

ASCÈSE 3.16 *Inférer la proposition*

$$(p \to q, q \to r) \vdash p \to r$$

- *sans utiliser le th. de déduction ;*
- *en utilisant le th. de déduction.*

Remplacer les hypothèses d'une conséquence sémantique par d'autres plus simples peut parfois être utile. Le théorème d'interpolation de Craig fournit une réponse à cette préoccupation.

THÉORÈME 3.6.12 (Th. d'interpolation) *Soient A et B deux fbf telles que* $\models A \to B$. *Alors il existe une fbf C composée uniquement par des propositions qui sont communes à A et à B et telle que* $\models A \to C$ *et* $\models C \to B$.

Le théorème suivant dû à E.Beth, affirme qu'en logique propositionnelle une définition implicite peut toujours avoir une forme explicite. C'est une technique applicable à l'intelligence artificielle et qui permet d'obtenir des informations explicites à partir des informations exprimées implicitement.

THÉORÈME 3.6.13 (Th. de définissabilité) *Soit A une fbf ne contenant pas q et r. Si la fbf* $(A(p/q) \wedge A(p/r)) \to (q \leftrightarrow r)$ *est une tautologie, alors il existe une fbf B ne contenant pas* p, q *et r et telle que* $A \to (p \leftrightarrow B)$ *soit une tautologie.*

Nous allons finir cete section avec le théorème de compacité qui assure la possibilité, étant donné un ensemble infini d'hypothèses (i.e. des fbf) qui induit une fbf, d'en extraire un sous-ensemble fini qui induit la même fbf.

DÉFINITION 3.6.2 *Un ensemble infini de fbf est* finiment consistant *si tous ses sous-ensembles finis sont consistants.*

Un ensemble finiment consistant est maximal *s'il n'existe pas un sur-ensemble qui est finiment consistant.*

Concrètement si F est un ensemble infini de fbf et si pout tout $A \subset F$, avec A fini, il existe un modèle $M(A)$ de A, alors F est finiment consistant.

De plus étant donnée une proposition p, si on a $p \notin F$, alors $\neg p \in F$.

Nous avons les deux théorèmes suivants :

THÉORÈME 3.6.14 *Tout ensemble finiment consistant maximal est consistant et admet un modèle unique.*

THÉORÈME 3.6.15 (Th. de la compacité) *Tout ensemble finiment consistant est consistant.*

3.7 Arborescences sémantiques

Afin de représenter une fbf nous pouvons utiliser une arborescence [6] qui est une structure particulière de la théorie des graphes.

Les définitions de la théorie des graphes qui seront utilisées ici sont les suivantes (cf. *C. Berge : Théorie des graphes et ses applications, Dunod, 1959*) :

Un *graphe* est un couple $G = (X, \Gamma)$, où X est un ensemble d'éléments, appelés *sommets* du graphe, et Γ est une application de X dans X. Une paire (x, y) d'éléments de X avec $y \in \Gamma(x)$ est appelée *arc* du graphe et sera noté par u et l'ensemble des arcs d'un graphe sera noté U. D'où une autre définition d'un graphe comme étant le couple $G = (X, U)$.

Une suite $c = [u_1, u_2, \ldots, u_n]$ d'arcs tels que l'extrémité terminale de chaque arc coï ncide avec l'extrémité initiale de l'arc suivant, est appelé *chemin*. Si dans un chemin le sommet initial x_1 coï ncide ave le sommet terminal x_n, alors nous avons un *circuit*.

Un graphe fini $G = (X, U)$ est une *arborescence de racine* $x_0 \in X$ si :

1^0 $\forall x \in X$, $x \neq x_0$ est l'extrémité terminale d'un seul arc ;

2^0 x_0 n'est l'extrémité terminale d'aucun arc ;

3^0 G ne contient pas de circuits.

6. Notons que l'habitude en Intelligence Artificielle est d'appeler arbre l'arborescence, à cause d'une fâcheuse simplification de la terminologie de la théorie des graphes.

Les sommets qui ne sont pas l'extrémité initiale d'un arc quelconque s'appellent *sommets terminaux* ou *feuilles*.

On introduit d'abord la définition suivante :

DÉFINITION 3.7.1 *Étant donné un atome p, un* littéral *relatif à cet atome est soit p, soit $\neg p$.*
On note par $[p] = \{p, \neg p\}$ [7].

L'arborescence sémantique d'une fbf F composée des atomes $p_1, p_2, \ldots, p_n$ est construite de la façon suivante :

- Pour chaque atome $p \in F$, les deux éléments de $[p]$ sont deux sommets distincts de l'arborescence.
- Deux arcs dont l'extrémité terminale du premier est le sommet p et et du second le sommet $\neg p$, ont leur extrémité initiale commune.
- Aucun chemin ne comporte plus d'une occurrence de chaque atome.

L'arborescence sémantique est complète si chaque chemin contient une et une seule fois chaque atome. Elle est partielle si chaque chemin contient au plus une fois chaque atome.

Il est clair que l'arborescence sémantique complète de n atomes contient 2^n sommets terminaux. Les tables de vérité conduisent à l'examen de ces 2^n sommets terminaux et par conséquent leur utilisation est complètement inefficace dès que n dépasse la valeur de 5.

3.8 Formes clausales

Afin de construire des méthodes plus efficaces pour la preuve de la validité d'une fbf, nous allons utiliser la notion de la clause.

DÉFINITION 3.8.1 *Une* clause *C est une disjonction de littéraux $p_1 \vee p_2 \vee \cdots \vee p_n$.*
La clause vide sera notée par $\perp$. Il s'agit d'une fbf toujours fausse.

DÉFINITION 3.8.2 *Une conjonction de clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ est une* forme conjonctive normale (fcn).
Une fcn est parfois notée sous forme ensembliste $\{C_1, C_2, \ldots, C_n\}$.

On peut penser à transformer l'examen de la satisfiabilité d'une fbf en un examen de la validité d'une fcn, en espérant que ce dernier soit plus facile à réaliser. Néanmoins il faudrait qu'on puisse passer de la fbf à la fcn. Le théorème suivant assure le bien fondé de cette démarche.

THÉORÈME 3.8.1 (Théorème de normalisation) *Toute fbf peut se transformer à une fcn qui est logiquement équivalente.*

7. $[p]$ est un symbole extra-logique.

L'algorithme de la transformation d'une fbf en une fcn est le suivant :

(1) Élimination du connecteur $\leftrightarrow$: $(A \leftrightarrow B) \equiv (A \rightarrow B) \wedge (B \rightarrow A)$

(2) Élimination du connecteur $\rightarrow$: $(A \rightarrow B) \equiv (\neg A \vee B)$

(3) Transfert de la négation au niveau le plus intérieur (i.e. devant les atomes) par utilisation des formules :
   - des lois de de Morgan $\neg(A \wedge B) \equiv \neg A \vee \neg B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$ en tant que règles de réécriture ;
   - de la loi de l'involution : $\neg\neg A \equiv A$ qui permet la suppression des doubles négations.

(4) Application de la distributivité
   - de $\vee$ par rapport à $\wedge$: $A \vee (B \wedge C) \equiv (A \vee B) \wedge (B \vee C)$
   - de $\wedge$ par rapport à $\vee$: $A \wedge (B \vee C) \equiv (A \wedge B) \vee (B \wedge C)$

   en tant que règles de réécriture.

La formule obtenue au terme de cet algorithme est une fcn, équivalente à la formule initiale.

L'examen de la validité d'une fcn est fondé sur les trois remarques suivantes :
- Une fcn vide est consistante.
- Une fcn contenant la clause absurde $\perp$ est inconsistante.
- On réduit une fcn contenant deux clauses opposées en $\perp$, i.e. $A \wedge \neg A = \perp$.

ASCÈSE 3.17 *Appliquer l'algorithme ci-dessus à la fbf*

$$p \longleftrightarrow (q \rightarrow r)$$

Un type particulier de clause et – opératoirement – très utilisé est la clause de Horn.

DÉFINITION 3.8.3 *Une clause de Horn est une clause qui a une des trois formes suivantes :*

Clauses de Horn strictes (règles)	$p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow q$
Clauses de Horn négatives (questions)	$p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow$
Clauses de Horn positives (faits)	$\rightarrow q$

Donc une clause de Horn est une clause qui a au plus un atome positif.

Les clauses de Horn strictes sont équivalentes à $\{p_1, p_2, \ldots, p_n\} \models q$. Les clauses de Horn négatives sont équivalentes à $\{p_1, p_2, \ldots, p_n\} \models \perp$. Les clauses de Horn positives sont équivalentes à $\models q$.

Notons que la clause vide $\rightarrow$ sans atome ni à gauche, ni à droite, est la seule clause inconsistante.

ASCÈSE 3.18 *Vérifier si les formules suivantes sont des clauses de Horn :*

(1) $(p \wedge q \wedge r \rightarrow p) \wedge (q \wedge s \rightarrow p) \wedge (p \wedge r \rightarrow r)$

(2) $(p \wedge q \wedge r \rightarrow \bot) \wedge (q \wedge s \rightarrow p) \wedge (\top \rightarrow r)$

(3) $(p \wedge q \wedge r \rightarrow \neg p) \wedge (q \wedge s \rightarrow p) \wedge (p \wedge r \rightarrow r)$

(4) $(p \wedge q \wedge r \rightarrow \bot) \wedge (\neg q \wedge s \rightarrow p) \wedge (\top \rightarrow r)$

Actuellement, un de problèmes majeurs de l'industrie du logiciel est la vérification, du point de vue des spécifications, du code écrit. Une voie de résolution est de transformer ce problème à un problème de satisfiabilité propositionnelle (PSAT) qui consiste à chercher un modèle pour une fbf donnée. Une procédure exhaustive pour résoudre le PSAT en utilisant des fcn est de tester systématiquement tous les éléments de la fcn. S'il y n atomes différents dans la formule, alors le problème s'appelle n-SAT et il faut faire 2^n tests. Il y a de cas spéciaux : 2-SAT avec complexité polynômial et 3-SAT avec complexité NP-complète.

L'importance des fcn vient aussi du fait que elle peut être utilisée pour démontrer des théorèmes en logique propositionnelle. En effet pour prouver A, on peut la réduire en sa fcn. Si cette forme prend la valeur vraie, alors A est aussi vraie car la transformation en fcn préserve l'équivalence logique. En effet, supposons que la fcn de A est $A_1 \wedge A_1 \wedge \cdots \wedge A_n$. Si A est valide, alors chaque A_i l'est aussi. Supposons que A_i est une disjonction des littéraux $L_1 \vee \cdots \vee L_{m_i}$. On cherche à savoir s'il y a une interprétation qui rend faux tous les littéraux L_j. Une telle interprétation existe sauf s'il y a deux littéraux L_j et L_k différents et tels que $L_j = \neg L_k$. Dans ce cas $L_1 \vee \cdots \vee L_{m_i} = \top$.

ASCÈSE 3.19 *Vérifier, en utilisant la reduction à la fcn si les fbf suivantes sont des théorèmes.*

(1) $(p \rightarrow q) \rightarrow ((q \rightarrow r) \rightarrow (p \rightarrow r))$

(2) $(p \vee q) \rightarrow (q \vee r)$

Cette méthode d'évaluation de la valeur de vérité d'une fbf est une méthode syntaxique. Elle est plus facilement mécanisable que les tables de vérité, mais elle souffre du même inconvénient que ces dernières : temps de calcul exponentiel.

3.9 Algorithmes pour le calcul propositionnel

Le problème de la satisfiabilité d'une fbf F à partir des hypothèses $A_1, A_2, \ldots, A_n$ avec n fini, se décline de plusieurs façons :

- La fbf F est une conséquence valide des fbf $A_1, A_2, \ldots, A_n$) ?
- La fbf $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow F$ est une formule valide ?
- $(A_1, A_2, \ldots, A_n, \neg F)$ est un ensemble de formules insatisfiable ?

Cette pluralité d'approches du problème a donné naissance à une multitude d'algorithmes concernant la preuve de la validité d'une fbf.

Dans la suite nous présenterons les algorithmes suivants :

- Algorithme de Quine (arborescences sémantiques).
- Algorithme de réduction (tableaux sémantiques).
- Algorithme de Davis et Putnam (arborescences sémantiques).
- Algorithme de résolution (règle de la réfutation).

Tous les algorithmes qui suivent utilisent la fbf :

$$(\omega) \qquad A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow F$$

dont on cherche à prouver la validité.

3.9.1 Algorithme de Quine

Il est fondé sur les arborescences sémantiques mais il évite leur développement complet.

L'algorithme de Quine consiste en l'élaboration de l'arborescence sémantique étape par étape.

- À chaque sommet p ou $\neg p$ d'une branche de l'arborescence :
  - On procède à la substitution $\sigma = (p/1, \neg p/0)$.
  - On évalue la formule réduite $\omega' = \omega\sigma$.
  - Si ω' se réduit à une valeur de vérité (0 ou 1), arrêt de l'exploration de la branche.
    Sinon on poursuit l'exploration de la branche.
- Le processus s'arrête :
  - avec succés, lorsque on a obtenu un résultat sur une branche avec valeur de vérité à 1 et sur une autre branche avec valeur de vérité à 0
  - avec échec, lorsque on a parcouru toutes les branches.

ASCÈSE 3.20 *Appliquer l'algorithme de Quine à la fbf*

$$((p \rightarrow q) \wedge p) \rightarrow q$$

3.9.2 Algorithme de réduction

Cet algorithme utilise la technique de la preuve par l'absurde. Il consiste à supposer que la fbf à démontrer soit fausse, c'est-à-dire que la fbf $(A_1, A_2, \ldots, A_n, \neg F)$ est insatisfiable, et à arriver, en appliquant l'algorithme, à une contradiction.

- On construit une table avec deux colonnes : 1 et 0.
- On met la fbf (ω) dans la colonne 0 (i.e. on suppose que (ω) est fausse).
- On décompose (ω) en sous-formules qu'on place soit à la colonne 1, soit à la colonne 0, selon les règles suivantes :
  - Si on a $p \rightarrow q$ dans la colonne 0, alors on place p dans la colonne 1 et q dans la colonne 0.
  - Si on a $p \wedge q$ dans la colonne 1, alors on place p et q dans la colonne 1.
  - Si on a $p \vee q$ dans la colonne 0, alors on place p et q dans la colonne 0.

– Si on arrive à avoir dans les deux colonnes, 0 et 1, la même formule, alors on est en présence d'une contradiction (la même formule est à la fois vraie et fausse !) et on peut s'arrêter car la fbf (ω) est vraie.

ASCÈSE 3.21 *Appliquer l'algorithme de la réduction à la fbf*

$$((p \wedge q) \rightarrow r) \rightarrow (p \rightarrow (q \rightarrow r))$$

3.9.3 Algorithme de Davis - Putnam

Soit C une fcn et p un atome. Alors C en tant qu'ensemble est partionné en trois sous-ensembles :

– C_p ensemble des clauses contenant p ;
– $C_{\neg p}$ ensemble des clauses contenant $\neg p$;
– $C_R = C - (C_p \cup C_{\neg p})$

D'autre part on note par

– C_p^- l'ensemble des clauses de C_p privées de p.
– $C_{\neg p}^-$ l'ensemble des clauses de $C_{\neg p}$ privées de $\neg p$.

L'algorithme est fondé sur les deux propriétés suivantes :

PROPRIÉTÉ 3.9.1 *Si p est vraie, alors l'ensemble C équivaut à $C_{\neg p}^- \cup C_R$.*

PROPRIÉTÉ 3.9.2 *Si p est fausse, alors l'ensemble C équivaut à $C_p^- \cup C_R$.*

L'algorithme est le suivant :

(1) Si $C = \perp$, alors C est satisfiable.
(2) Si $C = \{\perp\}$, alors C est insatisfiable.
(3) Sinon
   (a) Choisir une proposition $p \in C$
   (b) Calculer $C_p, C_{\neg p}$ et C_R.
   (c) Calculer C_p^- et $C_{\neg p}^-$.
   (d) Si les deux ensembles $C_{\neg p}^- \cup C_R$ et $C_p^- \cup C_R$ sont insatisfiables, alors C est insatisfiable.

Nous pouvons nous apercevoir que cet algorithme à chaque étape :

– supprime de la fcn sous examen un atome ;
– décompose la fcn sous examen en deux fcn plus simples dans la mesure où il n'y figure plus l'atome supprimé.

ASCÈSE 3.22 *Examiner si l'implication suivante*

$$\{p \rightarrow q,\ q \rightarrow r\} \models (p \vee q) \rightarrow r$$

est valide.

3.9.4 Algorithme de résolution

Soit C une fcn. Si cette fcn est insatisfiable, alors il est toujours possible d'obtenir, à partir de C, une contradiction qui, sous forme clausale, équivaut à la clause vide. Ainsi si on veut automatiser la procédure de la détermination de l'insatisfiabilité, on peut envisager une méthode qui produirait des conséquences à partir de la fcn sous test et qui s'arrêterait dès que la clause vide serait engendrée.

Il est évident que cette technique peut aussi s'appliquer quand on veut établir qu'une formule est une conséquence logique d'un ensemble de formules, i.e. si on veut établir que $C \models A$. En utilisant le théorème de réfutation on sait que $C \models A$ ssi $C \cup \{\neg A\}$ est insatisfiable.

L'algorithme de résolution par réfutation est le suivant :

(1) Étant données la forme clausale C et la fbf A pour montrer que $C \models A$ on construit une nouvelle formule $C' = C \cup \{\neg A\}$.

(2) On transforme C' en fcn C''

(3) On applique la résolution à C'' . Si la résolution engendre la clause vide, alors C' est insatisfiable et, donc, A est une conséquence logique de C.

Dans le cas où C'' est composé des clauses de Horn, nous pouvons rendre l'algorithme de résolution plus efficace en procédant comme suit :

(1) Si la clause vide $\perp$ est dans C'', alors l'ensemble est insatisfiable et la résolution s'arrête.

(2) Sinon

   (a) Choisir dans C'' deux clauses :
      – Une clause positive de Horn A_p réduite à p (si possible).
      – Une clause de Horn $A_{\neg p}$ contenant $\neg p$.

   (b) Calculer l'ensemble $A^-_{\neg p} = A_{\neg p} - \{\neg p\}$c'est-à-dire l'ensemble des clauses de $A_{\neg p}$ privées de la clause $\neg p$.

   (c) Remplacer C'' par $C'' = (C'' - A_{\neg p}) \cup A^-_{\neg p}$ et recommencer l'algorithme.

Notons que si l'ensemble C'' initial ne contient pas des clauses de Horn positives réduites à un seul élément, il faut en produire, en utilisant e.g. le modus ponens.

ASCÈSE 3.23 *Examiner si l'implication suivante*

$$\{p \to q, q \to r\} \models (p \vee q) \to r$$

est valide.

3.10 Exercices

EXERCICE 3.1 *On se place dans l'univers des tribunaux de justice. Considérons la phrase*

Si Toto est innocent d'un crime, alors il n'est pas suspect.

(1) Écrire la proposition contrapositive et calculer sa valeur de vérité.

(2) Calculer la valeur de vérité de la phrase donnée.

(3) Écrire la proposition inverse et calculer sa valeur de vérité.

(4) Écrire la négation de cette phrase et calculer sa valeur de vérité.

EXERCICE 3.2 *Vérifier si*

$$\{p \to q, r \to p \wedge q\} \models r \to q$$

.

EXERCICE 3.3 *Considérons l'opérateur logique* nand dont la table de vérité est la suivante :

p	q	p nand q
1	1	0
1	0	1
0	1	1
0	0	1

(1) Trouver une proposition équivalente à $\neg p$ en utilisant uniquement le connecteur nand.

(2) Trouver une proposition équivalente à $p \vee q$ en utilisant uniquement le connecteur nand.

(3) Trouver une proposition équivalente à $p \wedge q$ en utilisant uniquement le connecteur nand.

EXERCICE 3.4 *Soit l'opérateur logique $\oslash$ nor dont la table de vérité s'exprime comme suit :*

p	q	$p \oslash q$
1	1	0
1	0	0
0	1	0
0	0	1

Déterminer tous les connecteurs logiques en utilisant seulement $\oslash$.

EXERCICE 3.5 *soit la fbf*

$$A = (Q \wedge R) \to (P \leftrightarrow (\neg Q \vee R))$$

dans laquelle P, Q, R sont des variables propositionnelles.

(1) *Déterminer une formule équivalente à A, écrite en utilisant uniquement les connecteurs* $\rightarrow$ *et* $\leftrightarrow$.

(2) *Donner pour A la forme normale disjonctive la plus réduite.*

(3) *Montrer que les formules*

$$B = R \rightarrow (Q \rightarrow (P \leftrightarrow (Q \rightarrow R))) \text{ et } C = R \rightarrow (Q \rightarrow P)$$

sont équivalentes.

EXERCICE 3.6 *On additionne deux nombres binaires à deux digits ab et cd. Le résultat est un nombre à trois digits au plus : pqr. Utiliser le calcul propositionnel pour obtenir une expression de* p,q,r *en fonction de* a,b,c,d.

EXERCICE 3.7 *Donner la fcn de la fbf*

$$(\neg p \wedge q) \rightarrow (p \wedge (r \rightarrow q))$$

EXERCICE 3.8 *Pour effectuer le test de bon fonctionnement d'un procédé de fabrication on vérifie la présence des quatre symptômes :* S_1, S_2, S_3 *et* S_4. *Lors des différents tests, nous avons observé que :*

- *Si* S_1 *et* S_2 *sont présents, alors un de* S_3, S_4 *est aussi présent.*
- *Si* S_2 *et* S_3 *sont présents, alors soit aucun de* S_1, S_4 *n'est présent, soit tous les deux sont présents.*
- *Si aucun de* S_1 *et* S_2 *n'est présent, alors aucun de* S_3, S_4 *n'est non plus présent.*
- *Si aucun de* S_3 *et* S_4 *n'est présent, alors aucun de* S_1, S_2 *n'est non plus présent.*

Faire le travail suivant :

(1) *Traduire les observations en langage des propositions.*

(2) *Montrer que les deux propositions*

   (a) *Les trois symptômes* S_1, S_2 *et* S_3 *ne peuvent pas être présents en même temps.*

   (b) *Si les symptômes* S_1, S_2 *ne sont pas présents, alors* S_3 *n'est pas non plus présent.*

sont valides, en utilisant

- *les tables de vérité ;*
- *l'algorithme de Davis-Putnam ;*
- *l'algorithme de résolution, et*
- *le système de démonstartion des propositions.*

4

CALCUL DES PRÉDICATS

Pour pouvoir utiliser la logique en tant que mode de calcul il faut d'abord procéder à une conceptualisation de la partie du monde qui nous intéresse, c'est-à-dire établir les objets du monde et leurs inter-relations. Il s'agit, en réalité, de procéder à une *représentation de la connaissance*. Les objets sur lesquels nous allons exprimer notre connaissance constituent l'*univers du discours*. La représentation de la connaissance est la mise en relation des plusieurs objets de l'univers du discours par l'intermédiaire de la formulation des fbf.

Avec le calcul propositionnel on ne peut pas formaliser toutes les connaissances relatives à un univers du discours. En particulier on ne peut pas construire une fbf correspondante à une proposition générale du type par exemple « tous les hommes sont mortels » ou particulariser des propositions générales comme, par exemple, « il existe des élèves qui sont fous de la logique ».

Ce type de proposition est un cas particulier de ce qu'on pourrait considérer comme étant des *fonctions propositionnelles logiques* et qui sont des fonctions au sens classique du terme mais dont le résultat est une valeur de vérité - 0 (faux) ou 1 (vrai). Ces fonctions propositionnelles logiques on les appellera, par la suite,

prédicats.

Nous pouvons aussi envisager l'existence des fonctions dont le résultat est un objet de l'univers du discours et non pas une valeur de vérité. Ce sont donc des fonctions au sens habituel du terme et pour les distinguer des autres fonctions qui sont les prédicats, nous les appellerons *foncteurs*.

Comme nous avons fait au chapitre précédent, il faut développer un langage qui permettra la création et l'étude des prédicats. C'est l'objet du présent chapitre. On débute avec la définition du langage. On continue avec les évaluations sémantique et syntaxique d'une fbf avant d'aborder leur équivalence. On termine avec la définition des formes clausales.

4.1 Les éléments du langage

Pour l'étude des prédicats nous allons définir un langage formel $\mathcal{L}_1$ dont l'alphabet A_1 est composé des éléments suivants :

- Un ensemble **V**, au plus dénombrable, des variables qui seront notées $x, y, \ldots$ ou $X, Y, \ldots$.
- Un ensemble Ξ, au plus dénombrable, des constantes qui seront notées $a, b, \ldots$.
- Un ensemble **F** de fonctions $f : \mathbf{V} \times \Xi \rightarrow \mathbf{V} \cup \Xi$ d'arité quelconque[1]. On appelle ces fonctions des *foncteurs* que l'on notera $f, g, \ldots$ ou $F, G, \ldots$.
- Un ensemble **P** de fonctions d'arité quelconque $f : \mathbf{V} \times \Xi \rightarrow \{Vrai, Faux\}$ qui seront appelées *prédicats* et qui seront notés par $p, q, \ldots$ ou $P, Q, \ldots$. Un prédicat particulier est la relation d'égalité qui sera notée soit de façon fonctionnelle $eg(x, y)$, soit, lorsqu'il n'y a pas risque de confusion, $x = y$[2].
- Un ensemble **L** des connecteurs et quantificateurs :
  - Connecteur logique unaire : la négation $\neg$
  - Connecteurs propositionnels binaires :
    - Conjonction : $\wedge$
    - Disjonction : $\vee$
    - Implication : $\rightarrow$
    - Équivalence (ou double implication) : $\leftrightarrow$
  - Quantificateurs :
    - Quantificateur existentiel : $\exists$

1. L'arité d'une fonction est le nombre d'arguments de la fonction. Remarquons qu'une constate est une fonction d'arité zéro.

2. Il faut clairement distinguer entre *égalité* "$=$" et *équivalence* "$\equiv$". La égalité entre deux termes est un prédicat qui indique si oui ou non les deux termes ont la même valeur. L'équivalence entre deux fbf indique que les deux fbf ont la même table de vérité. La question peut se poser si deux termes égaux sont équivalents. La réponse est non. Par exemple nous avons "10= 3+7" mais par rapport au prédicat *nombre pair* il n'y a pas d'équivalence. De même deux fbf équivalentes ne sont pas égales. Par exemple la fbf $a \vee \neg a$ est équivalente à la fbf $a \vee (a \rightarrow b)$ car toutes les deux sont des tautologies, mais elles ne sont pas égales

  - Quantificateur universel : $\forall$
- Les séparateurs (symboles auxiliaires) : $($, $)$, $[$, $]$.

Les séparateurs ne font pas partie, à proprement parler, du langage. Leur présence permet de faciliter la lecture des formules.

Les éléments du langage déterminent un alphabet $A_1 = \{\mathbf{V}, \Xi, \mathbf{F}, \mathbf{P}, \mathbf{L}\}$.

La brique élémentaire du calcul des prédicats est le *terme*. Une variable est un terme. Une constante aussi. Plus généralement

DÉFINITION 4.1.1 *Un* terme *est défini de façon itérative comme suit :*

- *une constante est un terme ;*
- *une variable est un terme ;*
- *si f est un foncteur d'arité n et $t_1, t_2, \ldots, t_n$ sont des termes, alors $f(t_1, t_2, \ldots, t_n)$ est un terme.*

Tout terme est obtenu par application des règles précédentes un nombre fini de fois. L'ensemble de termes sera noté par $\mathbb{T}$.

On peut construire, à partir des termes, des formules bien formées (fbf).

DÉFINITION 4.1.2 *Soit $\mathbb{T}$ l'ensemble des termes sur un alphabet A_1. L'ensemble $\mathbb{F}$ des* formules bien formées *(par rapport à A_1) est le plus petit ensemble tel que :*

- *Si p est un prédicat d'arité n et $t_1, t_2, \ldots, t_n$ sont des termes, alors $p(t_1, t_2, \ldots, t_n) \in \mathbb{F}$.*
- *Si F , $G \in \mathbb{F}$ alors les constructions suivantes :*
  - $\neg F$
  - $F \vee G$, $F \wedge G$
  - $F \rightarrow G$, $F \leftrightarrow G$

  sont aussi des fbf.
- *Si $F \in \mathbb{F}$ et X est une variable, alors $(\forall X) F \in \mathbb{F}$ et $(\exists X) F \in \mathbb{F}$.*

ASCÈSE 4.1 *Donner l'équivalent en français de deux fbf suivantes :*

(1) $\exists X \, (p(X) \wedge (\forall Y \, p(Y) \rightarrow X = Y))$

(2) $(\exists X \, p(X)) \wedge (\forall X \forall Y \, p(X) \wedge p(Y) \rightarrow X = Y)$

L'équivalent de l'atome est donné par la définition suivante :

DÉFINITION 4.1.3 *Si p est un prédicat d'arité n et $t_1, t_2, \ldots, t_n$ sont des termes, alors $p(t_1, t_2, \ldots, t_n)$ est une* formule atomique.

DÉFINITION 4.1.4 *Le triplet*

$$\mathcal{L}_1 = \{A_1, \mathbb{T}, \mathbb{F}\}$$

est le langage d'ordre un *ou le* langage du calcul des prédicats

Par construction $\mathbb{F}$ est un ensemble dénombrable, défini itérativement.

ASCÈSE 4.2 *On se place dans l'ensemble des nombres réels. Écrire, en utilisant le langage des prédicats, les propositions suivantes :*

(1) Pour tout réel il existe un autre réel qui est plus grand.

(2) Il existe un réel qui est plus grand que tout autre réel.

(3) Chaque réel positif est un carré.

(4) Si un réel est plus petit qu'un autre réel, alors il existe un troisième réel, différent de deux précédents, et qui est entre les deux.

ASCÈSE 4.3 *Pour les propositions de l'ascèse précédent, déterminer les prédicats et les foncteurs utilisés.*

ASCÈSE 4.4 *Soit les textes suivants :*

(1) Tous les hommes sont mortels. Socrate est un homme. Donc, Socrate est mortel.

(2) Toute personne saine d'esprit peut comprendre la logique. Aucun des fils de Socrate ne peut comprendre la logique. Aucune personne non saine d'esprit a le droit de vote. Donc, aucun des fils de Socrate n'a le droit de vote.

Pour chaque texte

(1) Écrire la fbf correspondante.

(2) Montrer que chaque fbf est valide.

Les quantificateurs ont une *portée*.

DÉFINITION 4.1.5 *La* portée *d'un quantificateur dans une formule est la partie de la formule qui se trouve sous l'influence du quantificateur.*
Une variable d'une formule qui est sous la portée d'un quantificateur de la même variable est appélée variable liée *(ou* bornée*). Sinon elle est une* variable libre*.*
Une formule qui n'a pas des variables libres s'appelle formule close*. Une formule qui n'a pas des variables s'appelle* formule filtrée*.*
Une formule sans quantificateurs s'appelle formule ouverte*.*

ASCÈSE 4.5 *Pour les fbf ci-après indiquer les variables libres et les variables liées.*

(1) $\exists y\ (p(r) \wedge f(x,y) \vee q(y) \wedge f(x,y)) \leftrightarrow p(r) \vee q(x) \vee p(y)$

(2) $\forall x \exists y\ (p(w) \vee q(z)) \vee \exists z\ (q(w) \vee p(r))$

(3) $\forall x \exists y\ (p(w) \vee q(z)) \vee \exists z\ (q(w) \vee p(y))$

(4) $\forall x (\forall z p(y) \wedge \neg (q(z) \wedge s(y)) \rightarrow \exists y (q(y) \vee p(x) \vee s(z))) \wedge (p(z) \wedge s(y))$

Pour résoudre les problèmes de l'ambiguï té entre démonstration sous forme conditionnelle et démonstration sous forme générale, on introduit la définition ci-après :

DÉFINITION 4.1.6 *Soit F une fbf avec variables libres $x_1, x_2, \ldots, x_n$.*
La clôture universelle *de F , notée $\forall F$, est la formule close $\forall x_1, \forall x_2, \ldots, \forall x_n\ F$.*
La clôture existentielle *de F, notée $\exists F$, est la formule close $\exists x_1, \exists x_2, \ldots, \exists x_n\ F$.*

Notons qu'il est obligatoire quand on passe d'une fbf F à une clôture (soit universelle, soit existentielle) de renommer, par substitution, les variables qui ont le même nom mais qui ne dépendent pas du même quantificateur afin d'éliminer les ambiguïtés. Par exemple pour la fbf $\forall x \exists y\,(p(x) \wedge q(y)) \vee (\exists y \neg p(x) \wedge q(y))$ on doit renommer la deuxième occurrence de x et de y et obtenir ainsi $\forall x \exists y\,(p(x) \wedge q(y)) \vee (\exists z \neg p(x) \wedge q(z))$.

ASCÈSE 4.6 *On se place dans le cadre des nombres entiers et on suppose que nous disposons des opérations (foncteurs) d'addition ($+$) et de multiplication ($\times$). Donner, en langage des prédicats, la défintion des prédicats suivants :*

(1) $pair\,(x)$ = x est un nombre pair.

(2) $div\,(x, y)$ = x divise y, c'est-à-dire que y/x est un nombre entier.

(3) $premier\,(x)$ = x est un nombre premier.

ASCÈSE 4.7 *En appliquant la définition des prédicats pair et div, écrire le prédicat $div2(x)$ = 2 divise le nombre pair x.*

Les principales propriétés des connecteurs sont données ci-après :

(1) Double négation

$$p \leftrightarrow \neg\neg p$$

(2) Loi de de Morgan

$$\neg\,(p \vee q) \leftrightarrow \neg p \wedge \neg q$$
$$\neg\,(p \wedge q) \leftrightarrow \neg p \vee \neg q$$

(3) Définition de l'implication

$$(p \to q) \leftrightarrow (\neg p \vee q)$$

(4) Introduction de l'implication

$$p \to (q \to p)$$

(5) Distributivité de l'implication

$$(p \to (q \to r)) \to ((p \to q) \to (p \to r))$$

(6) Contradiction

$$(p \to q) \to ((p \to \neg q) \to \neg p)$$

(7) Négation des quantificateurs universel et existentiel :

$$\neg(\forall X) \leftrightarrow (\exists X) \ , \ \neg(\exists X) \leftrightarrow (\forall X)$$

Il est important de se rappeler que nous avons pour les négations les relations suivantes :

$$(\neg\forall X \ p(X)) \leftrightarrow (\exists X \ \neg p(X))$$

et

$$(\neg\exists X \ p(X)) \leftrightarrow (\forall X \ \neg p(X))$$

ASCÈSE 4.8 *Traduire en langage des prédicats dans* $\mathbb{N}$ *la phrase suivante :*

(1) $x \equiv y$ *modulo* n.

4.2 Substitution

Pendant le traitement d'une fbf, les variables peuvent être remplacées (substituées) par des termes ou, encore, par des fbf.

DÉFINITION 4.2.1 *Une* substitution *est un ensemble fini de couples de la forme* $(x_1/A_1, x_2/A_2, \ldots, x_n/A_n)$ *où chaque* A_i *est soit un terme, soit une fbf et chaque* x_i *une variable telle que* $x_i \neq A_i$ *et* $A_i \neq A_j$ *si* $x_i \neq x_j$.

La substitution vide sera notée par ε.

La substitution peut être considérée comme une application de l'ensemble des variables V dans celui des termes et des fbf. Ainsi soit l'ensemble E qui contient les termes ou fbf $A_1, A_2, \ldots, A_n$. La substitution $\sigma = (x_1/A_1, x_2/A_2, \ldots, x_n/A_n)$ permet d'avoir la fbf $E\sigma$ qui est le résultat de l'application σ à E et qui est obtenu en remplaçant chaque occurrence dans E de x_i par A_i , $i = 1, \ldots, n$. $E\sigma$ est appelé une *exemplification* (*instance*) de E.

DÉFINITION 4.2.2 *Soient* $\sigma = (x_1/A_1, x_2/A_2, \ldots, x_n/A_n)$ *et* $\tau = (y_1/B_1, y_2/B_2, \ldots, y_m/B_m)$ *deux substitutions. La composition* $\sigma\tau$ *de* σ *et* τ *est obtenue à partir de l'ensemble* $\{x_1/A_1\tau, x_2/A_2\tau, \ldots, x_n/A_n\tau, y_1/B_1, y_2/B_2, \ldots, y_m/B_m\}$ *en supprimant*

- *tous les* $x_i/A_i\tau$ *pour lesquels on a* $x_i = A_i\tau$, $i = 1, \ldots, n$
- *tous les* y_j/B_j *pour lesquels on a* $y_j \in \{x_1, x_2, \ldots, x_n\}$, $j = 1, \ldots, m$.

EXEMPLE 4.2.1 $\sigma = (x/Z, y/W)$, $\tau = (x/a, Z/b, W/y) \Rightarrow \sigma\tau = (x/b, Z/b, W/y)$

La composition des substitutions n'est pas en général commutative.

ASCÈSE 4.9 *Vérifier la non commutativité de la substitution en utilisant l'exemple ci–dessus.*

Les propriétés des substitution sont les suivantes :

PROPRIÉTÉ 4.2.1 $E(\sigma\tau) = (E\sigma)\tau$

PROPRIÉTÉ 4.2.2 $(\rho\sigma)\tau = \rho(\sigma\tau)$

PROPRIÉTÉ 4.2.3 $\varepsilon\theta = \theta\varepsilon = \theta$

PROPRIÉTÉ 4.2.4 *Si* $\models E$, *alors* $\models E\sigma$

4.3 Interprétation sémantique - Modèles

La première tâche du calcul des prédicats est de donner un sens aux fbf, c'est-à-dire d'établir la valeur de vérité de ces formules. Néanmoins il n'est pas possible à chaque fbf d'associer une valeur de vérité, car, pour une formule atomique, il faut en même temps donner une valeur aux arguments du prédicat et au prédicat lui-même. Or la valeur du prédicat dépend des valeurs de ses arguments. Pour cette raison nous avons besoin d'une fonction d'interprétation qui donnera la valeur de vérité au prédicat en fonction des valeurs de ses arguments.

Nous donnons d'abord une présentation informelle de l'interprétation.

On considère un langage $\mathcal{L}$ du 1er ordre et un univers du discours $\mathcal{U}$. Nous allons essayer de mettre en association les éléments du langage d'une part et les éléments de l'univers du discours, d'autre part. Cette correspondance entre éléments du langage et éléments de l'univers du discours constitue ce qu'on appelle une *interprétation* I du langage ou, encore, *structure*. Nous pouvons envisager que cette interprétation I ne couvre pas la totalité de l'univers du discours $\mathcal{U}$ mais une partie seulement, notée $\mathbb{D}(I) \subset \mathcal{U}$ et qui est le *domaine* de l'interprétation.

La correspondance I que nous venons de présenter n'est pas une application au sens habituel du terme mais plutôt un ensemble d'applications, chacune de ses applications étant spécifique à un type d'éléments de $\mathcal{L}$.

Ainsi pour les constantes a de $\mathcal{L}$ il faut trouver des constantes $a_{I.}$ dans $\mathcal{U}$ pour les faire appliquer.

Pour un foncteur f de $\mathcal{L}$ il faut trouver une relation f_I dans $\mathcal{U}$ qui peut être utilisée comme une fonction à valeurs dans $\mathcal{U}$. Dans ce cas, si l'arité de f est n, alors l'interprétation de $f(t_1,\ldots,t_n)$, où t_i ce sont des termes, est $f_{I.}((t_1)_{I.},\ldots,(t_n)_I)$ avec $(t_i)_I$ interprétation du terme t_i.

Pour un prédicat p dans $\mathcal{L}$ il faut trouver une relation p_I dans $\mathcal{U}$ qui peut être utilisée comme une fonction ayant deux valeurs : 0 ou 1. Dans ce cas, si l'arité de p est n, alors l'interprétation de $p(t_1,\ldots,t_n)$, où t_i ce sont des termes, est $p_{I.}((t_1)_{I.},\ldots,(t_n)_I)$ avec $(t_i)_I$ interprétation du terme t_i.

Nous obtenons ainsi la définition suivante :

DÉFINITION 4.3.1 (Interprétation) *Une* interprétation *I sur un langage $\mathcal{L}$ est un domaine non vide $\mathbb{D}(I)$ (qui parfois est noté $\mid I \mid$), appelé* domaine de l'interprétation, *et une application qui associe :*

- *chaque constante $a \in \Xi$ avec un élément $a_I \in \mathbb{D}(I)$;*
- *chaque foncteur $f \in \mathbf{F}$ d'arité n avec une fonction $f_I : \mathbb{D}(I)^n \to \mathbb{D}(I)$;*
- *chaque prédicat $p \in \mathbf{P}$ d'arité n avec une relation $p_I \subseteq \mathbb{D}(I)^n$.*

Remarquons que la dernière association peut se comprendre comme étant une application de $\mathbb{D}(I)^n$ dans l'ensemble $\{0,1\}$ (faux, vrai).

ASCÈSE 4.10 *Considérons un langage $\mathcal{L}$ et un prédicat unaire $p \in \mathcal{L}$. Supposons que le domaine de définition de ce prédicat est l'ensemble $D = \{1,2\}$. Établissez pour le prédicat p les quatre interprétations I_i ; $i = 1,\ldots,4$ possibles et distinctes en utilisant comme domaine d'interprétation l'ensemble D.*

Grâce à cette définition de l'interprétation, on peut attribuer des valeurs de vérité aux fbf. En effet la valeur de vérité d'une fbf sera définie en fonction des valeurs de vérité de ses composantes qui sont soit des fbf à leur tour, soit des termes. Il faut donc pouvoir établir l'interprétation des termes. Dans la mesure où les termes contiennent des variables, il faut pouvoir les associer au domaine. Nous voyons donc que l'interprétation des variables du langage $\mathcal{L}$ pose un problème. Car pour pouvoir interpréter une variable x il faudrait savoir quel objet désigne exactement x. Mais dans ce cas x ne serait plus une variable. On s'en sort de cette situation embarrassante par un artifice du type suivant : les variables du langage sont interprétées comme étant des éléments variables de l'univers du discours ! Et pour formaliser cette belle interprétation on utilise ce qu'on appelle l'*assignation* des variables par rapport à une interprétation, qui est une application $\overline{\varphi}_I : V \to \mathbb{D}(I)$ où V l'ensemble des variables du langage $\mathcal{L}$ c'est-à-dire formellement :

DÉFINITION 4.3.2 (Sémantique des variables) *On appelle assignation d'un ensemble des variables $W \subset V$ relativement à une interprétation I , une application $\overline{\varphi}_I : W \to \mathbb{D}(I)$.*

EXEMPLE 4.3.1 *Considérons un langage $\mathcal{L}$ avec une constante a, une variable x, un foncteur unaire f et un prédicat binaire p. On peut envisager la formule $p(f(a),a)$. La transformation de cette formule par l'interprétation I serait $p_I(f_I(a_I),a_I)$. Mais qu'en est-il de la transformation de la formule $p(f(x),a)$? Si on procède à l'assignation x_I de la variable x, où x_I une variable indiquant un élément quelconque de l'univers du discours, alors on peut écrire pour l'interprétation : $p_I(f_I(x_I),a_I)$. Bien sûr si on pose $x = a$, alors les deux interprétations sont identiques.*

L'assignation des variables nous permet maintenant de définir la signification d'un terme.

DÉFINITION 4.3.3 (Sémantique des termes) *La* signification [3] φ_I *d'un ensemble des termes* $\mathbb{T}$ *relativement à une interprétation* I *, est définie comme suit :*

- *si* $t \in \mathbb{T}$ *est une constante, alors* $\varphi_I(t) = t_I$ *;*
- *si* $t \in \mathbb{T}$ *est une variable, alors* $\varphi_I(t) = \overline{\varphi}_I(t)$ *;*
- *si* $t \in \mathbb{T}$ *est un terme de la forme* $f(t_1, t_2, \ldots, t_n)$*, alors* $\varphi_I(t) = f_I(\varphi_I(t_1), \varphi_I(t_2), \ldots, \varphi_I(t_n))$.

ASCÈSE 4.11 *Considérons un ensemble des termes* $\mathbb{T}$ *qui contient la constante zéro (l'élément neutre de l'addition), le foncteur unaire s (l'élément successeur) et le foncteur binaire plus (l'addition de deux valeurs). On cherche à établir une signification relativement à une interprétation* I *dont son domaine* $\mathbb{D}(I)$ *est l'ensemble des entiers non négatifs muni de l'opération de l'addition* $+$.

(1) Quelle doit-être, selon vous, l'interprétation des termes zéro , s et plus ?

(2) Calculer la signification du terme $plus(s(zero), X)$ *où* X *une variable avec assignation*

$$\overline{\varphi}_I(X) = \begin{cases} 0 & si\ X \neq \text{ nombre entier} \\ \\ |X| & si\ X = \text{ nombre entier} \end{cases}$$

On peut maintenant, grâce aux deux définitions précédentes, calculer la valeur de vérité d'une fbf. Formellement nous avons la définition suivante :

DÉFINITION 4.3.4 (Sémantique des fbf) *La* valeur de vérité de la signification d'une fbf F relativement à une interprétation I [4] *, est définie comme suit :*

- *Si la fbf est de la forme* $F = p(t_1, t_2, \ldots, t_n)$ *, alors* $I(F)$ *est égale à la valeur de vérité de* $p_I(\varphi_I(t_1), \varphi_I(t_2), \ldots, \varphi_I(t_n))$.
- *Si la fbf* F *a une des formes* $\neg G$ *,* $G \vee H$ *,* $G \wedge H$ *,* $G \rightarrow H$ *,* $G \leftrightarrow H$*, alors* $I(F)$ *est égale à la valeur de vérité de la forme correspondante.*
- *Si la fbf* F *est de la forme* $\forall x G(x, y, z, \ldots)$*, alors* $I(F) = 1$ *si* $\forall\sigma = (x/a)$ *avec* $a \in \mathbb{D}(I)$ *nous avons* $I(G\sigma) = 1$ *(vraie). Sinon* $I(F) = 0$ *(fausse).*
- *Si la fbf* F *est de la forme* $\exists x G(x, y, z, \ldots)$*, alors* $I(F) = 1$ *si* $\exists\sigma = (x/a)$ *avec* $a \in \mathbb{D}(I)$ *nous avons* $I(G\sigma) = 1$ *(vraie). Sinon* $I(F) = 0$ *(fausse).*

EXEMPLE 4.3.2 *Si* $\mathbb{D}(I)$ *est l'ensemble des nombres réels avec la relation d'égalité "=" qui correspond au prèdicat* eg *du langage* $\mathcal{L}_1$*, alors la formule atomique* $F = \text{eg}(a, b)$ *a comme valeur de vérité* $I(F)$*, la valeur de vérité de la relation* $a = b$.

---

3. La terminologie n'est pas stabilisée en français. Certains auteurs utilisent le terme *assignation* pour signification, tandis que d'autres auteurs préfèrent parler d'*interprétation*. Il est curieux de constater que personne n'utilise la traduction du terme anglais *meaning* (= signification) qui est, à mon avis, très éclairante ici.

4. ou, de façon plus succincte, la *valeur de vérité de* F *relativement à* I.

De cette définition on déduit que la valeur de vérité d'une fbf close, par rapport à une interprétation, dépend seulement de cette interprétation et elle est indépendante de l'assignation des variables. Si nous avons une fbf avec des variables libres, alors on utilise la clôture universelle qui nous permet d'obtenir une fbf close et on applique ensuite la définition précédente. L'utilisation de la clôture universelle est tout à fait concevable car, comme nous venons de le dire, l'assignation des variables n'intervient pas à la détermination de la valeur de vérité d'une fbf.

ASCÈSE 4.12 *Considérons l'ascèse 4.10. Donner, pour une interprétation, la valeur de vérité de fbf*

$$p(x) \vee \forall y (p(y) \rightarrow q)$$

Nous donnons dans la suite la définition d'une fbf satisfiable.

DÉFINITION 4.3.5 *Une fbf F est* satisfiable *ou* sémantiquement consistante *s'il existe une interprétation I telle que la valeur de vérité de F par rapport I est ègale à* 1. *L'interprétation est alors un* modèle *de F et l'on note par $I \models F$.*
Une fbf qui ne possède pas de modèle est appelée sémantiquement inconsistante *ou* insatisfiable.

ASCÈSE 4.13 *En reprenant l'ascèse 4.10, trouver un modèle pour la fbf*

$$p(x) \vee \forall y (p(y) \rightarrow q)$$

Nous arrivons ainsi à la notion de la fbf valide.

DÉFINITION 4.3.6 *Une fbf F qui est vraie pour toute interprétation est appelée* formule valide *et sera notée par $\models F$.*

ASCÈSE 4.14 *Vérifier si la fbf*

$$p(x) \vee \forall y (p(y) \rightarrow q)$$

est une formule valide.

Nous allons maintenant examiner brièvement la nature de deux quantificateurs $\forall$ et $\exists$. Considérons un prédicat p quelconque, que nous prendrons pour la circonstance et sans perte de généralité, unaire. La fbf $\forall x p(x)$ a comme valeur de vérité pour toute interprétation d'un domaine quelconque $\mathbb{D}(I)$ la valeur $\min \{ p_I / x_I \in \mathbb{D}(I) \}$, c'est-à-dire la valuation d'une fbf universellement quantifiée sur la variable x est la valuation minimale de cette formule pour toutes les interprétations x_I de x appliquées à l'interprétation p_I du prédicat p. Dans la mesure où $\min \{ p_I / x_I \in \mathbb{D}(I) \} = \bigwedge \{ p_I / x_I \in \mathbb{D}(I) \}$ on peut dire que le quantificateur universel $\forall$ est une généralisation du connecteur $\bigwedge$. De même la valuation de la fbf $\exists x p(x)$est donnée par la

valeur $\max\{p_I / x_I \in \mathbb{D}(I)\} = \bigvee\{p_I / x_I \in \mathbb{D}(I)\}$, on peut dire que le quantificateur existentiel $\exists$ est une généralisation du connecteur $\bigvee$.

Examinons maintenant la notion de la conséquence sémantique.

DÉFINITION 4.3.7 *Soit la fbf close B et un ensemble des fbf closes $A = \{A_1, A_2, \ldots, A_n\}$* [5]. *$B$ est une* conséquence sémantique *des $A_1, A_2, \ldots, A_n$, que l'on note par $A \models B$ ou $A_1, A_2, \ldots, A_n \models B$, si pour toute interprétation I telle que $I(A_i) = 1 \forall i = 1, \ldots, n$, on a $I(B) = 1$.*

En d'autres termes, on peut dire que B doit être vérifiée pour tout modèle de A.

ASCÈSE 4.15 *Considérons les deux fbf closes*

$$A = \{\forall X (\forall Y ((p(X) \wedge q(Y,X)) \rightarrow r(X,Y))), p(toto) \wedge q(koko, toto)\}$$

Montrer que la fbf close

$$r(toto, koko)$$

est une conséquence sémantique de A. (Démonstration avec l'utilisation du modus ponens, cf. infra).

En général il est difficile de vérifier si une fbf close B est une conséquence logique d'un ensemble de fbf closes A, car il faut vérifier B pour tout modèle de A. Une autre façon de démontrer que $A \models B$ est de montrer que $\neg B$ est fausse pour tout modèle de A ou, ce qui revient au même, que l'ensemble de fbf $A \cup \{\neg B\}$ est insatisfiable, c'est-à-dire n'a pas de modèle. Ce procédé est plus facile parce qu'il suffit de trouver un modèle de A qui n'est pas un modèle pour B. Formellement nous avons :

THÉORÈME 4.3.1 (de l'insatisfiabilité) *Soient A ensemble de fbf closes et B fbf close. Alors $A \models B$ si et seulement si $A \cup \{\neg B\}$ est insatisfiable.*

Une notion importante pour la sémantique des fbf est celle de l'équivalence :

DÉFINITION 4.3.8 *Deux fbf F et G sont* logiquement équivalentes *si et seulement si elles ont la même valeur de vérité pour toute interprétation I.*

EXEMPLE 4.3.3 *Les formules suivantes sont logiquement équivalentes :*

- $\neg\neg A \equiv A$
- $A \rightarrow B \equiv \neg A \vee B$
- $A \rightarrow B \equiv \neg B \rightarrow \neg A$
- $A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A)$
- *Lois de de Morgan*

5. Il faut prendre l'habitude d'interpréter un ensemble $\{A_1, A_2, \ldots, A_n\}$ de fbf comme étant une conjonction $A_1 \wedge A_2 \wedge \ldots \wedge A_n$

- $\neg(A \vee B) \equiv \neg A \wedge \neg B$
- $\neg(A \wedge B) \equiv \neg A \vee \neg B$

- $\neg \forall x A(x) \equiv \exists x \neg A(x)$
- $\forall x A(x) \equiv \neg \exists x \neg A(x)$
- $\neg \exists x A(x) \equiv \forall x \neg A(x)$
- $\exists x A(x) \equiv \neg \forall x \neg A(x)$
- $\forall x (A \vee B(x)) \equiv A \vee \forall x B(x)$ *s'il n'y a pas d'occurrences de x dans A.*

Le théorème suivant fournit des formules supplémentaires.

THÉORÈME 4.3.2 *Pour la distributivité des quantificateurs nous avons les relations suivantes :*

(1) $\models \forall x(\varphi \wedge \psi) \leftrightarrow \forall x\varphi \wedge \forall x\psi$

(2) $\models \exists x(\varphi \vee \psi) \leftrightarrow \exists x\varphi \vee \exists x\psi$

(3) $\models \forall x(\varphi(x) \vee \psi) \leftrightarrow \forall x\varphi(x) \vee \psi$ *si x n'est pas une variable libre de* ψ.

(4) $\models \exists x(\varphi(x) \wedge \psi) \leftrightarrow \exists x\varphi(x) \wedge \psi$ *si x n'est pas une variable libre de* ψ.

Attention : Les formules suivantes :
- $\forall x(\varphi(x) \vee \psi(x)) \rightarrow \forall x\varphi(x) \vee \forall x\psi(x)$
- $\exists x\varphi(x) \wedge \exists x\psi(x) \rightarrow \exists x(\varphi(x) \wedge \psi(x))$

ne sont pas vraies.

4.4 Évaluation syntaxique - Démonstration

L'évaluation syntaxique est un procédé mécanique qui permet de déduire le bien fondé d'une formule indépendamment du sens de ses composantes. Ainsi considérée, l'évaluation syntaxique est une théorie de la démonstration. La première notion de la théorie de démonstration est le théorème.

DÉFINITION 4.4.1 *Une fbf A est un* théorème*, et l'on note* $\vdash A$ *, si A est un axiome*[6] *ou si A est une formule obtenue par application des règles d'inférence sur d'autres théorèmes.*

Notons que les règles d'inférence sont celles utilisées par le mécanisme de démonstration en logique propositionnelle, plus des règles spécifiques aux quantificateurs. Pour le calcul des prédicats nous avons donc comme règles d'inférence les suivantes :

R1.- Modus ponens (règle du détachement)

$$\frac{\vdash A, \vdash A \rightarrow B}{\vdash B}$$

6. Les axiomes de la logique sont les principes fondamentaux de la logique.

R2.- Modus tollens (l'inverse de modus ponens)

$$\frac{\vdash A \rightarrow B, \vdash \neg B}{\vdash \neg A}$$

R3.- Élimination de « ET »

$$\frac{\vdash A \wedge B}{\vdash A, \vdash B}$$

R4.- Introduction de « ET »

$$\frac{\vdash A, \vdash B}{\vdash A \wedge B}$$

R5.- Généralisation

$$\frac{\vdash A}{\vdash \forall x A}$$

R6.- Exemplification universelle (instanciation universelle)

$$\frac{\vdash \forall x A}{\vdash A\sigma}, \ \sigma = (x/a)$$

R7.- Exemplification existentielle (instanciation existentielle)

$$\frac{\vdash \exists x A}{\vdash A\sigma}, \ \sigma = (x/s(x))$$

où $s(\cdot)$ est une constante qui est le résultat de l'application d'une fonction s à x.

Le calcul des prédicats possède les trois axiomes du calcul propositionnel, à savoir :

A1.- Introduction de l'implication

$$A \rightarrow (B \rightarrow A)$$

A2.- Distributivité de l'implication

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

A3.- Réalisation de contradiction

$$(\neg B \rightarrow \neg A) \rightarrow (A \rightarrow B)$$

où, encore

$$(\neg B \rightarrow \neg A) \rightarrow ((\neg B \rightarrow A) \rightarrow B)$$

plus deux axiomes supplémentaires :

A4.- Exemplification (instanciation) universelle

$$\forall x A(x) \rightarrow A(c)$$

A5.- Généralisation universelle

$$((A \to B) \to (A \to \forall x B))$$

Nous sommes maintenant en mesure de définir la démonstration et la déduction.

DÉFINITION 4.4.2 *Soit un théorème A. Une* démonstration *de A est une suite finie* $(A_1, A_2, \ldots, A_n, A)$ *où chaque* A_i *est soit un axiome, soit le résultat d'une règle d'inférence appliquée sur des éléments* A_j *précédemment obtenus.*

DÉFINITION 4.4.3 *Une fbf close A est une* déduction *de l'ensemble de fbf closes* $B_1, B_2, \ldots, B_n$, *que l'on note* $B_1, B_2, \ldots, B_n \vdash A$ *s'il existe une suite finie* $(A_1, A_2, \ldots, A_n, A)$ *où chaque* A_i *est soit un axiome, soit un des* B_i *soit il est obtenu par application d'une règle d'inférence sur des éléments* A_j *précédemment obtenus.*

Les fbf B_i sont appelées des *hypothèses*.

THÉORÈME 4.4.1 (de la déduction) *Soit A une fbf close. Si* $A \vdash B$, *alors* $\vdash A \to B$ *et vice versa.*

On introduit maintenant la notion de la théorie.

DÉFINITION 4.4.4 *Une* théorie $\mathcal{T}$ *est une collection des théorèmes avec la propriété* $\mathcal{T} \vdash p \to p \in \mathcal{T}$.

Deux théories peuvent être en relations selon la définition suivante.

DÉFINITION 4.4.5 *Soient* $\mathcal{T}$ *et* $\mathcal{T}'$ *deux théories sur les langages* $\mathcal{L}$ *rt* $\mathcal{L}'$ *respectivement.*

- $\mathcal{T}'$ *est une* extension *de* $\mathcal{T}$ *si* $\mathcal{T} \subseteq \mathcal{T}'$
- $\mathcal{T}'$ *est une* extension conservative *de* $\mathcal{T}$ *si* $\mathcal{T}' \cap \mathcal{L} = \mathcal{T}$, *i.e. tous les théorèmes de* $\mathcal{T}'$ *dans le language* $\mathcal{L}$ *sont aussi des théorèmes de* $\mathcal{T}$.

Nous terminons ce paragraphe par la notion de la consistance.

DÉFINITION 4.4.6 *Une logique est* syntaxiquement consistante *s'il n'existe aucune formule du langage telle que nous avons en même temps* $\vdash A$ *et* $\neg \vdash A$.

Nous avons le

THÉORÈME 4.4.2 *Le calcul des prédicats est syntaxiquement consistant.*

4.5 Équivalence entre modèles et théorie de démonstration

Nous allons voir que les modèles et la théorie de démonstration sont équivalentes en calcul des prédicats comme c'était déjà le cas en calcul propositionnel. Cette équivalence s'établit à l'aide de deux notions : adéquation et complétude.

DÉFINITION 4.5.1 *Une logique est* adéquate *si tout théorème* $\vdash A$ *est une formule valide* $\models A$.

Une logique est complète *si toute formule valide est un théorème, i.e.* $(\models A) \rightarrow (\vdash A)$.

L'équivalence cherchée est obtenue à l'aide des trois théorèmes suivants :

THÉORÈME 4.5.1 *Le calcul des prédicats est adéquat, i.e.*$(\vdash A) \rightarrow (\models A)$.

THÉORÈME 4.5.2 *Le calcul des prédicats est (fortement) complet, i.e.*
$(\models A) \rightarrow (\vdash A)$.

THÉORÈME 4.5.3 (de complétude) $A \vdash p \leftrightarrow A \models p$, *pour tout fbf* p *avec* $p \in \mathcal{L}$.

4.6 Quelques méta-théorèmes

Dans les paragraphes précédents nous avons introduit un certain nombre de concepts concernant les fbf prises individuellement. Nous allons étendre ces notions à un ensemble E des fbf.

THÉORÈME 4.6.1 (de la réfutation) *Une fbf* A *est* conséquence valide *d'un ensemble* E *de fbf, i.e.* $E \vdash A$, *si et seulement si* $E \cup \{\neg A\}$*est insatisfiable*[7].

Si on note par $\perp$ la fbf qui est toujours fausse, on déduit qu'un ensemble de fbf est insatisfiable si et seulement s'il a comme conséquence la formule $\perp$. Il s'ensuit donc, que toute vérification de la validité d'un ensemble de fbf peut se réduire à la preuve de son inconsistance sémantique.

Cette remarque permet d'établir une autre méthode pour la vérification d'une fbf en calcul des prédicats. Elle est fondée sur le fait que si on accepte la négation d'une fbf et on aboutit à une contradiction, alors la fbf originale est vérifiée. Notons qu'un ensemble de fbf E contient une contradiction si et seulement s'il existe une fbf A telle que nous avons à la fois $E \vdash A$ et $E \vdash \neg A$.

Nous donnons ci-après quelques métathéorèmes supplémentaires du calcul des prédicats.

THÉORÈME 4.6.2 (Règle T) *Si* $E \vdash A_1, E \vdash A_2, \ldots, E \vdash A_n$ *et*
$\{A_1, A_2, \ldots, A_{n-1}\} \vdash B$, *alors* $E \vdash B$.

THÉORÈME 4.6.3 (de contraposition) $E \cup \{A\} \vdash \neg B$ *si et seulement si* $E \cup \{B\} \vdash \neg A$.

---

7. Une autre forme équivalente (et utile) de ce théoreme est la suivante : Si $E \cup \{A\}$ est inconsistante, alors $E \vdash \neg A$.
Notons qu'il faut bien interpréter le symbole $E \cup \{A\}$. Si E est un ensemble des fbf $f_1, \ldots, f_n$, $E = \{f_1, f_2 \ldots, f_n\}$, alors $E \cup \{A\}$ signifie que nous avons f_1 ET f_2 ET ... ET f_n ET A, c'est-à-dire la virgule dans la notation de l'ensemble E joue le rôle du connecteur $\bigwedge$.

THÉORÈME 4.6.4 (de la généralisation) *Si $E \vdash B$ et x est une variable qui n'a pas d'occurrences libres dans E, alors $E \vdash \forall xB$.*

4.7 Formes clausales

Nous avons déjà introduit au chapitre précédent la notion de la clause. Quand on passe au calcul des prédicats il est possible qu'une clause contient des quantificateurs. Afin d'obtenir une normalisation des différents types des clauses nous introduisons la notion de la clause sous forme prénexe.

DÉFINITION 4.7.1 *Une* clause sous forme prénexe *est une clause de la forme $\Diamond x_1, \Diamond x_2, \ldots \Diamond x_n C$, où $\Diamond$ désigne un quantificateur et C est une fbf sans quantificateurs.*
La clause vide sera notée par $\perp$. Il s'agit d'une fbf toujours fausse.

Nous avons le théorème suivant :

THÉORÈME 4.7.1 *Toute fbf du calcul des prédicats admet une forme prénexe équivalente*

La démonstration de ce théorème peut se faire, de manière constructive, par l'algorithme suivant de la transformation d'une fbf en une forme prénexe équivalente :

(1) Élimination du connecteur $\leftrightarrow$: $(A \leftrightarrow B) \equiv (A \rightarrow B) \wedge (B \rightarrow A)$

(2) Élimination du connecteur $\rightarrow$: $(A \rightarrow B) \equiv (\neg A \vee B)$

(3) Application des substitutions d'une variable par une autre de sorte que chaque variable apparaî t sous la portée d'un seul quantificateur.

(4) Suppression des quantificateurs non opérationnels, i.e. dont la variable quantifiée n'apparaî t pas sous leur portée.

(5) Transfert de la négation au niveau le plus intérieur (i.e. devant les atomes) par utilisation des règles :
   - des lois de de Morgan $\neg(A \wedge B) \equiv \neg A \vee \neg B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$ en tant que règles de reécriture.
   - de la loi de l'involution $\neg\neg A \equiv A$ qui permet la suppression des doubles négations.
   - $\neg\forall xA \equiv \exists x\neg A$
   - $\neg\exists xA \equiv \forall x\neg A$

(6) Transfert des quantificateurs au début de la fbf en utilisant les règles :
   - $(\forall xA \wedge B) \equiv \forall x(A \wedge B)$ si B ne contient pas x.
   - $(\exists xA \wedge B) \equiv \exists x(A \wedge B)$ si B ne contient pas x.
   - $(\forall xA \vee B) \equiv \forall x(A \vee B)$ si B ne contient pas x.
   - $(\exists xA \vee B) \equiv \exists x(A \vee B)$ si B ne contient pas x.

ASCÈSE 4.16 *Calculer la forme prénexe équivalente de la fbf suivante :*

$$\forall x A(x) \wedge \exists y B(y) \rightarrow \exists y (A(y) \wedge B(y))$$

Dans une forme prénexe la présence des quantificateurs universels n'est pas significative. En effet on considère que "$\forall x\, A(x)$ *est vraie*" et "$A(x)$ *est vraie*" sont deux formules par convention équivalentes (En fait la seconde est une exemplification – instance – de la première). Par contre la présence d'un quantificateur existentiel pour une variable d'une fbf, pose le problème de la « construction » d'une telle variable.

Par exemple si nous avons $\exists x\, A(x)$, on doit pouvoir construire une valeur c pour la variable x qui permet que la fbf $A(c)$ soit vraie. Dans ce cas on dit que nous avons remplacé la variable x par une fonction s d'arité 0, c'est-à-dire par une constante. Une autre possibilité est d'avoir la variable d'un quantificateur existentielle sous la portée d'un (ou plusieurs) quantificateur(s) universel(s) relatif(s) à d'autre(s) variable(s). Ainsi considérons la fbf $\forall x \exists y\, A(y,x)$. Dans cette formule pour chaque x on postule l'existence d'un y tel que $A(y,x)$ soit vérifiée. Pour résoudre ce problème on doit avoir une fonction $s(\cdot)$ d'arité 1, qui permet pour chaque x donné, de fabriquer un y. La fbf devient ainsi $\forall x\, A(s(x),x)$.

Bien évidemment on ne cherche pas à construire cette fonction s ni, a fortiori, à lui donner une interprétation. On se contente de lui donner un nom. Une telle fonction s'appelle fonction de Skolem. Nous avons :

DÉFINITION 4.7.2 *Une* fonction de Skolem *est une fonction qui, dans une fbf close, remplace la variable d'un quantificateur existentiel et prend comme arguments les variables des quantificateurs universels sous la portée desquels était la variable du quantificateur existentiel.*
Si la variable qui est replacée par une fonction de Skolem n'était pas sous la portée d'un quantificateur universel, alors l'arité de la fonction de Skolem serait nulle, c'est-à-dire la fonction de Skolem serait une constante.

La formule qu'on obtient si on remplace une variable sous un quantificateur existentiel par une fonction de Skolem n'est pas logiquement équivalente avec la formule initiale. Seules les consistances de deux formules sont identiques, ce qui est suffisant quand, en utilisant le théorème de réfutation, on procède par preuve de l'inconsistance.

Ainsi la forme de Skolem d'une fbf close n'a que des quantificateurs universels. Ces quantificateurs peuvent être supprimés. On obtient ainsi une *exemplification* (ou une *instance*) de la forme de Skolem.

DÉFINITION 4.7.3 *Une conjonction de clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ du calcul des prédicats dans lesquels*

- *les variables sous des quantificateurs existentiels sont remplacées par des fonctions de Skolem, et*
- *les variables sous des quantificateurs universels sont omises*

est une forme conjonctive normale (fcn) *ou* forme standard.
Une fcn est parfois notée sous forme ensembliste $\{C_1, C_2, \ldots, C_n\}$.

Soit une fbf close F. L'algorithme pour aboutir à une forme standard est le suivant :

(1) Élimination du connecteur $\leftrightarrow$: $(A \leftrightarrow B) \equiv (A \rightarrow B) \wedge (B \rightarrow A)$

(2) Élimination du connecteur $\rightarrow$: $(A \rightarrow B) \equiv (\neg A \vee B)$

(3) Transfert de la négation au niveau le plus intérieur (i.e. devant les atomes) par utilisation des règles :
- des lois de de Morgan $\neg(A \wedge B) \equiv \neg A \vee \neg B$, $\neg(A \vee B) \equiv \neg A \wedge \neg B$ en tant que règles de reécriture.
- de la loi de l'involution $\neg\neg A \equiv A$ qui permet la suppression des doubles négations.
- $\neg\forall x A \equiv \exists x \neg A$
- $\neg\exists x A \equiv \forall x \neg A$

(4) Application des substitutions d'une variable par une autre de sorte que chaque variable apparaî t sous la portée d'un seul quantificateur.

(5) Suppression des quantificateurs non opérationnels, i.e. dont la variable quantifiée n'apparaî t pas sous leur portée.

(6) Suppression des quantificateurs existentiels par application de la fonction de Skolem.

(7) Conversion en forme prénexe, i.e. transfert des quantificateurs au début de la fbf en utilisant les règles :
- $(\forall x A \wedge B) \equiv \forall x (A \wedge B)$ si B ne contient pas x.
- $(\exists x A \wedge B) \equiv \exists x (A \wedge B)$ si B ne contient pas x.
- $(\forall x A \vee B) \equiv \forall x (A \vee B)$ si B ne contient pas x.
- $(\exists x A \vee B) \equiv \exists x (A \vee B)$ si B ne contient pas x.

(8) Suppression des quantificateurs universels.

(9) Conversion en forme standard par application de la distributivité
- de $\vee$ par rapport à $\wedge$: $A \vee (B \wedge C) = (A \vee B) \wedge (B \vee C)$
- de $\wedge$ par rapport à $\vee$: $A \wedge (B \vee C) = (A \wedge B) \vee (B \wedge C)$

en tant que règles de réécriture.

ASCÈSE 4.17 *Appliquer l'algorithme ci-dessus à la fbf*

$$\exists x (p(X) \wedge \forall y (p(y) \rightarrow \neg q(Y, X))) \wedge \neg\exists x (p(X) \wedge \forall y (p(y) \rightarrow \neg q(Y, X)))$$

De façon analogue nous pouvons définir la *forme disjonctive normale (fdn)* : il s'agit d'une disjonction de conjonctions des littéraux.

Un cas particulier de fdn est le problème SAT de satisfaction d'une fdn. Dans la littérature on trouve les problèmes k-SAT qui concernent la satisfiabilité des fdn dont chaque terme a au plus k littéraux. Il y a beaucoup de résultats pour $k = 2$, qui sont des problèmes de classe de complexité P et pour $k = 3$ qui sont des problèmes de classe de copmlexité NP-complet [8]. Une étude plus complète de ces problèmes se fera dans le cours de Décidabilité.

En utilisant les règles de de Morgan on constate que la négation d'une fcn est une fdn et vice-versa. Par conséquent ce qui vient d'être dit pour les problèmes k-SAT peut être transposé dans le cas des fcn. Ce qui nous intéresse ici est de savoir si un programme logique donné E permet d'induire une fbf A, c'est-à-dire si $E \vdash A$. Pour résoudre ce problème on transforme d'abord E et A sous forme conjonctive normale. Ensuite nous avons deux possibilités :

(1) Soit on construit le programme $E \cup \{A\}$ et on cherche à savoir s'il est valide, c'est-à-dire si toutes les interprétations de ce programme sont des modèles pour ce programme. Il s'agit, comme nous venons de voir, d'un problème NP-complet.

(2) Soit on construit le programme $E \cup \{\neg A\}$ et on cherche à savoir si $E \cup \{\neg A\} \vdash \bot$ ce qui est plus facile que le précédent car il suffit de trouver une interprétation qui rend fausse la fcn $E \cup \{\neg A\}$.
Il est à noter que Prolog fonctionne selon ce principe.

Il est utile de préciser ici que le calcul de la réponse d'un programme à une question est différent de la validité de ce même programme. Ce dernier problème est NP-complet et il n'y a pas actuellement une méthode de test de programme qui soit pleinement satisfaisante.

4.8 Exercices

EXERCICE 4.1 *Considérons l'univers du discours* $\{\bigcirc, *\}$ *avec*

- *les constantes* a, b
- *les variables* x, y, z
- *la relation binaire* f

Considérons une interprétation I *telle que*

- *l'interprétation des constantes est :* $a_I = \bigcirc, b_I = *$
- *l'assignation des variables est :* $\varphi_I(x) = \bigcirc, \varphi_I(y) = \bigcirc, \varphi_I(z) = *$

8. Un problème est de classe NP s'il peut être résolu par une machine de Turing non déterministe en temps polynômial (NP signifie non déterministe polynômial).
Un problème Π est *NP-difficile* si pour tout autre problème Π' de NP on a un algorithme de complexité inférieure à celle de la classe NP qui transforme toute instance de Π' en une instance de Π.
Un problème est NP-complet s'il est dans NP et s'il est NP-difficile.

– *l'interprétation de la relation est :* $f_I = \{\langle \bigcirc, * \rangle, \langle *, * \rangle\}$

Calculer la valeur de vérité des termes suivants :

(1) $\neg f(b,a)$

(2) $f(a,b) \wedge f(b,a)$

(3) $f(a,b) \vee f(b,a)$

(4) $f(a,b) \rightarrow f(b,a)$

(5) $f(a,b) \leftrightarrow \neg f(b,a)$

EXERCICE 4.2 *Considérons l'ascèse 2.2 et supposons que l'ensemble de termes contient en plus le prédicat p avec interprétation*

$$p_I = \{1,3,5,\ldots\}$$

Calculer la valeur de vérité de la fbf

$$p(\textit{zéro}) \wedge p(s(\textit{zéro}))$$

EXERCICE 4.3 *Traduire enf fbf du calcul des prédicats les propositions suivantes :*

(1) Les points X,Y,Z *sont sur la même droite.*

(2) Les points X,Y,Z *ne sont pas sur la même droite.*

(3) les lignes l *et* l' *ont un point commun unique.*

(4) Deux droites distinctes ont un point commun unique.

(5) l, l' *sont deux droites parallèles.*

(6) (5e postulat d'Euclide.) Pour toute droite L *et pour tout* X *il existe une droite unique qui passe par* X *est est parallèle à* L.

EXERCICE 4.4 *Considérons deux formules closes* F *et* G. *Montrer que* $F \equiv G$ *si et seulement si* $\{F\} \models G \wedge \{G\} \models F$.

EXERCICE 4.5 *Soit* $\mathcal{L}$ *un langage du 1er ordre. Considérons la formule*

$$\forall x \exists y p(x,y) \rightarrow \exists y \forall x p(x,y)$$

Est-ce que cette formule peut être satisfaite pour n'importe quelle interprétation de $p(x,y)$ *?*

EXERCICE 4.6 *Soit* $\mathcal{L}$ *un langage du 1er ordre constitué d'un prédicat unaire* p *et d'un prédicat binaire* r.

Nous considérons les formules suivantes :

(1) $\exists x \forall y \exists z ((p(x) \rightarrow r(x,y)) \wedge p(y) \wedge \neg r(x,y))$

(2) $\exists x \exists z (r(z,x) \rightarrow r(x,z) \rightarrow \forall y r(x,y))$

(3) $\forall y (\exists z \forall x r(x,z) \wedge \forall x (r(x,y) \rightarrow \neg r(x,y)))$

(4) $\exists x \forall y ((p(y) \rightarrow r(y,x)) \wedge (\forall t (p(t) \rightarrow r(t,y)) \rightarrow r(x,y))$

(5) $\forall x \forall y ((p(x) \wedge r(x,y)) \rightarrow ((p(x) \rightarrow \neg r(x,y)) \rightarrow \exists z (\neg r(z,x) \wedge \neg r(y,z))))$

(6) $\forall z \forall u \exists x \forall y ((r(x,y) \wedge p(u) \rightarrow (p(y) \rightarrow r(z,x)))$

Vérifier si ces formules sont satisfaites dans le modèle suivant : Le domaine est $\mathbb{N}$*, l'interprétation de r est la relation d'ordre* $\leq$ *et celle de p est le sous-ensemble des naturels pairs.*

EXERCICE 4.7 *La loi dit qu'il est interdit de posséder des armes à feu non enregistrées. Butch possède plusieurs armes à feu non enregistrées, achetées toutes chez Le Kid. Montrer que Le Kid est un hors-la-loi.*

4.A APPENDICE.- Introduction à Prolog

Prolog = PROgrammation en LOGique est un langage de programmation apparu pour la première fois en 1972. Son concepteur est A. Colmerauer de l'université de Marseille. Ensuite il a été popularisé par R. Kowalski sur des ordinateurs PDP-11. Kowalski étant à l'université d'Endibourg, cette version de Prolog est connue jusqu'à nos jours comme le Prolog d'Endibourg.

Prolog est un sous-ensemble de la logique du premier ordre. Prolog considère un programme comme une théorie et pour faire des calculs utilise le modus ponens.

Il s'agit d'un langage declaratif. On présente ce qui doit être calculé, non pas comment doit être calculé.

Exemple de fichier source.

```
porteParapluie :- ilPleut.
ilPleut.

mortel(X) :- humain(X).
existeHumain :- humain(X).
humain(socrate).
humain(aristote).

animal(X) :- lion(X).
```

Exemple des questions :

```
?- ilPleut.
yes

?- porteParapluie.
yes

?- animal(socrate).
no

?- mortel(X).
X = socrate;
X = aristote;
no

?- existeHumain.
yes
```

4.A.1 Éléments du langage

Prolog fonctionne avec des clauses de Horn sous leurs trois formes différentes, à savoir des faits, des prédicats et des questions.

- ***Faits :*** Ils sont de la forme
  `humain(socrate).`
  On affirme le fait que Socrate est un être humain.
  On distingue
  - Entiers. Ex. `6, factoriel(6).`
  - Réels. Ex. `3.1415, pi(3.1415).`
  - Constantes. Ce sont des atomes. Ex. `socrate, toto, ...` . Notons que les nombres entiers ou flottants sont considérés comme des constantes.
  - Variables. Ex. `X, Arbre`. Il y a une variable particulière, la variable anonyme notée "_", c'est-à-dire elle existe mais elle ne peut pas être utiliée par le programme.
    Bien sûr à proprement parler, une variable ne peut pas être considérée comme un fait. Mais en Prolog nous pouvons envisager d'avoir des données incomplètes. Par exemple une base de données dans laquelle pour un article donné tous les champs ne sont pas remplis. On peut palier à ce manque des données par une variable.
    N.B. Prolog n'est pas un langage typé. On doit seulement distinguer entre les constantes et les variables. Les noms des constantes commencent par une lettre minuscule. Ex. `socrate`. Les noms des variables commencent par une lettre majuscule. Ex. `Arbre`.
  - Données structurées composées des foncteurs et des arguments. Par exemple `date(10, novembre, 2007).`
  - Prédicats. Ils ont un nom, par exemple `humain(X)` et une arité positive. La notation utilisée est `nomPredicat/arité`. Ex. `humain/1`.
    En particulier on peut considérer des prédicats qui contiennent comme arguments des foncteurs.
    Exemple :
    - Tester si la valeur de sinus d'un angle est postive : `plusGrand(sin(3.1415/6), 0).`
    - Les dates anniversaires : `anniversaire(toto, date(10, novembre, 2007)).`
      Ici `date(10, novembre, 2007)` est un foncteur.
    - Description des branches gauche et droite d'un arbre binaire.
      `brancheGauche(arbre(L,R), L).`
      `brancheDroite(arbre(L,R),R).`
      Ici `brancheGauche` et `brancheDroite` sont des prédicats, `arbre` est un foncteur.
- ***Règles :*** Il s'agit des règles strictes de Horn composées par des prédicats.
  `porteParapluie :- ilPleut.`

On lira ce morceau de programme “je porte un parapluie SI il pleut”. Ici la notation “ `:-`” se lit “SI”.
L'écriture d'un programme en Prolog est en fait l'écriture des règles.
On comprend ainsi qu'un programme en Prolog est déclaratif et non pas procédural : les règles présentent (déclarent) le comportement du programme, ce qu'il doit faire le programme et on ne se préoccupe pas de la manière (procédure) avec laquelle ce qu'il doit être fait, se fera.

– ***Questions :*** Elles servent à savoir si un fait est vérifié, c'est-à-dire s'il est une implication logique du programme, considéré comme un ensemble des formules logiques. Ex. `?-mortel(socrate).`.
On peut aussi avoir des questions existentielles : On cherche à savoir s'il existe un élément qui satisfait à la propriété exprimée par le prédicat de la question. Ex. `?-mortel(X).`.

4.A.2 Fonctionnement (simplifié) de Prolog

Soit le programme E :

```
p.
q.
```

Le programme s'écrit sous forme ensembliste `E = {p, q}`. Soit maintenant la question.

Q : `?- p ∧ q`

Pour répondre à la question,Prolog applique la résolution par réfutation. Donc dans E est placée aussi la négation de la question :

`E = {p, q,` $\neg(p \wedge q)\} = \{$`p, q,` $\neg p \vee \neg q\}$

Comme E contient maintenant la clause absurde $p \wedge \neg p$, on conclut que $E \vdash Q$.

4.A.3 Repésentation des nombres naturels

Considérons l'univers de discours $\mathcal{U}$ composé de l'ensemble de nombres naturels et de l'opération de l'addition, munie de son élément neutre 0. Si on veut construire un langage $\mathcal{L}_0$ dont on chercherait une interprétation dans $\mathcal{U}$, on doit avoir un foncteur équivalent à l'opération de l'addition, ainsi que l'élément neutre. Plaçons-nous dans le cadre d'un Prolog pur, c'est-à-dire d'un Prolog qui ne contient pas des formes arithmétiques. Convenons d'appeler `add/3` l'opération d'addition dans $\mathcal{U}$[9]. Notons aussi par `zero` l'élément neutre de `add` qui est, par ailleurs, une des constantes du langage $\mathcal{L}_0$. Il nous faut aussi un prédicat, que nous appellerons `nat/1` pour caractériser les nombres naturels. On pourra ainsi écrire :

```
nat(zero).
```

9. `add/3` signifie que le foncteur `add` est d'arité 3, c'est-à-dire que le nombre de ses arguments est 3.

pour indiquer que `zero` est un nombre naturel. La question qui se pose maintenant concerne les autres nombres naturels, à savoir de quelle manière nous allons représenter ces nombres. Depuis Peano, au moins, on sait que nous pouvons construire l'ensemble des nombres naturels à partir de 0 et de l'opérateur de succession $s(X) = X + 1$, où X est un nombre naturel. On peut utiliser cette même technique pour le langage $\mathcal{L}_0$. On se dote donc d'un foncteur `s/1` qui représente l'opérateur de succession dans $\mathcal{L}_0$ et, par conséquent, le programme complet de caractérisation des nombres naturels s'écrit :

```
nat(zero).
nat(s(X)) :- nat(X).
```

On peut, par exemple, poser la question

```
?- nat(s(s(s(s(zero))))).
```

et on aura comme réponse `yes`. Mais le plus rigolo c'est quand on pose la question

```
?- nat(X).
```

Dans ce cas on recupère comme réponse

```
X = zero ;
X = s(zero) ;
X = s(s(zero)) ;
X = s(s(s(zero))) ;
X = s(s(s(s(zero)))) ;
X = s(s(s(s(s(zero))))) ;
. . . . . .
```

c'est-à-dire l'ensemble des nombres naturels.

Avant d'avancer en programmation, essayons de voir, sous l'aspect logique des prédicats, ce que nous venons de construire. Nous avons une logique du 1er ordre $\mathcal{L}_0$ dont les termes sont $\mathbb{T} = \{\texttt{zero}, \texttt{s/1}, \texttt{add/3}, \texttt{X}, \texttt{Y}\}$ et nous avons aussi le prédicat `nat/1`. L'interprétation I que nous avons adoptée conduit à la signification ϕ_I suivante des termes : (cf. Définition 4.3.3, p.52 du poly de logique) :

- `zero` $\rightarrow \phi_I(\texttt{zero}) = \texttt{zero}_I = 0$
- s(X) $\rightarrow \phi_I\left(\texttt{s}(\overline{\phi}_I(\texttt{X}))\right) = \texttt{s}_I(\overline{\phi}_I(\texttt{X})) = \overline{\phi}_I(\texttt{X}) + 1$
- add(X,Y,Z) $\rightarrow \phi_I\left(\texttt{add}(\overline{\phi}_I(\texttt{X}), \overline{\phi}_I(\texttt{Y}), \overline{\phi}_I(\texttt{Z}))\right) = \texttt{add}_I\left(\texttt{add}(\overline{\phi}_I(\texttt{X}), \overline{\phi}_I(\texttt{Y}), \overline{\phi}_I(\texttt{Z})\right)$ d'où $\overline{\phi}_I(\texttt{X}) + \overline{\phi}_I(\texttt{Y}) = \overline{\phi}_I(\texttt{Z})$.

Pour écrire le programme de l'addition de deux naturels en Prolog nous allons utiliser les deux remarques suivantes :

(1) Si on ajoute 0 à une valeur X, le résultat est X. En Prolog on a

```
add(zero,X,X).
```

(2) Si on a X+Y=Z, alors (X+1) + Y = (Z+1). En Prolog on a

```
add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

On peut, par exemple, faire l'ddition

```
?- add(s(s(s(s(s(zero))))), s(s(zero)), Z).
```

et avoir comme réponse

```
Z = s(s(s(s(s(s(s(zero)))))))
```

Mais on peut aussi faire la soustraction entre le troisième terme et le premier

```
?- add(s(s(s(s(s(zero))))), Y, s(s(s(s(s(s(s(zero))))))).
```

avec réponse

```
Y = s(s(zero))
```

et aussi entre le troisième terme et le deuxième

```
?- add(X, s(s(zero)), s(s(s(s(s(s(s(zero))))))).
```

avec réponse

```
X = s(s(s(s(s(zero)))))
```

4.A.4 Exercices

1. Écrire le programme Prolog `mult(X,Y,Z)` qui calcule le produit de deux naturels `X`, `Y` et stocke le résultat dans `Z`.

Exemple : `mult(s(s(zero)), s(s(s(zero))), Z)` donnera comme résultat `Z = s(s(s(s(s(zero)))))`.

2. En utilisant le programme précédent, calculer le résultat de la division de deux naturels `X`, `Y`. On fait l'hypothèse que `X` divise `Y`, c'est-à-dire que Y/X est un entier.

3. Écrire le programme Prolog `egal(X,Y)` qui teste si les variables `X`, `Y` ont la même valeur.

4. Écrire le programme Prolog `pair(X)` qui teste si la valeur de `X` est paire. On fera l'hypothèse que `zero` est pair.

5. Même chose pour le prédicat `impair(X)`.

6. Écrire le programme Prolog `ppe(X,Y)` qui teste si `X` est plus petit ou égal à `Y`.

5

MODÈLES DE HERBRAND. UNIFICATION

Le calcul des prédicats est utilisé dans des problèmes concernant la satisfaction des contraintes ou les bases de données déductives ou, encore, la vérification formelle du logiciel. Ce calcul a toujours la même forme : étant donné un ensemble E des fbf, vérifier si, pour une fbf A, on a $E \models A$. Nous avons déjà rencontré et résolu ce problème dans le cas du calcul des propositions. En effet dans ce cas précis le problème de la satisfiabilité des fbf propositionnelles est décidable. Mais dans le cas du calcul des prédicats nous pouvons montrer que le même problème est indécidable [1] !

Afin de surmonter ces difficultés nous avons déjà établi quelques simplifications syntaxiques : passage d'une fbf en fcn et transformation de cette dernière en clause de Horn. De plus le théorème de la réfutation du chapitre précédent nous montre que si on veut démontrer que l'ensemble des clauses de Horn E fournit la preuve de la clause A, il suffit de démontrer que l'ensemble des clauses $E \cup \{\neg A\}$ est insatisfiable. Autrement dit il suffit de montrer qu'il n'y a pas d'interprétation

1. c'est-à-dire étant donné un ensemble E des fbf on ne peut pas, dans tous les cas, conclure qu'il soit satisfiable ni qu'il soit insatisfiable

qui est un modèle pour $E \cup \{\neg A\}$. Il est évident que sous cette forme la tâche de démonstration est quasiment impossible, sauf à réduire la taille de l'espace de recherche.

L'objectif de ce chapitre est de présenter la méthode pour opérer cette réduction qui est composée de deux étapes :

- Établir un lien entre la logique des prédicats et celle des propositions. Ce lien est obtenu grâce à l'univers et la base de Herbrand.
- En utilisant l'univers et la base de Herbrand, construire une énumération récursive des modèles de E.

5.1 La preuve dans la logique des prédicats

Considérons l'ensemble de fbf $E = \{B_1, \ldots, B_n\}$.

Le problème de la déduction est de montrer que la fbf A est une conséquence logique de E.

En reformulant ce problème en termes de validité on a le problème suivant :

Est-ce que la formule

$$(\omega) \quad B_1 \wedge \ldots \wedge B_n \rightarrow A$$

est une formule valide ?

On peut aussi reformuler le même problème en termes de satisfiabilité.

Est-ce que l'ensemble

$$E \cup \{\neg A\}$$

est un ensemble insatisfiable ?

Par conséquent, la preuve,du point de vue sémantique, dans la logique des prédicats consiste à établir que chaque interprétation qui est un modèle pour l'ensemble E des fbf closes, l'est aussi pour la fbf close A. En général il y a une infinité d'interprétations et donc l'approche sémantique ne peut pas s'appliquer. Sauf si on arrive à réduire de manière drastique le nombre d'interprétations à prendre en considération lors d'une procédure de déduction.

Une méthode pour effectuer cette reduction a été introduite par Löwenheim en 1915. Elle était fondée sur la possibilité de substituer dans une formule logique les variables par des constantes, de sorte que la formule ne contenait plus de variables. Herbrand en 1930 a élaboré une extension de cette méthode, pour faire la démonstration des théorèmes du point de vue syntaxique. Dans ce chapitre nous présentons d'abord la méthode de Herbrand. On continue avec la présentation de l'unification et on termine avec la discussion du modèle minimal de Herbrand.

5.2 Interprétations de Herbrand

On rappelle une définition que nous avons vu en calcul des prédicats et on en profite pour faire une extension.

DÉFINITION 5.2.1 *Une fbf sans variables est appelée* fbf filtrée.
Un terme sans variables est appelé terme filtré *(ou* terme de base *ou* terme clos*).*
Une formule atomique ou atome sans variables est appelé atome filtré *(ou* atome de base *ou* atome clos*).*

Comme on cherche à réduire le nombre d'interprétations possibles pour un ensemble donné des fbf, il est évident qu'il faut modifier notre fade voir l'univers du discours. Pour ce faire, nous allons d'abord construire un univers du discours à partir d'un langage de la logique du 1er ordre.

Soit $\mathcal{L} = (A, \mathbb{T}, \mathbb{F})$ un langage du premier ordre. Considérons un ensemble des fbf E qui a comme langage sous-jacent $\mathcal{L}$, c'est-à-dire que ses prédicats, foncteurs, variables et constantes font partie de $\mathcal{L}$. Dans la suite on supposera toujours que E est un ensemble des clauses sans éléments contradictoires et contenant seulement de la connaissance positive, c'est-à-dire contenant seulement des règles et des faits et ne contenant pas des questions. Un tel programme s'appellera *programme défini*. L'importance des programmes définis vient du fait qu'ils ont au moins un modèle. Pour établir ce modèle nous allons introduire l'univers et la base de Herbrand.

DÉFINITION 5.2.2 (Univers de Herbrand) *Soit un ensemble de clauses E qui a comme langage sous-jacent $\mathcal{L}$. L'*univers de Herbrand *$\mathcal{U}_E$ de l'ensemble E est l'ensemble de tous les termes filtrés que l'on peut former en utilisant les constantes et les foncteurs de E.*
Si E ne contient pas un symbole de constante, alors on y ajoute arbitrairement un symbole de constante a_H (la constante de Herbrand), afin de pouvoir former des termes filtrés.

Ainsi défini, l'univers de Herbrand est composé des tous les termes filtrés que nous pouvons construire à partir de E.

DÉFINITION 5.2.3 (Base de Herbrand) *Soit un ensemble de clauses E qui a comme langage sous-jacent $\mathcal{L}$. La* base de Herbrand *$\mathcal{B}_E$ de l'ensemble E est l'ensemble de tous les atomes filtrés que l'on peut former en utilisant les symboles de prédicats de E, appliqués aux termes filtrés de $\mathcal{U}_E$, en prenant ces derniers comme des arguments.*

On voit donc que la base de Herbrand est constitué de tous les prédicats filtrés que nous pouvons construire en utilisant les éléments de E.

De faprécise l'univers de Herbrand $\mathcal{U}_E$ se construit comme suit :

- À chaque constante a de E correspond la même constante dans $\mathcal{U}_E$.
- S'il n'existe aucune constante dans E, alors on introduit dans $\mathcal{U}_E$ la constante de Herbrand a_H.

– À chaque foncteur f d'arité n, correspond dans $\mathcal{U}_E$ le même foncteur avec comme arguments des termes clos de $\mathcal{U}_E$, e.g. $f(t_1,\ldots,t_n)$, $t_i \in \mathcal{U}_E$.

Pour la construction de la base de Herbrand $\mathcal{B}_E$, nous avons

– À chaque prédicat p d'arité n correspond dans $\mathcal{B}_E$ le même prédicat avec comme arguments des termes clos de $\mathcal{U}_E$, e.g. $p(t_1,\ldots,t_n)$, $t_i \in \mathcal{U}_E$ [2].

Nous voyons ainsi que $\mathcal{U}_E$ et $\mathcal{B}_E$ sont définis pour un programme E et contiennent exactement les symboles qui sont dans le programme.

ASCÈSE 5.1 *Soit le programme*

$$E = \{plus(zero,N,N),\ plus(s(K),L,s(M)) \leftarrow plus(K,L,M)\}$$

Déterminer l'univers et la base de Herbrand pour E.

ASCÈSE 5.2 *Soit le programme*

$$E = \{\ impair\ (X) \rightarrow\ impair\ (s(s(X)))\}$$

où $s(X)$ est le successeur de X.
Déterminer l'univers et la base de Herbrand pour E.
Quelle peut être la constante a_H ?

ASCÈSE 5.3 *Soit le programme*

$$\begin{aligned} E = \{\ & chante\ (chanteur\ (chanson)\ , chanson), \\ & chante\ (X\ , chanson)\ \rightarrow joyeux\ (X) \end{aligned}$$

Déterminer l'univers et la base de Herbrand pour E.

En suivant cette construction, on peut comprendre que l'univers de Herbrand est plus « pauvre » qu'un univers de discours tel que nous l'avions examiner au chapitre précédent, car il n'a pas des variables. Il est donc plus facile de calculer la valeur de vérité d'un ensemble de clauses E et de savoir ainsi s'il est satisfiable ou pas, car, en absence des variables, d'une part il n'est pas nécessaire de procéder à une assignation de variables et, d'autre part, la signification des termes se simplifie. Néanmoins il faut savoir que l'univers de Herbrand peut être infini comme le stipule la proposition suivante :

PROPRIÉTÉ 5.2.1 *Si dans éléments d'un programme il y a un foncteur, alors l'univers de Herbrand correspondant est de cardinal infini.*

2. Il est temps de signaler que lors de deux définitions concernant l'univers et la base de Herbrand, nous avons pris quelques libertés avec la rigueur, afin de mieux faire comprendre au lecteur l'approche de Herbrand au calcul logique. En fait l'univers de Herbrand ne se construit pas sur un ensemble E des formules mais sur le langage $\mathcal{L}$ du 1er ordre dont, d'aiilleurs, l'utilisation permet d'engendrer les formules qui sont dans E. Quand on travaille sur un ensemble de formules, on suppose que le langage $\mathcal{L}$ est défini par les constantes, les foncteurs et les prédicats qui apparaissent dans cet ensemble de formules. L'univers, donc, de Herbrand est construit sur ce langage. Cette mise au point étant faite, on pourra continuer à parler – mais seulement par abus de langage – de l'univers de Herbrand, de la base de Herbrand et de l'interprétation de Herbrand associés à l'ensemble de clauses E.

Il reste maintenant à établir la nouvelle forme de l'interprétation.

DÉFINITION 5.2.4 (Interprétation de Herbrand) *Soit un ensemble de clauses E qui a comme langage sous-jacent $\mathcal{L}$. L'* interprétation de Herbrand I_E *de l'ensemble E est constituée par l'univers de Herbrand $\mathcal{U}_E$ comme domaine de l'interprétation et une application entre E et $\mathcal{U}_E$ qui associe :*

- *à chaque constante $c \in E$, la même constante $c_{I_E} = c \in \mathcal{U}_E$;*
- *à chaque foncteur f de E d'arité n , le foncteur $f_{I_E}\left((t_1)_{I_E}, \ldots, (t_n)_{I_E}\right) = f(t_1, \ldots, t_n) \in \mathcal{U}_E$;*
- *à chaque prédicat p de E d'arité n , une relation quelconque $p_{I_E} \subseteq \mathcal{U}_E^n$ [3] (i.e. une application quelconque de $\mathcal{U}_E^n$ dans l'ensemble $\{0,1\}$ - faux, vrai).*

Ainsi pour cette interprétation l'univers du discours est l'univers de Herbrand $\mathcal{U}_E$. Pour la sémantique des termes la signification d'une constante est la constante elle-même. Pour la signification des foncteurs on utilise la signification prédéfinie des foncteurs lors de l'interprétation. En somme pour toute interprétation les termes représentent des éléments de $\mathcal{U}_E$. Nous pouvons donc envisager une interprétation pour laquelle chaque terme représente exactement un élément du domaine de l'univers du discours et chaque élément du domaine est représenté. Enfin, en ce qui concerne l'interprétation d'un prédicat, il suffit de spécifier la relation à associer avec chaque symbole de prédicat, c'est-à-dire la valeur des arguments pour lesquelles le prédicat retourne la valeur "vraie". Dans la mesure où tous ces prédicats se trouvent, sous forme filtré, dans la base de Herbrand $\mathcal{B}_E$, il suffit d'en extraire un sous-ensemble $\mathcal{B}'_E$ des prédicats qui retournent la valeur "vraie" pour l'interprétation.

En conclusion, une interprétation de Herbrand peut s'identifier à un sous-ensemble de la base de Herbrand. En effet étant donné que, pour un ensemble de clauses E, l'interprétation des constantes et des foncteurs soit fixée, une interprétation de Herbrand est la détermination du sous-ensemble de la base de Herbrand $\mathcal{B}_E$ dont les éléments ont comme valeur de vérité 1 (vrai) pour cette interprétation. Réciroquement si on considère un sous-ensemble quelconque B de la base de Herbrand $\mathcal{B}_E$ et si pour chaque prédicat de E qui est aussi dans B avec les mêmes arguments, on associe la valeur de vérité 1 (vrai), alors on obtient une interprétation de Herbrand issue de B. Nous avons donc que si I_E est une inteprétation de Herbrand, alors elle est équivalente au sous-ensemble $\{B_i \in \mathcal{B}_E \ / \models_{I_E} B_i\} \subseteq \mathcal{B}_E$. L'interprétation, donc, de Herbrand est l'interprétation la plus générale que nous pouvons donné au langage $\mathcal{L}$.

DÉFINITION 5.2.5 (Modèle de Herbrand) *Soit un ensemble de clauses E qui a comme langage sous-jacent $\mathcal{L}$. Considérons une interprétation de Herbrand I_E . Si cette interpré-*

3. qui est l'ensemble des tous les n-tuples de termes filtrés.

tation est un modèle pour chaque clause de E, alors elle est un modèle de Herbrand *pour E.*

Notons aussi que toute interprétation de Herbrand est une interprétation au sens de la définition donnée au chapitre précédent mais le contraire n'est pas toujours vrai.

ASCÈSE 5.4 *Donner quelques interprétations pour le programme de l'ascèse 5.1.*

Grâce au théorème suivant nous avons que si un ensemble de clauses E a un modèle, alors il a aussi un modèle de Herbrand.

THÉORÈME 5.2.1 *Soit I'_E un modèle pour l'ensemble de clauses E. Alors l'ensemble $I_E = \left\{ B \in \mathcal{B}_E \;\middle/\; \models_{I'_E} B \right\}$ est un modèle de Herbrand pour E.*

La conséquence immédiate de ce théorème est que nous pouvons associer des interprétations de Herbrand avec des sous-ensembles de la base de Herbrand et décrire ainsi l'interprétation par les éléments de $\mathcal{B}_E$ qui sont vrais pour cette interprétation.

ASCÈSE 5.5 *Parmi les interprétations établies à l'ascèse 5.1, quelles sont celles qui sont un modèle pour E ?*

Nous avons aussi les deux corollaires suivants :

COROLLAIRE 5.2.1 *Si un ensemble de clauses E est satisfiable, alors cet ensemble possède un modèle de Herbrand.*

D'après ce corollaire, pour prouver qu'un ensemble de clauses E est satisfiable, on doit chercher de trouver un modèle de Herbrand, c'est-à-dire il faut examiner les sous-ensembles de la base de Herbrand $\mathcal{B}_E$ de E. Nous voyons que de cette façon l'espace de recherche se réduit fortement.

COROLLAIRE 5.2.2 *Si l'ensemble de clauses E n'a pas de modèle de Herbrand, alors il est insatisfiable.*

ASCÈSE 5.6 *Soit le programme*

$$E = \left\{ \begin{array}{l} p(f(f(a))). \\ p(f(X)) \leftarrow p(X). \end{array} \right\}$$

Préciser parmi les trois interprétations suivantes, celles qui sont des modèles de Herbrand pour E :

(1) $I_1 = \{p(a)\}$

(2) $I_2 = \{p(f(f(a))), p(f(f(f(a)))), \ldots\}$

(3) $I_3 = \{p(f^n(a)) \mid n \text{ est premier}\}$

Nous pouvons donc obtenir le théorème suivant :

THÉORÈME 5.2.2 (THÉORÈME DE HERBRAND) *Un ensemble E de clauses est insatisfiable si et seulement si n'a pas de modèle de Herbrand.*

Rappelons avant de terminer ce paragraphe que les résultats obtenus ici ne sont valables que si E est un ensemble de clauses.

ASCÈSE 5.7 *Soit l'ensemble de fbf*

$$E = \{\neg p(a)\ ,\ \exists X\ p(X)\}$$

Déterminer l'univers et la base de Herbrand pour E.
Donner les interprétations de Herbrand et les modèles de Herbrand.
Donner d'autres modèles de E.

5.3 Modèle minimal de Herbrand

Nous avons vu à la fin du § 5.2 qu'un ensemble des clauses peut ne pas avoir un modèle de Herbrand. Le théorème suivant montre que tout modèle non-Herbrand d'un programme défini peut produire un modèle de Herbrand.

THÉORÈME 5.3.1 *Soit E un programme défini*[4] *et I' un modèle non-Herbrand de E. Alors l'ensemble*

$$I = \left\{A \in \mathcal{B}_E \,/\, \models_{I'} A\right\}$$

est un modèle de Herbrand.

Comme conséquence de ce théorème nous avons un fait déjà connu :

COROLLAIRE 5.3.1 *Soit E un programme défini. La base de Herbrand $\mathcal{B}_E$ de E est un modèle de Herbrand de E.*

Considérons une exemplification (instanciation) d'une clause de E en utilisant uniquement des constantes $A \leftarrow B_1, \ldots, B_n$ avec $n \geq 0$. Nous avons que $B_i \in \mathcal{B}_E$, $i = 1, \ldots, n$ et $A \in \mathcal{B}_E$. Donc la clause entière est vraie. Bien évidemment ce modèle n'est pas très intéressant car il enfonce des portes ouvertes, c'est-à-dire il affirme que tout prédicat de E peut être interprété comme une relation sur l'ensemble de termes filtrés de E. Ce qui serait ici intéressant est d'avoir des modèles qui contiennent davantage des termes filtrés que E. Avant d'examiner cette possibilité, nous allons établir la structure de la famille des modèles de Herbrand. Pour cela on rappelle que tout modèle de Herbrand est un sous ensemble de $\mathcal{B}_E$. Nous pouvons

4. Un *programme défini* est un programme qui contient uniquement des *clauses définies*, c'est-à-dire des règles ou, encore, des *clauses de Horn strictes*, et aussi des faits, c'est-à-dire ce qu'on appelle des *clauses de Horn positives*.

donc envisager, étant donnés différents modèles de Herbrand, leur intersection qui forme un autre modèle de Herbrand. On a donc le théorème suivant :

THÉORÈME 5.3.2 *Soient E un programme défini, $\mathcal{M}_E = \{I_1, I_2, \ldots\}$ une famille non vide de modèles de Herbrand de E. L'intersection $I = I_1 \cap I_2 \cap \cdots$ est aussi un modèle de Herbrand pour $\mathcal{E}$*[5].

La famille $\mathcal{M}_E$ existe pour tout E, car il y a au moins un modèle de Herbrand pour E à savoir la base $\mathcal{B}_E$ de Herbrand. Tout modèle de Herbrand est un sous-ensemble de la base de Herbrand. Si $\mathcal{M}_E$ est la famille de tous les modèles de Herbrand de E, alors l'intersection $I = \bigcap_{I_i \in \mathcal{M}_E} I_i$ représente le *modèle minimal de Herbrand* (ou encore le *plus petit modèle de Herbrand*) pour E. Ce modèle sera noté par $\widetilde{I}_E$.

THÉORÈME 5.3.3 *Le modèle minimal de Herbrand $\widetilde{I}_E$ d'un programme défini E est l'ensemble de toutes les conséquences logiques filtrées :*

$$\widetilde{I}_E = \left\{ A \in \mathcal{B}_E \,/\, E \models A \right\}$$

Ce théorème nous montre que pour un programme défini donné E, le modèle minimal de Herbrand constitue une interprétation du programme. Mais il y a plus. On avait vu précédemment que les modèles de Herbrand sont, dans un certain sens, des interprétation les plus générales, c'est-à-dire des modèles les plus généraux. Donc le modèle minimal de Herbrand contient les clauses qui sont vraies pour chaque modèle de E, c'est-à-dire les clauses filtrés qui sont des conséquences logiques de E. On pourrait donc aller plus loin et suggérer qu'un programmeur aura tout intérêt à ce que l'interprétation qui fait de son programme soit exactement le modèle minimal de Herbrand.

Toutefois il faut se garder d'extrapoler ces résultats pour n'importe quelle fbf.

ASCÈSE 5.8 *Soit la fbf $p(a) \vee q(a)$. Trouver ses modèles de Herbrand. Vérifier qu'il n'y a pas de modèle minimal de Herbrand.*

L'existence d'un modèle minimal permet de structurer la famille de toutes les interprétations de E. Cette famille est en fait l'ensemble $\mathcal{P}(\mathcal{B}_E)$ des parties de $\mathcal{B}_E$. On peut démontrer que cet ensemble, muni de l'ordre partiel $\subseteq$, est un treillis complet[6]. Pour ce treillis nous avons $\bot = \emptyset$ et $\top = \mathcal{B}_E$. Pour tout ensemble d'interprétations de Herbrand la borne inférieure est l'interprétation de Herbrand qui est

5. Il faut se garder de toute interprétation abusive de ce théorème. En effet il est inexact d'affirmer que toute intersection de modèles pour un programme défini est un modèle.

6. Un *treillis* ou *ensemble réticulé* est un ensemble muni d'un ordre partiel et disposant d'un plus petit et d'un plus grand élément, notés $\bot$ et $\top$ respectivement. Le treillis est *complet* si pour chacun de ses sous-ensembles X il y a une borne inférieure et une borne supérieure.

l'intersection de toutes les interprétations de Herbrand de l'ensemble et la borne supérieure est la réunion de ces mêmes interprétations.

Pour une clause A de E on peut déterminer des substitutions particulières σ_f, appelées *substitutions filtrantes*, telles que $A\circ\sigma_f$ soient des clauses filtrées, c'est-à-dire sans occurrence des variables. En réalité $A\circ\sigma_f$ est une exemplification (instaciation) du fait A.

Considérons un élément du treillis, c-à-d. un modèle de Herbrand $I \in \mathcal{P}(\mathcal{B}_E)$.

Pour tout fait A de E nous avons que $A\circ\sigma_f$ appartient à I et ceci pour toute substitution filtrante σ_f. Car sinon A ne serait pas vrai dans I et, par conséquent, I ne serait pas un modèle pour E.

Soit maintenant une règle $A \leftarrow B_1,\ldots,B_n$ de E avec $n>0$ qui signifie, entre autre, que si les faits $B_1,\ldots,B_n$ sont vrais, il en de même pour A. Par conséquent si nous avons une substitution filtrante σ_f pour $B_1,\ldots,B_n$, A est vrai pour la même substitution filtrante, i.e. $A\circ\sigma_f \leftarrow B_1\circ\sigma_f,\ldots,B_n\circ\sigma_f$. Donc si I contient $B_1\circ\sigma_f,\ldots,B_n\circ\sigma_f$, il doit aussi contenir $A\circ\sigma_f$ sinon il ne serait pas un modèle. Notons que $A\circ\sigma_f \leftarrow B_1\circ\sigma_f,\ldots,B_n\circ\sigma_f$ est une exemplification (instanciation) de la règle $A \leftarrow B_1,\ldots,B_n$. Cette discussion permet de voir la manière dont nous pouvons enrichir en termes filtrés le programme E. En effet soit $I_1(E)$ l'ensemble des faits filtrés de E. Il est possible de lui adjoindre des nouveaux éléments en utilisant toutes les exemplifications (instanciations) des toutes les règles de E. On engendre ainsi l'ensemble $I_2(E)$ sur-ensemble de $I_1(E)$. Ce processus peut être répété tant qu'on peut engendrer des nouveaux éléments. Ces nouveaux éléments s'ajoutent à l'ensemble $I_k(E)$ afin de former l'ensemble $I_{k+1}(E)$ qui suit immédiatement $I_k(E)$.

Afin de formaliser ce processus on donne d'abord la définition suivante :

DÉFINITION 5.3.1 (Opérateur de la conséquence immédiate) *Soit E un programme défini. L'application $\mathcal{T}_E : \mathcal{P}(\mathcal{B}_E) \rightarrow \mathcal{P}(\mathcal{B}_E)$ définie par*

$$\mathcal{P}(\mathcal{B}_E) \ni I \mapsto \mathcal{T}_E(I) = \begin{array}{l}\{A\circ\sigma_f \,/\, A\circ\sigma_f \leftarrow B_1\circ\sigma_f,\ldots,B_n\circ\sigma_f \\ \quad où\ (A \leftarrow B_1,\ldots,B_n) \in E \\ \quad et\ \{B_1\circ\sigma_f,\ldots,B_n\circ\sigma_f\} \subseteq I\}\end{array}$$

c'est-à-dire $\mathcal{T}_E(I)$ est composé de toutes de conclusions filtrées $A\circ\sigma_f$ des règles $(A \leftarrow B_1,\ldots,B_n)$ de E et telles que les prémisses filtrées $\{B_1\circ\sigma_f,\ldots,B_n\circ\sigma_f\}$ soient dans I.

$\mathcal{T}_E$ est appelé opérateur de la conséquence immédiate *de E.*

Les propriétés de cet opérateur sont données par le

THÉORÈME 5.3.4 *Soit E un programme défini. L'opérateur de la conséquence immédiate de E a les propriétés suivantes :*

(1) $\mathcal{T}_E$ est une application monotone, i.e. si $I' \subseteq I$, alors $\mathcal{T}_E(I') \subseteq \mathcal{T}_E(I)$.

(2) T_E est une application continue, i.e. si $I \in \mathcal{P}(\mathcal{B}_E)$ tel que $\sup(I) \in I$ *, alors* $\mathcal{T}_E(\sup(I)) = \sup(\mathcal{T}_E(I))$.

(3) Soit I une interprétation de Herbrand. I est un modèle de Herbrand si et seulement si $\mathcal{T}_E(I) \subseteq I$.

ASCÈSE 5.9 *Soit le programme* $P = \{p(f(X)) \leftarrow p(X), q(a) \leftarrow p(X)\}$. *Calculer* $\mathcal{T}_E(\emptyset)$, $\mathcal{T}_E(\mathcal{B}_E)$ *et* $\mathcal{T}_E(\mathcal{T}_E(\mathcal{B}_E))$.

Pour pouvoir poursuivre la construction des ensembles $I_k(E)$ il est obligatoire d'introduire la notion des puissances ordinales de l'application $\mathcal{T}_E$.

Rappelons que les nombres ordinaux se déterminent de farécurrente comme suit :
Le premier nombre ordinal est $\emptyset$ et il est noté par 0, c'est-à-dire $0 = \emptyset$.
Ensuite nous avons :
$1 = \{\emptyset\} = \{0\}$;
$2 = \{\emptyset, \{\emptyset\}\} = \{0, 1\}$;
$3 = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} = \{0, 1, 2\}$;
$\cdots \cdots$
$\omega = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \ldots\} = \{0, 1, 2, \ldots\}$ qui est l'ensemble des nombres naturels. ω est le plus petit ordinal infini. Si nous poursuivons, nous avons la suite des ordinaux infinis :
$\omega + 1 = \omega \cup \{\omega\}$;
$\omega + 2 = (\omega + 1) + 1$;
$\cdots \cdots$

Dans la suite on notera les ordinaux infinis par des lettres grecques $\alpha, \beta, \cdots$.

Sur les ordinaux nous pouvons définir un ordre, noté « $<$ » comme suit : $\alpha < \beta$ si $\alpha \in \beta$. En général si α est un nombre ordinal, alors son successeur est l'ordinal $\alpha + 1 = \alpha \cup \{\alpha\}$. C'est le plus petit ordinal qui est plus grand que α. Un ordinal α est un *ordinal limite* si'il n'est successeur d'aucun ordinal.

Il est indispensable, pour pouvoir opérer avec les ordinaux, de pouvoir introduire [7] pour ces nombres l'équivalent du principe d'induction de Peano pour les nombres naturels. Il s'agit de l'*induction transfinie* pour les ordinaux. Du fait qu'il n'existe pas l'ensemble de tous les nombres ordinaux (tout au plus nous pouvons parler de collection de tous les nombres ordinaux), nous ne pouvons pas avoir une définition générale pour l'induction transfinie. Nous utilisons donc des définitions qui sont valables jusqu'à un certain ordinal. Il y a plusieurs formulations équivalentes de l'induction transfinie. Nous présentons la plus simple (cf. E. W. Beth, pp.372-373).

7. au grand dam des constructivistes, dont je fait modestement partie.

DÉFINITION 5.3.2 (Induction transfinie) *Considérons une propriété p. Supposons que nous avons démontré (ou admis) que pour tout ordinal α nous avons que*

$$\forall \beta \text{ ordinal } (\beta < \alpha \rightarrow p(\beta)) \rightarrow p(\alpha)$$

alors tout ordinal α a la propriété $p(\alpha)$.

Pour introduire les opérations d'addition et de multiplication sur les ordinaux il faut aussi pouvoir définir la *recursion transfinie* afin de pouvoir appliquer la notion du successeur de manière analogue à celle utilisée pour les nombres naturels.

DÉFINITION 5.3.3 (Recursion transfinie) *Supposons que nous avons une fonction f qui associe à tout ensemble X d'ordinaux, un ordinal $\alpha = f(X)$. Alors il existe une unique fonction F qui à chaque ordinal α associe un ordinal $F(\alpha)$ et qui satisfait à la condition*

$$\forall\, F(\alpha) = f(X_\alpha)$$

où $X_\alpha = \{F(\beta)/\beta < \alpha\}$.

Nous pouvons maintenant envisager d'introduire l'opérateur de la conséquence immédiate aux nombres ordinaux.

Nous avons vu que $\emptyset \in \mathcal{P}(\mathcal{B}_E)$. Nous convenons de noter
$\mathcal{T}_E \uparrow 0 = \emptyset$
Ensuite nous posons :
$\mathcal{T}_E \uparrow (\alpha+1) = \mathcal{T}_E(\mathcal{T}_E(\alpha))$ si $\alpha+1$ est l'ordinal successeur de α, i.e. si $\alpha+1 = \alpha \cup \{\alpha\}$.
$\mathcal{T}_E \uparrow \omega = \bigcup \{\mathcal{T}_E \uparrow \alpha / \alpha < \omega\}$.

Cette construction permet d'utiliser l'opérateur de la conséquence immédiate avec des nombres ordinaux α à la place des ensembles $I_\alpha(E)$.

THÉORÈME 5.3.5 (Caractérisation du modèle minimal de Herbrand) *Soient E un programme défini et $\widetilde{I}_E$ son modèle minimal de Herbrand. Alors*

(i) *$\widetilde{I}_E$ est un point fixe de l'opérateur de la conséquence immédiate : $\mathcal{T}_E\left(\widetilde{I}_E\right) = \widetilde{I}_E$.*

(ii) *$\widetilde{I}_E = \mathcal{T}_E \uparrow \omega$.*

ASCÈSE 5.10 *Donner le modèle minimal de Herbrand pour le programme de l'ascèse 5.1*

5.4 Unification

Avant de pouvoir examiner si un ensemble de clauses E est satisfiable, il faut essayer de réduire le nombre de ses éléments. On cherchera donc de trouver dans E des clauses qui bien qu'ayant des formes différentes, sont en réalité équivalents logiquement. On effectuera cette recherche en utilisant l'algorithme de l'unification.

DÉFINITION 5.4.1 *Soit $E = \{A_1, \ldots, A_n\}$ un ensemble de clauses. Une substitution σ est un* unificateur *de E ssi $A_1\sigma = \cdots = A_n\sigma$.*
Dans ce cas on dit que E est unifiable *par σ.*

On voit ainsi que si E est unifiable par σ, alors $E\sigma$ est réduit à un singleton.

ASCÈSE 5.11 *Calculer la substitution qui unifie les deux termes*

$p(X, a, f(X, a))$ *et* $p(b, U, V)$

ASCÈSE 5.12 *Calculer la substitution qui unifie les termes suivants :*

$p(X, f(Y), g(b))$, $p(X, f(c), g(Z))$, $p(X, f(c), g(U))$, $p(X, f(V), W)$

DÉFINITION 5.4.2 *Soit $E = \{A_1, \ldots, A_n\}$ un ensemble de clauses. Un unificateur θ de E est l'*unificateur le plus général (UPG) *si pour tout unificateur σ de E il existe une substitution τ telle que $\sigma = \theta \circ \tau$.*

On peut facilement constater que l'UPG est unique à une substitution de renommage près [8].

ASCÈSE 5.13 *Trouver l'UPG de termes suivants :*

$p(a, Y, Z)$ *et* $p(X, b, Z)$

ASCÈSE 5.14 *Trouver l'UPG de termes suivants :*

$p(X, f(Y))$ *et* $p(Z, f(Z))$

Est-il unique ?

L'algorithme de l'unification permet de calculer l'UPG d'un ensemble de clauses s'il existe, ou de signaler l'impossiblité de l'existence d'un tel unificateur. Avant de présenter cet algorithme, nous allons faire quelques remarques.

Les foncteurs et les prédicats sont de structures de la forme suivante : $r(s_1, \ldots, s_n)$ avec r le nom de la structure et s_i la i-ième sous-structure. Cette structure peut se représenter comme une liste $[r, s_1, \ldots, s_n] = [s_0, s_1, \ldots, s_n]$. Chaque s_i représente soit un terme, soit un prédicat. La longueur de la structure est égale au nombre de termes dans la liste. Si on note par $\mid r \mid$ cette longueur, on a $\mid r \mid = n + 1$.

Il est évident que, d'une part, deux structures ne sont unifiables que si leur longueur est identique. D'autre part si deux structures peuvent s'unifier, alors soit au moins l'une de deux est une variable, soit elles ont le même nom. Et la même chose doit être valable pour leurs sous-structures correspondantes.

8. Une substitution de renommage est une substitution du type $\sigma = (t_1/x_1, t_2/x_2, \ldots, t_n/x_n)$ où $t_1, t_2, \ldots, t_n$ sont des variables et $x_1, x_2, \ldots, x_n$ sont des variables distinctes.

DÉFINITION 5.4.3 *Soient deux structures $u_1 = [s_0, s_1, \ldots, s_n]$ et $u_2 = [r_0, r_1, \ \ldots, r_n]$. L'* ensemble non apparié *de u_1 et u_2 est l'ensemble $D(u_1, u_2)$ d'un couple de structures, défini comme suit :*

- *Si s_0 et r_0 sont différents, alors $D(u_1, u_2) = \{u_1, u_2\}$.*
- *Si s_0 et r_0 sont identiques et si les sous-structures s_i et r_i sont identiques pour $i = 1, \ldots k-1$, tandis que les sous-structures de s_k et r_k sont différentes, alors l'ensemble non apparié de u_1 et u_2 est*
  $D_k(u_1, u_2) = \{t_1 = (s_k, \ldots, s_n), t_2 = (r_k, \ldots, r_n)\}$, où t_1 et t_2 les termes respectifs de u_1 et u_2 commençant au k-ième rang.

ASCÈSE 5.15 *Calculer les ensembles non appariés de deux structures suivantes :*

$u_1 = p(f(X), h(Y), a)$ *et* $u_2 = p(f(X), Z, a)$

Si θ est le UPG de u_1 et u_2 est si $\{t_1, t_2\}$ est un de leurs ensembles non appariés, alors θ est aussi un unificateur de t_1 et t_2. Par conséquent l'UPG de u_1 et u_2 est la composition de l'UPG σ de t_1 et t_2 avec l'UPG de $u_1\sigma$ et $u_2\sigma$.

Nous présentons l'algorithme de l'unification pour un ensemble de deux structures u_1 et u_2.

(1) Si $u_1 = u_2$, alors l'UPG de u_1 et u_2 est $\sigma = \varepsilon$ (substitution identité). Fin.
Sinon, posons $\sigma = \varepsilon$.

(2) Tant que $u_1\sigma \neq u_2\sigma$ faire

DÉBUT

(a) Trouver la première sous-structure de u_1 qui soit différente de la sous-structure correspondante de u_2. Soit $D_k(u_1, u_2) = \{t_1, t_2\}$ l'ensemble non apparié à ces deux sous-structures non identiques.

i. *(Échec normal)* Si ni t_1 ni t_2 n'est une variable, alors sortie en échec.

ii. *(Vérification d'occurrence)* Si l'un de t_1, t_2 est une variable contenue dans l'autre terme, alors sortie en échec.

iii. *(Passage à l'itération suivante)* Sinon, supposons que t_1 soit une variable. Alors on pose $\sigma = \sigma \circ \tau$ où $\tau = (t_1/t_2)$ (indépendamment de la nature de t_2).

FIN

On peut appliquer ce même algorithme à un ensemble contenant m structures, de faitérative, à condition d'appliquer chaque substitution intermédiaire à la totalité des structures.

ASCÈSE 5.16 *Appliquer l'algorithme de l'unification aux deux termes suivants :*

$u_1 = p(f(X, Y), a)$ *et* $u_2 = p(f(Z, Z), Z)$

La validité de cet algorithme est donnée par le théorème suivant :

THÉORÈME 5.4.1 (Th. de l'Unification) *Soit E un ensemble fini de clauses. Si E est unifiable, alors l'algorithme de l'unification termine en donnant l'UPG pour E. Sinon l'algorithme se termine en échec.*

À cause de l'étape de la vérification d'occurrence, l'algorithme de l'unification risque d'être très coûteux en temps, car le temps de l'exécution de cette étape peut être une fonction exponentielle de la longueur de l'entrée. C'est la raison pour laquelle le langage `Prolog` utilise l'algorithme de l'unification sans l'étape de la vérification d'occurrence. Du point de vue théorique c'est une catastrophe. Du point de vue pratique on n'évite pas l'existence des cercles vicieux. Supposons, par exemple, qu'on cherche à unifier X et $f(X)$. Alors on obtient la substitution $\tau = (X/f(X))$. Bien évidemment τ n'est pas un unificateur car $X \circ \tau = f(X) \neq f(X) \circ \tau = f(f(X))$. Le deuxième appel récursif fournit la même substitution qui n'est pas, pas plus qu'avant, un unificateur, car $f(X) \circ \tau = f(f(X)) \neq f(f(X)) \circ \tau = f(f(f(X)))$. Nous arriverons ainsi à une substitution infinie $f(f(f(\cdots)))$.

5.5 Réponse correcte à un programme

Nous terminons ce chapitre par la présentation de la réponse correcte à un programme.

DÉFINITION 5.5.1 *Soient E un programme défini, B un but $\leftarrow B_1, \ldots, B_n$. Une réponse à $E \cup \{B\}$ est une substitution $\sigma_{E,B}$ pour les variables du but B.*

Il va de soi que cette substitution ne concerne pas obligatoirement toutes les variables de B. De plus si B n'a pas de variables, la seule réponse possible est la substitution identité.

DÉFINITION 5.5.2 *Soient E un programme défini et B un but $\leftarrow B_1, \ldots, B_n$ et une réponse $\sigma_{E,B}$ à $E \cup \{B\}$. On dit que $\sigma_{E,B}$ est une réponse correcte à $E \cup \{B\}$ si $B_k \circ \sigma_{E,B}$ est une conséquence logique de E pour tout $k = 1, \ldots, n$, i.e.*

$$E \models B_k \circ \sigma_{E,B} \ , \ \forall\, k = 1, \ldots, n$$

Remarquons qu'un programme au lieu de retourner une substitution comme réponse à un but, peut retourner la réponse « non ». Cette réponse est correcte si c'est le programme $E \cup \{\neg B\}$ (et non pas $E \cup \{B\}$) qui est satisfiable.

En utilisant le th. 5.3.2 on peut montrer qu'une substitution $\sigma_{E,B}$ est une réponse correcte si et seulement si $B_k \circ \sigma_{E,B}$, est vrai pour tout $k = 1, \ldots, n$ dans $\widetilde{I}_E$.

THÉORÈME 5.5.1 *Soient E un programme défini, B un but $\leftarrow B_1,\ldots,B_n$ et une réponse $\sigma_{E,B}$ à $E\cup\{B\}$ telle que $B_k\circ\sigma_{E,B}$ soit filtré pour tout $k=1,\ldots,n$. Alors les propositions suivantes son équivalentes :*

(i) *$\sigma_{E,B}$ est correcte.*

(ii) *$B_k\circ\sigma_{E,B}$, est vrai pour tout $k=1,\ldots,n$ dans tout modèle de Herbrand de E.*

(iii) *$B_k\circ\sigma_{E,B}$, est vrai pour tout $k=1,\ldots,n$ dans $\widetilde{I}_E$.*

Remarquons que ce théorème est vrai si $B_k\circ\sigma_{E,B}$ soit filtré pour tout $k=1,\ldots,n$.

En récapitulant nous pouvons dire qu'il y a trois étapes lors de la construction d'un programme :

(1) Trouver un modèle de Herbrand à partir des modèles non-Herbrand.

(2) Trouver le modèle minimal de Herbrand.

(3) Extraire toute la connaissance positive qui est incluse dans le modèle minimal de Herbrand afin de l'utiliser pour la construction du programme. En effet tous les éléments d'un programme qui se vérifient, font partie du modèle minimal de Herbrand.

ASCÈSE 5.17 *Soit le programme $E=\{p(a)\leftarrow\}$. Considérons le but $B=\{\leftarrow p(X)\}$. Montrer que la substitution identité n'est pas une réponse correcte à $E\cup\{B\}$.*

5.6 Exercices

EXERCICE 5.1 *Donner l'univers et la base de Herbrand pour le programme suivant :*

$$q(X,g(X))\rightarrow p(f(X))\ ,\rightarrow\ q(a,g(b))\ ,\ \rightarrow q(b,g(b))$$

EXERCICE 5.2 *Même chose pour le programme*

$$p(X,Y,Z)\rightarrow p(s(X),Y,s(Z))\ \textit{et}\ p(0,X,X)$$

EXERCICE 5.3 *Supposons que les constantes d'un univers de Herbrand sont a,b,c et d. Soit I l'interprétation de Herbrand :*

$$\{p(a),p(b),q(a),q(b),q(c),q(d)\}$$

Trouver parmi les fbf suivantes, celles qui sont vraies pour l'interprétation I :

(1) $\forall X\ p(X)$

(2) $\forall X\ q(X)$

(3) $\exists X\,(q(X)\wedge p(X))$

(4) $\forall X\,(q(X)\rightarrow p(X))$

(5) $\forall X\,(p(X) \rightarrow q(X))$

EXERCICE 5.4 *Vérifier sur un exemple que la relation « la substitution* σ *est plus générale que la substitution* τ *» n'est pas antisymétrique.*

EXERCICE 5.5 *Soit* σ *une substitution de renommage. Montrer que dans ce cas* σ *dispose d'une substitution inverse, notée* σ^{-1}*, telle que* $\sigma^{-1} \circ \sigma = \sigma \circ \sigma^{-1} = \varepsilon$.

EXERCICE 5.6 *Soit* θ *l' UPG des termes s et t et* σ *une substitution de renommage. Montrer que* $\theta \circ \sigma$ *est aussi un UPG de s et t.*

EXERCICE 5.7 *Trouver tous les ensembles non appariés de deux structures suivantes :*

$u_1 = p(X, f(g(a), b), c)$ *et* $u_2 = p(a, f(Y, b), Z)$

EXERCICE 5.8 *Appliquer l'algorithme de l'unification aux programmes suivants :*

(1) $u_1 = p(f(a), g(X))$ *et* $u_2 = p(Y, Y)$

(2) $u_1 = p(a, X, h(g(Z)))$ *et* $u_2 = p(Z, h(Y), h(Y))$

(3) $u_1 = p(X, X)$ *et* $u_2 = p(Y, f(Y))$

EXERCICE 5.9 *Soit le programme* $E = \{impair(s(0)) \leftarrow,\ impair(s(s(X))) \leftarrow impair(X)\}$. *Calculer* $\widetilde{I}_E$.

EXERCICE 5.10 *Soit le programme* $E = \{p(f(X)) \leftarrow p(X),\ q(X) \leftarrow p(X)\}$. *Calculer* $\widetilde{I}_E$.

5.A APPENDICE AU CHAPITRE 5

5.A.1 Utilisation de l'unification par Prolog

Deux prédicats peuvent s'unifier si

(1) leurs noms sont identiques, et

(2) les variables de ces deux prédicats peuvent être substituées par des termes de sorte que, après substitution, les deux prédicats sont identiques.

Cette définition peut être appliquée à plus de deux prédicats.
Exemple : Soit le programme Prolog composé par le fait

```
\texttt{date(10, X, 2007).}
```

Alors à la question

```
\texttt{?-date(Jour, Mois, Annee).
```

Prolog repondra

```
\texttt{Jour = 10,}
\texttt{Annee = 2007}
```

c'es-à-dire il a fait la substitution suivante : `(Jour/10, X/Mois, Annee/2007)`. Comme la substitution `(X/Mois)` est une substitution de renommage (d'une variable par une autre variable), Prolog nous épargne de son affichage.

Remarquons que le fait est incomplet car à la place du mois il contient une variable, mais Prolog n'a pas été empeché pour fournir une réponse, bien sûr incomplète elle aussi.

5.A.2 Fonctionnement (un peu moins simplifié) de Prolog

De ce qui précède nous pouvons comprendre qu'un programme en Prolog est un ensemble des règles de Horn. Il s'agit soit des règles de Horn strictes du type `p` ← `q.` , soit des faits `a.` , qu'on peut assimiler à `a` ← `.` . La conséquence de la règle s'appelle *tête de la règle*.

Une question ? ← `r.` est la formulation d'une hypothèse qui demande à être vérifiée par les règles du programme.

Pour ce faire, Prolog examinera les têtes des règles. Il retiendra les règles pour lesquelles leurs têtes peuvent s'unifier avec le prédicat de la question. S'il n'y a pas une telle règle, Prolog repondra par la négative. S'il existe au moins une telle règle, il ya deux possibilités :

(1) soit la règle en question est un fait et, donc, la réponse de Prolog serait immédiate et positive ;

(2) soit la règle en question est stricte. Dans ce cas Prolog fera les substitutions nécessaires pour unifier la question et la tête de la règle et il propagera ces substitutions aux hypothèses de la règle. Ces hypothèses, sous leur forme nouvelle après substitution, deviendront les nouvelles questions à vérifier à la place de la question du départ.

Exemple. Soit le programme

```
p(X) :- q(X).
q(a).
q(b).
```

Supposons qu'on pose la question

```
?- p(b).
```

Prolog fonctionnera comme suit :

(1) unifiera la tête de la règle `p(X) :- q(X).` avec la question à l'aide de la substitution `(X/b)` qui dès lors deviendra `p(b) :- q(b).` ;

(2) remplacera la question `?- p(b).` de départ par la question `?- q(b).` issue de l'hypothèse de la règle et recommencera ;

(3) trouvera le fait `q(b).` qui repond à la question posé, et

(4) donnera la réponse `X=b`.

5.A.3 Modèle minimal de Herbrand

Un programme en Prolog est un programme défini, c'est-à-dire un programme qui ne comporte pas des clauses de Horn négatives. L'intérêt d'un tel programme vient du fait qu'il a toujours un modèle, à savoir la base de Herbrand du programme. Par conséquent un programme défini ne peut pas être insatisfiable.

En réalité ce qui est intéressant ici est la connaissance du plus petit modèle de Herbrand que nous pouvons construire et que nous avons appelé modèle minimal de Herbrand. Cette connaissance permettra de vérifier ce que réellement fera le programme pendant son exécution.

Considérons, en effet, un programme en Prolog E et une clause p. Soit $\widetilde{I}_E$ le modèle minimal de Herbrand pour E. Alors nous avons $E \models p$ si et seulement si $p \in \widetilde{I}_E$. La partie "seulement si" implique que tout ce que un programme Prolog peut produire comme résultat se trouve dans le modèle minimal de Herbrand.

Exemple. Soit le programme

$$E = \left\{ \begin{array}{l} \texttt{boisson(toto,grenadine).} \\ \texttt{boisson(jojo,pastis).} \\ \texttt{boisson(lolo,pastis).} \\ \texttt{paire(X,Y) :-boisson(X,Z),boisson(Y,Z),X==Y.} \end{array} \right\}$$

L'univers de Herbrand est $\mathcal{U}_E = \{\texttt{toto}, \texttt{jojo}, \texttt{lolo}, \texttt{grenadine}, \texttt{pastis}\}$. La base de Herbrand est

$B_E = \{\texttt{boisson(jojo,jojo)}, \texttt{boisson(jojo,lolo)}, \texttt{boisson(jojo,pastis)}, \ldots,$
$\texttt{paire(jojo,jojo)}, \ldots\}.$

En utilisant l'opérateur de la conséquence immédiate, on a

- $\mathcal{T}_E \uparrow 0 = \emptyset$
- $\mathcal{T}_E \uparrow 1 = \{\texttt{boisson(toto,grenadine)}, \texttt{boisson(jojo,pastis)}, \texttt{boisson(lolo,pastis)}\}$
- $\mathcal{T}_E \uparrow 2 = \mathcal{T}_E \uparrow 1 \cup \{\texttt{paire(jojo,lolo)}, \texttt{paire(lolo,jojo)}\}$
- $\mathcal{T}_E \uparrow 3 = \mathcal{T}_E \uparrow 2$, c'est-à-dire $\mathcal{T}_E \uparrow 2$ est un point fixe pour l'opérateur de la conséquence immédiate. Par conséquent $\widetilde{I_E} = \mathcal{T}_E \uparrow 2$.

Nous pouvons vérifier, en utilisant Prolog, que les réponses données par le programme sont celles contenues dans le modèle minimal.

5.A.4 Exercice

Considérons le graphe orienté dont la base de données correspondante en Prolog est

```
gr(a,b,2).
gr(a,g,6).
gr(b,e,2).
gr(b,c,7).
gr(g,e,1).
gr(g,h,4).
gr(e,f,2).
gr(f,c,3).
gr(f,h,2).
gr(c,d,3).
gr(h,d,2).
```

(1) Écrire en Prolog le prédicat `chemin(X,Y)` qui réussit s'il y a un chemin dans le graphe qui mène de `X` à `Y`.

(2) Vérifier, en utilisant le modèle minimal de Herbrand, la validité de votre programme.

6

RÉSOLUTION SLD

Ce chapitre est consacré à l'étude de la *résolution SLD*, c'est-à-dire de la technique d'inférence qui s'applique à des programmes logiques définis. Cette technique est issue du principe de la résolution introduit par J. A. Robinson en 1965 pour des programmes logiques plus généraux que les programmes logiques définis. La résolution SLD fut présentée pour la première fois par R. Kowalski en 1974 et signifie résolution **L***inéaire avec fonction de* **S***élection pour clauses* **D***éfinies* (Linear resolution for Definite clauses with Selection function).

6.1 Présentation de la résolution SLD

Pour démontrer qu'un fait peut être déduit d'un programme logique, on suppose que le contraire de ce fait existe dans le programme et en utilisant, de façon répétitive, le modus–ponens et la règle de l'élimination du quantificateur universel on établit un résultat absurde.

Considérons par exemple le fait $F = B_1 \wedge \ldots \wedge B_m$ et un programme défini E. Pour démontrer que F est une conséquence logique de E on place la négation de ce fait dans le programme E, c'est-à-dire on rajoute à E le but B représenté par $\leftarrow B_1 \ldots, B_m$ et, par la suite, on cherchera à montrer que le programme E, ainsi enrichi, contient la clause vide. Le principe de résolution – qui est un algorithme de raisonnement – peut nous aider à résoudre ce problème. Une étape de l'algorithme

de raisonnement transforme le but – qui est un ensemble de formules atomiques – en un nouveau but, c'est-à-dire en un nouvel ensemble de formules atomiques. Pour ce faire l'algorithme sélectionne une formule atomique parmi celles du but, mettons la formule B_k, et une clause de E, mettons la clause $C : A \leftarrow A_1, \ldots, A_n$, qui *filtre* B_k, ce qui signifie que B_k et A sont unifiables. Supposons que θ_{A,B_k} soit leur UPG. Alors le nouveau but est obtenu en remplala formule atomique B_k du but par $\left(A_1 \circ \theta_{A,B_k}, \ldots, A_n \circ \theta_{A,B_k}\right)$ qui est constituée par les conditions de la clause C sur lesquelles nous avons effectué la substitution θ_{A,B_k}. Si, en répétant cette démarche, on aboutit à la clause vide, on obtient alors la négation du but. Comme le but $\leftarrow B_1, \ldots, B_m$ peut aussi s'écrire sous la forme $\forall X_1 \ldots, X_l \neg (B_1 \wedge \ldots \wedge B_m)$, on en déduit que $\neg \exists X_1 \ldots, X_l \, (B_1 \wedge \ldots \wedge B_m)$. Ce résultat est obtenu car nous avons mis dans E la négation du fait F que nous voulions démontrer et nous concluons ainsi que cette négation est absurde eu égard au programme E.

Nous pouvons observer que les étapes de l'algorithme sont non déterministes, c'est-à-dire le choix de la formule atomique B_k du but n'est pas déterminé d'avance mais par contre il est libre. De plus, B_k étant choisie, il est possible qu'il y ait plusieurs clauses dans E qui filtrent B_k et, par conséquent, le choix de la clause à utiliser pour l'unification est aussi libre. Nous pouvons ainsi construire plusieurs solutions et même on peut avoir une infinité de solutions. Il est aussi possible, ayant sélectionné B_k, de ne pas pouvoir trouver dans E une clause qui filtre cette formule atomique et, dans ce cas, de ne pas pouvoir construire une solution.

Essayons maintenant de formaliser cette démarche.

DÉFINITION 6.1.1 (Principe de résolution SLD) *Soient un but B défini par la question $\leftarrow B_1, \ldots, B_k, \ldots, B_m$ et un programme défini E contenant la clause $C : A \leftarrow A_1, \ldots, A_n$. Alors nous pouvons dériver de B et C un nouveau but B' défini par $\leftarrow (B_1, \ldots, B_{k-1}, A_1, \ldots, A_n, B_{k+1,}, \ldots, B_m) \circ \theta_{A,B_k}$ si*

(i) *B_k est un atome, appelé l'*atome choisi *dans B,*

(ii) *A filtre B_k, et*

(iii) *θ_{A,B_k} est l'UPG de A et B_k.*

Alors B' est appelée la résolvante *de B et C .*

Cette définition introduit la règle d'inférence suivante :

(R-SLD) $$\frac{\forall \neg (B_1 \wedge \ldots \wedge B_{k-1} \wedge B_k \wedge B_{k+1} \wedge \ldots \wedge B_m) \quad \forall (A \leftarrow A_1, \ldots, A_n)}{\forall \neg (B_1 \wedge \ldots \wedge B_{k-1} \wedge A_1 \wedge \ldots \wedge A_n \wedge B_{k+1} \wedge \ldots \wedge B_m) \circ \theta_{A,B_k}}$$

où

- $B_1, \ldots, B_k, \ldots, B_m$ et $A, A_1, \ldots, A_n$ sont des formules atomiques.
- θ_{A,B_k} est l'UPG de A et B_k.

Remarquons que les deux ensembles des variables de l'antécédent de cette règle d'inférence doivent être distincts. Dans le cas contraire on peut utiliser une substitution de renommage afin de satisfaire à cette exigence.

Pour obtenir la clause vide on est amené à appliquer plusieurs fois itérativement cette règle d'inférence. Lors de ces itérations successives il faut que l'ensemble de variables de la clause C, utilisée pendant une itération, soit disjoint des ensembles de variables des clauses utilisées pendant les itérations précédentes et de l'ensemble de variables du but. Afin de satisfaire à cette exigence on doit utiliser, comme nous venons de voir, la substitution de renommage que nous pouvons mettre au point de manière très simple en portant des indices au but et aux différentes clauses utilisées. Ainsi le but initial B sera noté B_0 ce qui donne $\leftarrow B_{0_1}, \ldots, B_{0_m}$. La clause C, qui sera utilisée pour évaluer la résolvante B' de $B = B_0$ et de C, sera notée C_1 ce qui donne $A_1 \leftarrow A_{1_1}, \ldots, A_{1_n}$. L'UPG θ_{A,B_k} entre la formule atomique $B_k = B_{0_k}$ du but $B = B_0$ et la formule atomique $A = A_1$ de la clause $C = C_1$ sera noté θ_1. Enfin la résolvante B' de B_0 et C_1 sera notée B_1. En poursuivant cette démarche nous avons que chaque nouveau but B_{i+1} est la résolvante du but B_i et de la clause C_{i+1} par application de l'UPG θ_{i+1}. Cette démarche peut être finie ou infinie. Il y a en fait deux cas qui permettent l'arrêt du processus :

- soit $B_i = \emptyset$, c'est-à-dire la clause vide,
- soit le sous-but sélectionné B_{i_k} ne peut pas être unifié avec la tête d'une clause de E.

DÉFINITION 6.1.2 *Soient un but B défini par $\leftarrow B_1, \ldots, B_m$ et un programme défini E. Une* dérivation SLD *de $E \cup \{B\}$ est une suite finie ou infinie des triplets $\{(B_i, C_{i+1}, \theta_{i+1})\}_i$ avec $B_0 = B$, C_{i+1} une clause de E et θ_{i+1} UPG entre la tête de C_{i+1} et un sous-but B_{i_k} du but B_i de sorte que B_{i+1} dérive de B_i et de C_{i+1} en utilisant θ_{i+1}.*

Une dérivation SLD peut être représentée par une arborescence telle que

- *la racine est le but $B = B_0$.*
- *B_{i+1} est un successeur de B_i. L'arc entre B_i et B_{i+1} est étiqueté par C_{i+1} et θ_{i+1}.*

Si la dérivation est finie, le nombre d'éléments de la suite des triplets $\{(B_i, C_{i+1}, \theta_{i+1})\}_i$ s'appelle longueur de la dérivation.

DÉFINITION 6.1.3 *Soient un but B défini par $\leftarrow B_1, \ldots, B_m$ et un programme défini E. Une* réfutation SLD *ou* dérivation SLD réussie *de $E \cup \{B\}$ est une dérivation SLD finie dont le dernier élément est la clause vide*[1].

Si, de plus, les unificateurs θ_i de cette dérivation ne sont pas les unificateurs les plus généraux, alors on parle de réfutation SLD sans restriction.

1. Il faut remarquer qu'on parle ici de réfutation SLD et on précise que la dérivation SLD est réussie si $E \cup \{B\}$ aboutit à la clause vide, c'est-à-dire s'il échoue. Dans les chapitres précédents, on a établi que E induisait B si $E \cup \{\neg B\}$ échouait. La contradiction n'est qu'apparente. En effet dans les chapitres précédents, B était une clause qu'on transformait en question, c'est-à-dire à $\neg B$, tandis que dans tout le chapitre actuel, B est déjà sous forme d'une question et donc il n'est point nécessaire de la transformer en question.

Dans le cas où une dérivation finie n'aboutit pas à la clause vide on parle de dérivation échouée.

Dans le cas d'une dérivation réussie ou échouée, la suite $\{\theta_i\}_i$ *des UPG restreinte aux variables du but B est appelée la* réponse calculée de la dérivation.

Étant donné un programme E, un *but* est une question posée à la base de données constituée par les règles et les faits de ce programme. La *réponse calculée* est la réponse du programme à cette question.

ASCÈSE 6.1 *Soit le programme défini :*

- `p( X,Y) ← q( X,Z), r(Z,Y)`
- `p(X,Y) ← r(Y,X)`
- `q(X,Y) ← r(Y,X)`
- `r(a,b) ←`

Déterminer une dérivation SLD du but ← `p(b,X)` *et donner la réponse calculée de la dérivation.*

Nous terminons ce paragraphe par quelques notions qui seront utiles pour établir la justesse et la complétude de la résolution SLD.

LEMME 6.1.1 *Soient un but B et un programme défini E. Si* $E \cup \{B\}$ *a une réfutation SLD sans restriction de longueur n avec unificateurs* $\theta'_1, \ldots, \theta'_n$ *, alors* $E \cup \{B\}$ *a aussi une réfutation SLD avec UPG* $\theta_1, \ldots, \theta_n$ *et telle qu'il existe une substitution* σ *avec* $\theta'_1 \circ \ldots \circ \theta'_n = \theta_1 \circ \ldots \circ \theta_n \circ \sigma$.

LEMME 6.1.2 *Soient un but B, un programme défini E et une substitution* ρ. *S'il existe une réfutation SLD de* $E \cup \{B \circ \rho\}$ *de longueur n avec* $\theta'_1, \ldots, \theta'_n$ *les UPG, alors* $E \cup \{B\}$ *a aussi une réfutation SLD de même longueur, avec* $\theta_1, \ldots, \theta_n$ *les UPG correspondants et telle qu'il existe une substitution* σ *avec* $\theta'_1 \circ \ldots \circ \theta'_n = \theta_1 \circ \ldots \circ \theta_n \circ \sigma$.

DÉFINITION 6.1.4 *Soit E un programme défini. L'*ensemble des succés *de E est l'ensemble de tous les éléments* $B \in \mathcal{B}_E$ *de la base de Herbrand tels que* $E \cup \{\neg B\}$ *ait une réfutation SLD.*

6.2 Justesse de la résolution SLD

La résolution SLD est une technique d'inférence pour laquelle il faut établir sa *justesse*, c'est-à-dire le fait que les résultats obtenus lorsqu'on utilise cette technique sur un programme E sont justes. Ici la justesse des résultats signifie que ces résultats sont des conséquences logiques du programme. Nous avons le théorème suivant :

THÉORÈME 6.2.1 (Justesse de la résolution SLD) *Soient un but B, sous forme de question, et un programme défini E. Toute réponse calculée pour $E \cup \{B\}$ est une réponse correcte pour $E \cup \{B\}$.*

De ce théorème découlent les deux corollaires suivants :

COROLLAIRE 6.2.1 *Soient un but B, sous forme de question, et un programme défini E. S'il existe une réfutation SLD de $E \cup \{B\}$, alors $E \cup \{B\}$ est insatisfiable.*

COROLLAIRE 6.2.2 *L'ensemble des succès d'un programme défini E est égal à son plus petit modèle de Herbrand $\widetilde{I}_E$.*

Une version renforcée du dernier corollaire est fournie par le

THÉORÈME 6.2.2 *Soient un but B défini par $\leftarrow B_1, \ldots, B_m$ et un programme défini E. Si $E \cup \{B\}$ a une réfutation SLD de longueur n et $(\theta_i)_{i=1,n}$ est la réponse calculée de la réfutation, alors $\bigcup_{k=1}^{n} B_k \circ \theta_1 \circ \cdots \circ \theta_n \subseteq T_E \uparrow n$.*

6.3 Complétude de la résolution SLD

La deuxième question qu'il faut résoudre concernant la résolution SLD est sa *complétude*, c'est-à-dire sa capacité de fournir toutes les réponses correctes à un but donné.

THÉORÈME 6.3.1 *Soient un but B, sous forme de question, et un programme défini E. Si $E \cup \{B\}$ est insatisfiable, alors il existe une réfutation SLD de $E \cup \{B\}$.*

THÉORÈME 6.3.2 *Soient un atome A et un programme défini E. Si $\forall(A)$ est une conséquence logique de E, il existe une réfutation SLD de $E \cup \{A\}$ avec comme réponse calculée la substitution identité.*

THÉORÈME 6.3.3 (Complétude de la résolution SLD) *Soient un but B, sous forme de question, et un programme défini E. Pour toute réponse correcte σ de $E \cup \{B\}$ il existe une réponse calculée θ de $E \cup \{B\}$ et une substitution ρ telle que $\sigma = \theta \circ \rho$.*

Le théorème de complétude établit l'existence d'une réfutation qui construit une réponse plus générale que toute réponse correcte donnée. Mais, malheureusement il n'indique pas la manière d'obtenir cette réfutation c'est-à-dire la manière de choisir à chaque étape le sous-but et la clause à unifier. Il s'agit en réalité d'un problème ouvert dont quelques éléments de réponse seront donnés à la section suivante et aussi pendant les cours du langage de programmation Prolog et des systèmes experts.

ASCÈSE 6.2 *Soit le programme défini :*

- `p(f(Y)) ←`
- `q(a) ←`

et le but ← `p( X)`

Déterminer une réponse correcte ainsi que la réponse calculée de dérivation et vérifier que celle-ci est plus générale que celle-là.

6.4 Algorithme de résolution SLD

Nous présentons d'abord le principe de résolution par l'intermédiaire du

THÉORÈME 6.4.1 (Principe de la résolution) *Soit un programme défini E. Considérons deux clauses C_1 et C_2 de E. Soient deux termes t_1 et t_2 qui peuvent s'unifier et tels que $t_1 \in C_1$ et $\neg t_2 \in C_2$. Considérons aussi une substitution de renommage θ telle que l'ensemble de variables libres de $C_1 \circ \theta$ soit disjoint de l'ensemble de variables libres de C_2 . Soit σ l'UPG de $C_1 \circ \theta$ et C_2. Posons*

$$R = ((C_1 \setminus \{t_1\}) \circ \theta \vee (C_2 \setminus \{\neg t_2\})) \circ \sigma$$

Alors les programmes E et $E \cup \{R\}$ sont logiquement équivalents.

Nous pouvons remarquer que R est la *résolvante* du but t_2 et de la clause C_1.

Ce théorème qui est une généralisation de l'algorithme de résolution que nous avons vu au calcul propositionnel, il ne suffit pas pour assurer la complétude du principe de la résolution. Nous aurons cette complétude si on rajoute la *règle de factorisation*.

DÉFINITION 6.4.1 *Considérons une clause C de la forme $t_1 \vee t_2 \vee \cdots \vee t_n$ telle que t_1 et t_2 sont unifiables avec σ leur UPG. La clause*

$$C' = (C \setminus \{t_2\}) \circ \sigma$$

s'appelle facteur *de C.*

THÉORÈME 6.4.2 (Règle de factorisation) *Considérons une clause C de la forme $t_1 \vee t_2 \vee \cdots \vee t_n$ telle que t_1 et t_2 sont unifiables avec σ leur UPG. Le facteur*

$$C' = (C \setminus \{t_2\}) \circ \sigma$$

de C est une conséquence logique de C.

L'algorithme de résolution SLD est le suivant :

(1) *Données :* Un programme défini E représenté comme un ensemble de clauses.

(2) Si $\perp \in E$, alors fin et sortie en succès.
Sinon

(3) Choisir deux clauses C_1 et C_2 dans E telles que

- $t_1 \in C_1$,
- $\neg t_2 \in C_2$,
- les termes t_1 et t_2 sont unifiables.

(4) Trouver l'UPG de t_1 et t_2 et déterminer la clause résolvante R.

(5) Remplacer E par $(E - \{C_1, C_2\}) \cup \{R\}$ et retour en 2.

Remarquons que la meilleure fapour que les termes t_1 et t_2 soient unifiables est de prendre $t_1 = t_2 = t$. Dans ce cas l'algorithme de résolution est fondé sur la tautologie suivante :

$$\textbf{(TR)} \qquad (t \vee F) \wedge (\neg t \vee G) \rightarrow F \vee G$$

où F, G sont des fbf.

ASCÈSE 6.3 *Considérons le programme défini suivant :*

- $p \leftarrow q, r$
- $p \leftarrow s$
- $q \leftarrow$
- $q \leftarrow s$
- $r \leftarrow$
- $s \leftarrow t$
- $s \leftarrow$

(1) Donner l'arbre de dérivation SLD pour le but $\leftarrow p$.

(2) Même chose si on retire du programme la 3e clause.

Nous avons déjà démontré que cet algorithme converge vers la solution, si celle-ci existe. Mais du fait qu'il est un algorithme non déterministe – à cause du mot « choisir » de l'étape 3 – nous ne pouvons rien affirmer au sujet de la vitesse de convergence et, donc, de l'efficacité de cet algorithme. Afin d'améliorer l'efficacité de l'algorithme nous disposons des stratégies heuristiques([2]) pour effectuer le choix des clauses en étape 3. Ainsi nous avons :

- *Stratégie fondée sur le nombre d'atomes dans une clause*. On sait que pour obtenir la clause vide $\perp$ il faut obtenir l'implication $vrai \rightarrow faux$. De faempirique on constate qu'on arrive plus rapidement à cette implication si on utilise des clauses avec un nombre réduit de formules atomiques. L'idéal serait qu'à chaque itération de l'algorithme nous puissions trouver une clause contenant un seul terme, par exemple t, et que $\neg t$ se trouve dans une autre clause que nous pouvons mettre sous la forme $\neg t \vee F$. La résolution remplace les clauses t et $\neg t \vee F$ par F, c'est-à-dire par une clause qui contient un nombre moins important de formules atomiques.
- *Stratégie fondée sur l'utilisation des entrées*. À chaque itération, sauf à la première, on utilise une clause qui existe dans la base de données et une

2. Un algorithme heuristique ou une heuristique est un algorithme fondé sur des principes empiriques et pour lequel nous ne pouvons pas obtenir une preuve formelle de son efficacité.

clause obtenue pendant les étapes précédentes de l'algorithme de résolution. On a constaté de faempirique, que cette démarche aboutit à une arborescence de la dérivation SLD plus courte que d'autres arborescences obtenues différemment.

– *Stratégie dite de la résolution linéaire*. Il s'agit d'une modification de la précédente. À chaque itération, sauf à la première, on utilise une clause qui soit elle existe dans la base de données, soit elle est une des antécédentes de la deuxième clause dans l'arbre de dérivation SLD. La deuxième clause est toujours une clause obtenue pendant les étapes précédentes de l'algorithme de résolution.

– *Stratégie fondée sur l'action de subsumer*. À chaque itération on n'utilise pas des clauses qui nous donneraient comme résultat une clause qui est subsumée par une autre clause déjà existante dans la base de données. Ainsi la base de données reste, pendant l'algorithme de résolution, de taille relativement modeste.

Aucune de ces différentes stratégies n'est utilisée par Prolog. Par contre beaucoup de programmes générateurs des Systèmes Experts utilisent des stratégies de cette nature.

ASCÈSE 6.4 *Soit le programme défini :*

- `A ←`
- `B ← A`
- `C ← B`

Montrer que le but `← B` *est une conséquence logique du programme.*

6.5 Exercices

EXERCICE 6.1 *Considérons la base de données suivante :*

- `Chaque gamin qui se respecte adore les gâteaux préparés par un bon pâtissier.`
- `Toto adore les gâteaux.`
- `Koko est un bon pâtissier.`
- `Toto achète ses gâteaux chez Koko.`

Déterminer une dérivation SLD du but

`X est un gamin qui se respecte`

et donner la réponse calculée de la dérivation.

EXERCICE 6.2 *Considérons le programme :*

- `← plus(zero,N,N)`
- `plus(suc(K),L,suc(M)) ← plus(K,L,M)`

Montrer que `2+2=N`*, où* `N` *nombre naturel, est une conséquence logique de ce programme.*

EXERCICE 6.3 *Considérons une structure abstraite des termes, construite avec les définitions suivantes :*

*(1) Un terme est appelé $f-$*lié *avec un terme T si et seulement s'il est de la forme $f(T,Y)$ pour tout terme Y.*

(2) Un terme X est appelé bizarre *si et seulement s'il est $f-$lié avec lui-même.*

(3) Un terme X est appelé fou *si et seulement s'il est la deuxième sous-structure d'un terme bizarre.*

Y a-t-il dans cette structure des termes fous ? Que se passe-t-il si on n'utilise pas la vérification de l'occurrence lors de l'unification ?

EXERCICE 6.4 *Montrer que si dans un groupe, chaque élément est son propre inverse, alors le groupe est commutatif.*

EXERCICE 6.5 (Petit exercice de recapitulation) *Un problème de Lewis Carroll.- Soient les phrases suivantes :*

(i) *Les fleurs colorées ont toujours un parfum.*

(ii) *Toto n'aime pas les fleurs qui n'ont pas poussé en plein air.*

(iii) *Il n'existe pas de fleurs qui ont poussé en plein air et qui ne sont pas colorées.*([3])

(iv) *Toto n'aime pas les fleurs qui n'ont pas de parfum.*

On se propose de faire le travail suivant.

(1) En utilisant le calcul propositionnel, mettre ces phrases sous forme de fbf.

(2) En utilisant l'algorithme de Davis–Putnam, déduire la 4e fbf à l'aide de trois premières

(3) En utilisant une logique du premier ordre, mettre ces phrases sous forme de fbf.

(4) Déduire la 4e fbf à l'aide de trois premières.

(5) Calculer l'univers et la base d'Herbrand de quatre fbf.

(6) On modifie les 2e et 4e fbf ci-dessus comme suit :

(ii)' *Personne n'aime les fleurs qui n'ont pas poussé en plein air.*

(iv)' *Personne n'aime les fleurs qui n'ont pas de parfum.*

Calculer l'univers et la base d'Herbrand pour ce nouvel ensemble de fbf.
En utilisant l'ensemble des fbf modifiées, déduire la 4e fbf des trois précédentes.
Au vu de cette déduction que peut-on dire des éléments de l'univers d'Herbrand ?

(7) Traduire les fbf en clauses de Horn.

(8) Appliquer l'algorithme de résolution pour déduire la 4e clause de trois premières.

---

3. Ce qui prouve que les connaissances horticoles de L.Carroll étaient relativement limitées.

6.A APPENDICE AU CHAPITRE 6
Prolog et SLD

L'algorithme SLD (p. 96) est non déterministe à cause du fait qu'à l'étape 3 on choisit

- le littéral $\neg B_k$ qui est un élément du but, et
- la clause C dont la tête A est unifiable avec B_k.

Le choix de la clause C se fait habituellement selon une stratégie propre au langage utilisé. Prolog choisit systématiquement la première clause de la base de données qui est unifiable avec le but. Il est évident que si l'ordre des clauses dans la base de données, c'est-à-dire le programme défini E, se trouve modifié, l'ordre des réponses de Prolog sera aussi modifié.

Nous pouvons étudier ce point en utilisant l'arbre de dérivation SLD présenté à l'ascèse 6.3. En effet chaque sommet de l'arbre de dérivation SLD correspond au double choix, à savoir choix du litéral du but et choix de la clause dont la tête est unifiable avec le littéral, présenté ci-dessus. On peut comprendre que pour un programme défini, nous pouvons avoir plusieurs arbres de dérivation SLD, selon l'ordre des clauses dans les programme.

L'étude de la trace de l'exécution par Prolog d'un programme montre que Prolog choisit toujours le littéral le plus à gauche dans la liste des buts à vérifier. Il opère ainsi selon une stratégie qui s'appelle *en profondeur d'abord* et qui permet de parcourir d'abord la branche de l'arbre qui est la plus à gauche. Ensuite il y a retour en arrière et parcours des autres branches, non encore explorées, selon le même principe : parcours de la branche la plus à gauche qui n'est pas encore explorée.

EXERCICE 6.6 *Soient les trois programmes Prolog suivants :*

```
p(X,Z) :- a(X,Y), p(Y,Z).
p(X,X).
a(b,c).
% tst :- p(X,c).

p1(X,Z) :- a1(X,Y), p1(Y,Z).
a1(b,c).
p1(X,X).
% tst1 :- p1(X,c).

p2(X,Z) :- p2(Y,Z), a2(X,Y).
```

```
p2(X,X).
a2(b,c).
% tst2 :- p2(X,c).
```

(1) Établir leur arbre de résolution SLD selon la règle de Prolog (branche la plus à gauche).

(2) Éditer leurs traces en Prolog et vérifier la concordance avec les arbres de résolution correspondants.

7

PROGRAMMES GÉNÉRAUX OU NORMAUX

Nous avons travailler jusqu'à maintenant avec des programmes définis, ce qui exclut la possibilité d'avoir dans les premisses d'une règle des négations des clauses et, par voie de conséquence, d'avoir un but composé par des négations des clauses. Dans ce chapitre nous présentons les *programmes généraux* ou encore les *programmes normaux*, c'est-à-dire des programmes dont les clauses sont composés des littéraux, qui sont, comme nous avons déjà vu, soit des atomes, soit des négations des atomes. Nous terminons ce chapitre avec la résolution SLDNF qui est la résolution SLD appliquée aux programmes généraux.

7.1 Programmes logiques et information négative

Considérons un programme défini E et soit $\mathcal{B}_E$ la base de Herbrand de E. Chaque élément de $\mathcal{B}_E$ peut être induit logiquement de E, i.e. $\forall B \in \mathcal{B}_E$ on a $E \models B$ (voir Corollaire **??** et Théorème 5.3.3). Comme E est un programme défini, c'est-à-dire il contient des connaissances positives, la base de Herbrand contient seulement des faits positifs. Par conséquent on n'a pas la possibilité d'induire des connaissances négatives. Malgré tout nous avons besoin de savoir, par des méthodes d'induction logique, qu'un fait n'existe pas. Il s'agit essentiellement d'un problème

d'interprétation de la négation, c'est-à-dire trouver une méthode qui permet d'obtenir de l'information négative. Il existe plusieurs méthodes mais pour ce cours introductif nous allons nous restreindre aux trois plus anciennes parues simultanément dans le livre *Logic and Data Bases*, édité par H. Gallaire et J.Minker en 1978.

– *Hypothèse du monde fermé* (CWA – Closed World Assumption) qui fut introduite par R. Reiter. Selon cette hypothèse – qui est, en fait, une règle – toute connaissance qui ne fait pas partie explicitement d'une base de données ou qu'il ne peut pas être induite logiquement de cette même base de données, n'existe pas et donc c'est sa négation qui a la valeur de vérité « vraie ». Il s'agit d'une hypothèse très naturelle dans les bases de données relationnelles. Mais pour les bases de données logiques, que sont tous les programmes logiques, il en va tout autrement. En effet, formellement l'hypothèse du monde fermé introduit la règle d'inférence suivante :

(R-MF)
$$\frac{\vdash \neg (E \vDash A)}{\vdash \neg A}$$

En toute rigueur on aurait dû conclure que $E \vDash \neg A$ si on avait la preuve que A n'est pas une conséquence logique du programme E. Or cette affirmation n'est licite que si on fait fi de la base de Herbrand $\mathcal{B}_E$ de E et qu'on tient compte seulement de la base de données du programme. En effet, en général la base de Herbrand est un ensemble infini et de ce fait le problème de savoir si une fbf est une conséquence logique d'un programme est indécidable ([1]). Nous pouvons donc en conclure que l'hypothèse du monde fermé n'est pas, en général, applicable.

– *La négation considérée comme un échec*. Il s'agit d'une idée introduite par K. L. Clark. Nous pouvons la présenter comme étant une restriction de la règle (R-MF) dans le cas fini. En effet la règle (R-MF) stipule en substance que

> $\neg A$ réussit ssi A ne peut pas être prouvé

Clark a suggeré que, dans cette proposition, la négation soit consïdérée comme le résultat d'un échec fini, c'est-à-dire d'un échec qu'on peut obtenir, à l'aide de l'arbre de dérivation SLD, en un nombre fini d'étapes. Dans ce cas nous avons l'interprétation affaiblie suivante :

> $\neg A$ réussit ssi A échoue de façon finie

De façon formelle nous avons la définition suivante :

DÉFINITION 7.1.1 *Soit E un programme défini et B un but défini. Un arbre de*

1. ce qui signifie qu'il n' y a pas un algorithme qui, pour un programme donné E et un but B, peut répondre si $E \vDash B$ dans un temps fini. Notons que l'élève cultivé peut, avec profit, relier ce problème avec celui de l'arrêt dans les machines de Turing en se reportant à la bibliographie spécialisée et au cours de la complexité

dérivation SLD d'échecs fini pour $E \cup \{B\}$ est un arbre qui est fini et ne contient pas de branches de succés.

Si nous voulons établir que le programme E induit $\neg A$, il faut montrer que l'arbre de dérivation pour le but $B : \leftarrow \neg A$ est un arbre d'échecs finis. Nous introduisons ainsi un nouveau type de résolution, la résolution SLDNF qui est une résolution SLD avec « Negation » comme « Failure ». Elle complète la résolution SLD lorsque le but est sous forme d'un atome négatif, i.e. $B : \leftarrow \neg A$, en introduisant les règles suivantes :

$\neg A$ réussit ssi A donne un arbre SLD d'échecs fini
$\neg A$ donne un arbre SLD d'échecs fini ssi A réussi

Ainsi si $\neg A$ réussit, alors il est supprimé de la question et on passe à l'examen de la suite de la question. Si par contre il échoue de façon finie, alors la question est considérée comme aboutissant à un échec. Nous avons donc, en utilisant la résolution SLDNF, les résultats suivants :

- $E \vdash B \circ \theta$ s'il existe un arbre de dérivation SLDNF de $E \cup \{B\}$ avec réponse calculée θ.
- $E \vdash \neg B$ s'il existe un arbre de dérivation SLDNF d'échecs fini pour $E \cup \{B\}$.

Telle quelle la résolution SLDNF pose le problème de sa justesse et aussi de l'équivalence entre implication sémantique et implication syntaxique.

- *La complétion des programmes*. Afin d'éviter la non équivalence entre implications sémantique et syntaxique des programmes logiques avec des buts contenant des atomes négatifs, Clark a proposé de compléter la définition d'un programme afin de rendre explicite ce qui s'y trouve sous forme implicite. Ainsi pour un programme

$$E = \{p(X)) \leftarrow q(X),\ q(a) \leftarrow,\ q(b) \leftarrow\}$$

sa complétion est

$$\forall X\,(p(X) \leftrightarrow (\exists Y\,((q(Y)) \wedge ((Y = a) \vee (Y = b)))))$$

À la place du programme E on considère son complété et on utilise la résolution SLD sur ce programme. La complétion d'un programme introduit un nouveau prédicat, l'égalité, noté « = » qu'il ne faut pas confondre avec d'autres prédicats utilisés éventuellement par le programme E. Remarquons aussi que la complétion d'un programme revient à interpréter l'implication comme une équivalence. Le problème qui surgit est celui de la consistance. En effet il est possible d'avoir un programme E consistant, obtenir le complété de E et découvrir qu'il n'est pas consistant.

En résumant nous pouvons dire que la caractéristique commune des trois méthodes ci-dessus est qu'en essayant de résoudre le problème posé par le trai-

tement des buts avec des atomes négatifs, on en crée d'autres qui sont aussi fondamentaux que la négation. Il faut donc, si nous voulons résoudre les différents problèmes qui se posent, de revoir les concepts utilisés et leurs relations.

7.2 Relations d'inférence non monotones

Remarquons d'abord, qu'avec la prise en compte de la négation nous avons abordé, sans crier gare, le domaine de la *logique non monotone*. En effet jusqu'à maintenant les relations d'inférence, soit sémantique $\models$, soit syntaxique $\vdash$, que nous avons utilisées, ont été *monotones*, c'est-à-dire obéissaient à la définition :

DÉFINITION 7.2.1 *La relation d'inférence* $\models$ *(resp.* $\vdash$*) est* monotone *si elle satisfait à la propriété*

$A \models B$ *implique* $A \cup A' \models B$

(resp. $A \vdash B$ *implique* $A \cup A' \vdash B$*), où* A *est un ensemble de clauses et* B *un littéral.*

La logique classique des prédicats est une logique monotone et, par conséquent, la règle d'inférence (R-SLD) du chapitre précédent concernant la résolution SLD est une règle d'inférence monotone. Par contre l'hypothèse du monde fermé, la négation considérée comme un échec et la complétion des programmes introduisent la non monotonicité comme conséquence de la dérivation des atomes négatifs par des programmes définis. Car nous avons par exemple pour la clause $p \leftarrow q$ que $\{p \leftarrow q\} \models \neg p$ (resp. $\{p \leftarrow q\} \vdash \neg p$) en utilisant n'importe quelle méthode parmi les trois que nous venons de citer. Mais si on considère l'union de cette clause avec $q \leftarrow$, alors nous avons $\{p \leftarrow q\} \cup \{q\} \leftarrow \models p$ (resp. $\{p \leftarrow q\} \cup \{q\} \leftarrow \vdash p$).

Il y a un énorme domaine concernant l'étude des logiques non monotones mais ces logiques sont en dehors du cadre de ce cours. Néanmoins du fait que la négation introduit une logique non monotone, nous présentons les propriétés nécessaires pour qu'une relation d'inférence non monotone soit logique.

- Coupure : $A \mid\sim B$ et $A \cup B \mid\sim C$ implique $A \mid\sim C$ ([2]).
- Monotonicité faible : $A \mid\sim B$ et $A \mid\sim C$ implique $A \cup B \mid\sim C$.
- Rationalité : $\neg (A \mid\sim B)$ et $A \mid\sim C$ implique $A \cup B \mid\sim C$.

Dans la suite nous examinerons sous le cadre de la non monotonicité, que nous venons d'établir, la résolution SLDNF et la complétion des programmes. L'exposé est volontairement très simple et il est essentiellement inspiré de l'article de K. R. Apt et R. Bol : Logic Programming and Negation : A Survey, publié dans le Journal of Logic Programming, 1994, vol. 19/20, pp. 9-71 et aussi des livres de J. W. Lloyd, A. Nerode et R. A. Shore, U. Nilsson et J. Małuszyński cités en bibliographie.

2. Par $\mid\sim$ on note une relation quelconque d'inférence, soit syntaxique, soit sémantique.

7.3 Résolution SLDNF

Nous avons vu que si nous utilisons la résolution SLDNF, nous avons les règles d'inférence suivantes :

SDLNF - 1 $E \vdash B \circ \theta$ s'il existe un arbre de dérivation SLDNF de $E \cup \{B\}$ avec réponse calculée θ.

SDLNF - 2 $E \vdash \neg B$ si il existe un arbre de dérivation SLDNF d'échecs fini pour $E \cup \{B\}$.

Ces règles ne garantissent pas la justesse du mécanisme d'inférence, ni sa complétude. Pour la justesse considérons dans l'univers des nombres naturels,le programme

$E = \{$`zero(0), positif(X) :-¬zero(X)`$\}$

Nous avons les résultats suivants :

(1) Le but B :←`zero(X)`$\circ\theta$ réussit et la réponse calculée est θ = `(X/0)` d'après (SDLNF - 1).

(2) Donc le but contraire B' :← ¬`zero(X)` échoue de façon finie.

(3) Par conséquent pour le programme $E \cup$ {`positif(X)`} il y a un arbre de dérivation SLDNF d'échecs fini et, d'après (SDLNF - 2), $\forall X$ $E \vdash$ ¬`positif(X)` .

(4) Le but B :←`zero(t)` où `t` est un terme filtré, différent de `0`, échoue de façon finie. Par conséquent, d'après le programme E, nous avons $E \vdash$`positif(t)`, avec t$\neq$ 0.

Nous voyons ainsi que la justesse ne peut pas être assurée. En ce qui concerne la complétude l'examen du programme

$E = \{$`p ← p`$\}$

montre que les questions ¬`p` et `p` ne réussissent pas. De plus n'échouent pas non plus de façon finie. Ce résultat est dû au fait que les deux règles de SLDNF n'evisagent pas une troisième possibilité pour une règle d'inférence, à savoir échouer de façon infinie.

Pour remédier à ces défauts de la résolution SLDNF, nous allons revoir le concept de la résolvante introduite par la définition 6.1.1 du chapitre précédent.

DÉFINITION 7.3.1 (Principe de résolution SLDNF) *Soient un but général* $Q :\leftarrow L_1 \wedge \cdots \wedge L_m$ *avec* L_i *littéral positif ou négatif et un programme général* E *contenant la clause* $C : H \leftarrow A_1 \wedge \cdots \wedge A_n$. *Alors nous pouvons dériver de* Q *et de* C *un nouveau but* Q' *défini*

- *soit par* $Q' :\leftarrow L_1 \wedge \cdots \wedge L_{k-1} \wedge (A_1 \wedge \cdots \wedge A_n) \circ \theta \wedge L_{k+1} \wedge \cdots \wedge L_m$, *si* H *et* L_k *sont unifiables par* θ *avec* θ *UPG pertinent (*[3]*).*

3. Un UPG θ entre deux fbf p et q est *pertinent* s'il n'y a aucune substitution de renommage dans θ.

– *soit par* $Q' :\leftarrow L_1 \wedge \cdots \wedge L_{k-1} \wedge L_{k+1} \wedge \cdots \wedge L_m$, *si on a pour la clause* $C :\leftarrow \neg L_k$. *Dans ce cas on prend* $\theta = \varepsilon$, *où* ε *est la substitution identité.*

Q' *est la* résolvante *de* Q *et* C *et l'on note* $Q \overset{\theta}{\Longrightarrow} Q'$.

Pour éviter toute confusion nous suivrons les notations suivantes :

Lors de la l-ième itération nous passerons du but Q_{l-1} au but $Q_l, l = 1, 2, \ldots$. Pour ce faire nous choisirons une clause $C : H \leftarrow A_1 \wedge \cdots \wedge A_n$ dans le programme E que nous noterons dorénavant $R_l : H_l \leftarrow A_{l1} \wedge \cdots \wedge A_{l_n}$.

D'autre part il est nécessaire de compléter la définition précédente pour obtenir une dérivation SLDNF valable du point de vue logique. Dans la suite nous utiliserons pour une fbf p la notation $V(p)$ pour indiquer l'ensemble de variables de p. Nous avons les définitions suivantes :

DÉFINITION 7.3.2 *Une suite (finie ou infinie) des étapes de résolution*

$$Q_0 \overset{\theta_1}{\Longrightarrow} Q_1 \overset{\theta_2}{\Longrightarrow} \cdots \overset{\theta_n}{\Longrightarrow} Q_n \overset{\theta_{n+1}}{\Longrightarrow} Q_{n+1}$$

est une pseudo-dérivation si

– $V(R_l) \cap V(Q_0) = \emptyset$ *et* $V(R_l) \cap V(R_j) = \emptyset \ \ \forall l = 1, \ldots, n+1$ *et* $\forall j = 1, \ldots, l-1$;
– θ_l *est un UPG pertinent* $\forall l = 1, \ldots, n+1$.

ASCÈSE 7.1 *Soit le programme général* $E = \{a \leftarrow \neg b, \quad b \leftarrow c, \neg a, \quad c \leftarrow a\}$.

(1) Donner la pseudo-dérivation pour le but $B : \leftarrow a$.

(2) Même question si on ajoute la clause $c \leftarrow$.

DÉFINITION 7.3.3 *Une* fôret *est un triplet* $\Delta = (\mathcal{F}, T_0, \sigma)$ *avec*

– $\mathcal{F}$ *ensemble d'arbres. Les sommets des arbres correspondent soit à des buts (questions), soit à des littéraux. Ces sommets peuvent éventuellemnt être étiquetés (Pour les buts les étiquettes peuvent être soit* échoué, *soit* réussi, *soit* enlisé]. Pour les littéraux il y a une étiquette : selectionné.) .
– T_0 est un arbre, élément de $\mathcal{F}$, appelé arbre principal ;
– $\sigma : \mathcal{F} \to \mathcal{F}$ une application qui crée un arbre subsidiaire pour chaque sommet qui correspond à un littéral négatif filtré $\neg A$ et étiqueté. L'arbre créé a comme racine A.

Un chemin *dans* Δ *est une suite de sommets* $A_1, A_2, \ldots$ *tels que pour tout* $i = 1, 2, \ldots$ *on a*

– *soit* A_{i+1} *appartient à l'arbre* $T \in \mathcal{F}$ *et il est obtenu comme successeur du sommet* A_i *qui appartient au même arbre ;*
– *soit* A_{i+1} *est la racine de l'arbre* $\sigma(A_i)$.

Une dérivation SLDNF est une fôret qui au commencement contient seulement l'arbre principal qui, à son tour, au début n'a qu'un sommet, celui qui correspond au but.

DÉFINITION 7.3.4 *Un arbre est appelé*

- réussi, *s'il contient un sommet terminal étiqueté « réussi » ;*
- échoué de façon finie *s'il est fini et si tous ses sommets terminaux sont étiquetés « échoué ».*

DÉFINITION 7.3.5 *Un* arbre pre-SLDNF *par rapport au programme E est une fôret $\mathcal{F}$ dont les sommets sont des buts (questions) ou des littéraux, qui peuvent être éventuellement étiquetés.*

Étant donné un arbre pré-SLDNF $\mathcal{F}$, on peut envisager des extensions de $\mathcal{F}$ pour chaque sommet terminal non étiqueté d'un arbre quelconque $T \in \mathcal{F}$ et qui correspond à un but $B :\leftarrow L_1 \wedge \cdots \wedge L_m$ non vide. On choisit un littéral L_i du but qui est étiqueté comme selectionné. S'il n'y a pas un tel littéral, alors on étiquette un au hasard. Il y a deux possibilités :

- Soit $L_i = A$ est positif.
  - S'il n'y a pas une clause dans E qui peut servir pour établir une résolvante de B en utilisant L_i, alors on étiquette le sommet de B comme échoué.
  - Sinon, pour chaque clause C de E, dont la tête est unifiable avec L_i, on calcule la résolvante B' telle que $B \overset{\theta}{\Longrightarrow} B'$ et on ajoute B' comme successeur de B dans l'arbre T. Notons que le choix de résolvantes doit se faire de sorte que toutes les branches de l'arbre T soient des pseudo-dérivations.
- Soit $L_i = \neg A$ est négatif.
  - A contient des variables, c'est-à-dire il n'est pas filtré. Alors le sommet B sera étiqueté comme enlisé.
  - A est filtré.
    - L'arbre subsidiaire $\sigma(B)$ n'existe pas. Alors un nouvel arbre T' avec un seul sommet qui correspond à A est ajouté à $\mathcal{F}$ et on pose $\sigma(B) = T'$.
    - L'arbre subsidiaire $\sigma(B)$ existe et il est étiqueté comme réussi. Alors B est étiqueté comme échoué.
    - L'arbre subsidiaire $\sigma(B)$ existe et il est étiqueté comme échoué de façon fini. Alors on ajoute comme successeur de B dans T la résolvante $B - \{L_i\}$.

  Notons que tous les buts vides sont étiquetés comme réussis.

  La construction d'un arbre pré-SLDNF se fait de manière inductive :
- Pour tout but B, la forêt qui consiste de l'arbre principal ayant le sommet B est un arbre pré-SLDFN.
- Si $\mathcal{F}$ est un arbre pré-SLDFN, alors toute extension de $\mathcal{F}$ est un arbre pré-SLDFN.

Nous pouvons maintenant donner la définition de l'arbre de la dérivation SLDNF.

DÉFINITION 7.3.6 *Un* arbre SLDNF *est une limite de la suite* $\mathcal{F}_0, \mathcal{F}_1, \ldots$ *où* $\mathcal{F}_0$ *est un arbre initial pré-SLDNF et* $\mathcal{F}_{i+1}$ *est une extension de* F_i.

Un arbre SLDNF pour le but B *est un arbre SLDNF avec comme racine de l'arbre principal* B.

Un arbre SLDNF est réussi *(resp.*échoué de façon finie*) si l'arbre principal est un arbre réussi (resp. échoué de façon finie).*

Un arbre SLDNF est fini *s'il n'a pas des chemins de longueur infinie.*

DÉFINITION 7.3.7 *Une* dérivation SLDNF *pour le but* B *est une branche dans l'arbre principal de l'arbre SLDNF* $\mathcal{F}$ *pour le but* B*, augmentée de l'ensemble de tous les arbres dans* $\mathcal{F}$ *dont les racines sont atteintes par les sommets de cette branche. Elle est appelée réussie si elle se termine avec le but vide.*

Une dérivation SLDNF est appelée finie, si tous les chemins contenus complètement dans cette branche et dans les arbres engendrés par elle, sont finis.

DÉFINITION 7.3.8 *Soit une branche dans l'arbre principal d'un arbre SLDNF* $\mathcal{F}$ *pour un but* B *qui se termine avec le but vide. Soient* $\theta_1, \theta_2, \ldots$ *les substitutions successives tout au long de cette branche. La restriction de* $(\theta_1, \theta_2, \ldots)$ *aux variables du but* B *est la réponse calculée de* B *dans* $\mathcal{F}$.

La dernière question que nous allons examiner concernant la résolution SLDNF est l'arrêt effectif d'un programme. Nous avons

DÉFINITION 7.3.9 *Un programme se termine si tous ses arbres SLDNF pour des buts filtrés sont finis.*

ASCÈSE 7.2 *Établir l'arbre de dérivation SLDNF pour le programme de l'ascese 7.1.*

ASCÈSE 7.3 *Soit le programme* $E = \{q \leftarrow \neg r, \quad r \leftarrow p, \quad r \leftarrow \neg p, \quad p \leftarrow p\}$. *Montrer que pour le but* $B : \leftarrow \neg q$ *il n'y a pas d'arbre SLDNF.*

Une extension vers des buts non filtrés passe par la construction d'une hiérarchie dans les programmes en utilisant les notions des applications de niveau et des programmes stratifiés. Le lecteur intéressé par ces développements pourrait se reporter à la bibliographie cité ci-dessus.

7.4 Complétion des programmes

Nous savons que si on a un programme défini E et un but B à prouver, alors

$E \models B$ ssi $E \vdash B$

Mais si E est un programme général, un résultat analogue n'est pas possible pour les buts qui ne sont pas filtrés. Clark a émis l'idée de remplacer un programme E

par son complété $\mathcal{C}(E)$ qui contient sous forme explicite toutes les connaissances – qui peuvent être implicites – du programme. Par exemple le programme

$$E = \{\texttt{eleve(toto)} \leftarrow \texttt{, eleve(koko)} \leftarrow\}$$

a comme complété

$$\mathcal{C}(E) = \{\, \forall \texttt{X (eleve(X) } \leftrightarrow \texttt{(eleve=toto)} \vee \texttt{(eleve=koko))}\}$$

Ici on fait l'hypothèse que « = » est un nouveau prédicat binaire qui n'apparaît pas dans E et qui est interprété comme identité. Nous avons pour ce symbole les axiomes suivants :

EG1 Pour tout foncteur $f : f(X) = f(Y) \rightarrow X = Y$.

EG2 Pour tous les foncteurs f, g : $\neg(f = g) \rightarrow \neg(f(X) = g(Y))$.

EG3 Pour tout prédicat $p : X = Y \rightarrow (p(X) \rightarrow p(Y))$.

EG4 Pour tout teme t contenant X et différent de X : $\neg(t(X) = X)$.

Étant donné un programme général E, nous avons pour son complété la construction donnée ci-après. Dans cette construction les variables notées X qui apparaissent sont des nouvelles variables qui ne figurent pas dans les clauses du programme E.

(1) Transformer chaque clause $p(t) \leftarrow A$ en la formule $p(X) \leftarrow X = t \wedge A$.

(2) Transformer chaque formule $p(X) \leftarrow F$ obtenue à l'étape précédente en $p(X) \leftarrow \exists Y F$ où Y sont les variables qui existaient dans la clause originale du programme E.

(3) Si $p(X) \leftarrow \exists Y F_1, \ldots, p(X) \leftarrow \exists Y F_k$ sont toutes les formules obtenues lors de l'étape précédente, on remplace ces formules par la formule unique $p(X) \leftarrow F_1 \vee \cdots \vee F_k$. Si $F_1 \vee \cdots \vee F_k$ est vide, on remplace par la clause $\top$ (vrai).

(4) Pour chaque prédicat q qui n'apparaît pas à la tête d'une clause dans E, on ajoute la formule $q(X) \leftarrow \bot$ (faux).

(5) On remplace chaque formule $p(X) \leftarrow F$ par $\forall X\,(p(X) \leftarrow F)$.

(6) Dans chaque formule on remplace "$\leftarrow$" par "$\leftrightarrow$".

ASCÈSE 7.4 *Soit le programme* $E = \{q \leftarrow \neg r, \quad r \leftarrow p, \quad r \leftarrow \neg p, \quad p \leftarrow p\}$. *Montrer que* $\neg q$ *est une conséquence logique de* $\mathcal{C}(E)$.

L'introduction de la complétion n'enlève l'inconsistance pour autant, c'est-à-dire $\mathcal{C}(E)$ peut être un programme inconsistant. Il y a une condition précise qui permet à $\mathcal{C}(E)$ d'être consistant. Avant de présenter cette condition il faut essayer de comprendre les raisons pour lesquelles la complétion peut être inconsistante. En reprenant l'opérateur de la conséquence immédiate $\mathcal{T}_E$, étudié au chapitre 5, et en procédant à son extension dans le cadre des programmes généraux, nous avons que du théorème 5.3.4 reste seulement la troisième partie, à savoir

THÉORÈME 7.4.1 *Soient E un programme général et I une interprétation de Herbrand. I est un modèle de Herbrand si et seulement si $\mathcal{T}_E(I) \subseteq I$.*

Ce que nous avons perdu en passant aux programmes généraux, et qui est important, est la monotonicité et la continuité de l'opérateur de la conséquence immédiate. Nous allons expliciter ce point. Le point fixe de $\mathcal{T}_E$ caractérise dans le cas des modèles généraux non pas le modèle minimal de Herbrand mais un modèle pour la complétion. En effet nous avons le théorème suivant qui est une extension aux programmes généraux du théorème 5.3.5 du point fixe :

THÉORÈME 7.4.2 *Soient E un programme général et I une interprétation de Herbrand. I est un modèle de Herbrand pour $\mathcal{C}(E)$ si et seulement si $\mathcal{T}_E(I) = I$.*

Nous pouvons évidement penser à adapter la construction par la recursion transfinie qui nous a permis d'obtenir le modèle minimal de Herbrand. Mais cette construction était concevable parce que l'opérateur de la conséquence immédiate, pour les programmes définis, était monotone et continu, ce qui n'est pas le cas pour les programmes généraux. Il nous reste donc à voir sous quelles conditions nous pouvons avoir un modèle de Herbrand pour $\mathcal{C}(E)$. Nous avons la définition suivante

DÉFINITION 7.4.1 *Un programme* call-consistant *est un programme dans lequel il n'y a pas d'atome qui dépend négativement de lui-même.*

L'importance des programmes call-consistants vient du théorème suivant.

THÉORÈME 7.4.3 *Si E est un programme call-consistant, alors sa complétion $\mathcal{C}(E)$ a un modèle de Herbrand.*

ASCÈSE 7.5 *Soit le programme général $E = \{q \leftarrow \neg r, \quad r \leftarrow q(X), \quad q(a) \leftarrow\}$ et le but $B : \leftarrow p$. Examiner*

(1) si ce programme a un modèle de Herbrand,

(2) si $E \cup \{B\}$ a un arbre de dérivation SLDNF, et

(3) si le but est une conséquence logique de $\mathcal{C}(E)$.

Nous avons examiné, de manière très simplifiée, les conséquences de l'introduction d'atomes négatifs dans des programmes. Pour résoudre les problèmes posés d'autres types de logiques furent introduites. Le lecteur intéressé pourra poursuivre l'étude de la logique computationnelle en se reportant aux ouvrages et aux revues spécialisés.

7.5 Exercices

EXERCICE 7.1 *Soit le programme*
$E = \{p(X) \leftarrow q(Y), r(Y), \quad q(f(Y)) \leftarrow q(Y), \quad r(g(Y)) \leftarrow\}$.
Trouver deux arbres SLD pour $E \cup \{\leftarrow p(a)\}$ *dont l'un soit infini et l'autre à échecs fini.*

EXERCICE 7.2 *Soit le programme*
$E = \{p(f(X)) \leftarrow p(Y), \quad q(a) \leftarrow p(Y)\}$.
Examiner si $\mathcal{C}(E) \cup \{q(a)\}$ *possède un modèle de Herbrand.*

8

LOGIQUE DE HOARE

De façon très rapide nous pouvons dire que la réalisation d'un logiciel est la concrétisation d'une spécification issue d'un cahier des charges qui, à son tour, est la formalisation d'un ensemble des réquis formulés par un utilisateur.[1] Il est normal, avant de livrer le logiciel de le tester afin de s'assurer qu'il fonctionne correctement et qu'il fournit les résultats attendus. On utilise plusieurs batteries de test et plusieurs types de tests, comme par exemple :

- les *tests fonctionnels* qui permettent de s'assurer que le logiciel fournit les résultats attendus ;
- les *tests structurels* qui permettent de s'assurer que logiciel fonctionne correctement ;
- les *tests d'intégration* qui permettent de s'assurer que l'interaction entre les diffrents modules du programme est correcte.

Néanmoins ces tests ne permettent d'assurer ni la justesse du logiciel, ni sa complétude. C'est-à-dire nous ne sommes pas sûrs que tous les résultats que le programme fournira à l'avenir seront corrects, ni que pour un problème donné qui a une solution, le programme nous donnera la solution. Pour pouvoir satisfaire à ces exigences il faut procéder à une *vérification* c'est-à-dire examiner si le programme réalise correctement les spécifications et seulement celles-ci. Pour ce faire il faut

1. Pour une description détaillée de ces notions, vous êtes priés de vous reporter au cours de méthodologie d'analyse.

effectuer des tests exhaustifs mais, bien évidemment, de tels tests, même pour des programmes modérement complexes, sont impossibles à réaliser.

Une autre méthode pour résoudre ce problème est de considérer qu'un programme est une succession des fbf et par conséquent la vérification devient la démonstration (la preuve) que le programme implique logiquement sa spécification. Nous avons donc besoin de pouvoir transformer un programme informatique en un objet mathématique assimilable par la logique. En fait étant donnée la difficulté de cette démarche, nous allons nous contenter d'établir un formalisme particulier, appelé *méthode de Hoare*, pour les programmes, qui permettra d'appliquer la logique des prédicats sur le comportement de ces programmes. Pour faciliter l'exposé nous allons considérer un langage simplifié de programmation itérative.

8.1 Programmes itératifs

Le langage de programmation avec lequel nous allons travailler est composé
- d'un ensemble infini dénombrable **Var** de variables ;
- d'un ensemble infini dénombrable **Expr** d'expressions ;
- d'un ensemble infini dénombrable **Bexp** d'expressions booleennes ;
- des mots-clé `skip`, `begin`, `end`, `if`, `then`, `else`, `fi`, `while`, `do`, `done` ;
- du signe " ; ".

On construit de façon inductive l'ensemble $\mathcal{P}$ des programmes itératifs comme suit :
- `skip` est un programme de $\mathcal{P}$;
- si `x` $\in$ **Var** et si `e` $\in$ Expr est une expression, alors (`x` $\leftarrow$ `e`)$\in \mathcal{P}$;
- si $P_1, P_2 \in \mathcal{P}$, alors (`begin` P_1, P_2 `end`)$\in \mathcal{P}$;
- si `b`$\in$ **Bexp** et si $P_1, P_2 \in \mathcal{P}$, alors (`if b then` P_1 `else` P_2 `fi`)$\in \mathcal{P}$;
- si `b`$\in$ **Bexp** et si P$\in \mathcal{P}$, alors (`while b do` P `done`)$\in \mathcal{P}$.

Dans la suite on simplifiera en écrivant
- seq(P_1, P_2) à la place de (`begin` P_1, P_2 `end`) ;
- cond(b, P_1, P_2) à la place de (`if b then` P_1 `else` P_2 `fi`) ;
- while(b, P) à la place de (`while b do` P `done`).

ASCÈSE 8.1 *Soit l'algorithme*

```
d\'ebut
  z <-- 0
  tant que y <> 0 faire
    si impair(y)
      alors d\'ebut y <-- y-1; z <-- z+x  fin
      sinon d\'ebut y <-- y div 2; z <-- x+x  fin
    finsi
```

```
  fintantque
fin
```

Écrire le programme correspondant en utilisant le langage simplifié.

Les variables de **Var** prennent leurs valeurs dans un univers du discours $\mathcal{U}$.

On appelle *environnement* toute application ρ : **Var** $\rightarrow \mathcal{U}$. On notera par $\mathfrak{E}$ l'ensemble de tous les environnements.

Étant donné un environnement $\rho \in \mathfrak{E}$, on suppose que

- à chaque variable $x \in$ **Var** est associée la valeur $\rho(x) \in \mathcal{U}$;
- à chaque variable booléenne $b \in$ **Bexp** est associée la valeur $\rho(b)$ qui est égale soit à 1 (vrai), soit à 0 (faux) ;
- à chaque expression $e \in$ **Expr** est associée $\rho(e) \in \mathcal{U}$.

On notera $\mathcal{M}(b) = \{\rho \in \mathfrak{E} \ / \rho(b) = 1\}$ et $\overline{\mathcal{M}(b)} = \{\rho \in \mathfrak{E} \ / \rho(b) = 0\}$ [2]

Étant donné l'environnement $\rho(x)$ et une valeur $v \in \mathcal{U}$, on peut définir l'environnement $\rho(y/x, v)$ comme suit :

$$\forall y \in \textbf{Var} : \rho(y/x, v) = \begin{cases} \rho(y), & \text{si } y \neq x \\ v, & \text{si } y = x \end{cases}, \text{ où } v \in \mathcal{U}$$

Un état s est

- soit un couple (ρ, P), c'est-à-dire le programme P qu'il faut exécuter dans l'environnement ρ,
- soit un environnment ρ qui est obtenu après la fin de l'exécution du programme.

On notera par $\mathcal{S}$ l'ensemble des états.

L'exécution d'un programme fait passer d'un état à un autre, en commençant par l'état initial, en passant par des états intermédiaires et en terminant avec l'état final. A l'état (ρ, P) succédera

- soit l'état (ρ', P') , c'est-à-dire un état intermédiaire,
- soit l'état ρ' qui est l'état terminal.

Dans la suite on examinera l'exécution d'un programme. En particulier on étudiera le passage d'un état à un autre et aussi le passage d'un état initial à un état final.

8.1.1 Forme opérationnelle d'un programme

Le passage d'un état à un autre lors de l'exécution d'un programme P constitue ce qu'on appelle *sémantique opérationnelle* du programme. Elle peut être matérialisée à l'aide d'un graphe $G(P) = (\mathcal{S}, \mathcal{A})$ orienté où $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S} \cup \mathcal{S}$. Les sommets du graphe sont les élements de $\mathcal{S}$ et les arcs indiquent la façon dont on passe d'un

2. Faire l'analogie entre d'une part interprétation et environnement et, d'autre part, modèle et ensemble $\mathcal{M}(b)$.

état à un autre. ont comme étiquette l'opération qui fait passer de l'état du début de l'arc à celui de lafin de l'arc. Ce graphe est défini de la façon suivante :

- $G(\texttt{skip}) = \{[(\rho, \texttt{skip}),\ \rho] \ /\ \rho \in \mathfrak{E}\}$
- $G(\texttt{x} \leftarrow \texttt{e}) = \{[(\rho, \texttt{x} \leftarrow \texttt{e}),\ \rho(\texttt{y}/\texttt{x}, \rho(\texttt{e}))] \ /\ \rho \in \mathfrak{E}\}$
- $G(\mathsf{P}) = \{[(\rho, \mathsf{P}), (\rho', \mathsf{P}')] \ /\ \rho, \rho' \in \mathfrak{E}\ ;\ \mathsf{P}, \mathsf{P}' \in \mathcal{P}\} \cup \{[(\rho, \mathsf{P}), \rho'] \ /\ \rho, \rho' \in \mathfrak{E}\ ;\ \mathsf{P} \in \mathcal{P}\}$
- $G(\mathsf{seq}(\mathsf{P}_1, \mathsf{P}_2)) = \{\left[(\rho', \mathsf{seq}(\mathsf{P}'_1, \mathsf{P}_2)),\ (\rho'', \mathsf{seq}(\mathsf{P}''_1, \mathsf{P}_2)\right] \ /\ [(\rho', \mathsf{P}'_1), (\rho'', \mathsf{P}''_1)] \in G(\mathsf{P})\}$
  $\cup \{\left[(\rho', \mathsf{seq}(\mathsf{P}'_1, \mathsf{P}_2)),\ (\rho'', \mathsf{P}_2)\right] \ /\ [(\rho', \mathsf{P}'_1), \rho''] \in G(\mathsf{P})\} \cup G(\mathsf{P}_2)$
- $G(\mathsf{cond}(\mathsf{b}, \mathsf{P}_1, \mathsf{P}_2)) = G(\mathsf{P}_1) \cup G(\mathsf{P}_2) \cup \{[(\rho, \mathsf{cond}(\mathsf{b}, \mathsf{P}_1, \mathsf{P}_2),\ (\rho, \mathsf{P}_1)] \ /\ \rho(\mathrm{b}) = \mathrm{vrai}\}$
  $\cup \{[(\rho, \mathsf{cond}(\mathsf{b}, \mathsf{P}_1, \mathsf{P}_2), (\rho, \mathsf{P}_2)] \ /\ \rho(\mathrm{b}) = \mathrm{vrai}\}$
- $G(\mathsf{while}(\mathsf{b}, \mathsf{P})) = \{[(\rho, \mathsf{while}(\mathsf{b}, \mathsf{P})),\ (\rho, \mathsf{seq}(\mathsf{P}, \mathsf{while}(\mathsf{b}, \mathsf{P})))] \ /\ \rho(\mathrm{b}) = \mathrm{vrai}\}$
  $\cup \{\left[(\rho', \mathsf{seq}(\mathsf{P}', \mathsf{while}(\mathsf{b}, \mathsf{P}))),\ (\rho'', \mathsf{seq}(\mathsf{P}'', \mathsf{while}(\mathsf{b}, \mathsf{P})))\right] \ /\ [(\rho', \mathsf{P}'),\ (\rho'', \mathsf{P}'')] \in G(\mathsf{P})\}$
  $\cup \{\left[(\rho', \mathsf{seq}(\mathsf{P}', \mathsf{while}(\mathsf{b}, \mathsf{P}))),\ (\rho'', \mathsf{while}(\mathsf{b}, \mathsf{P})))\right] \ /\ [(\rho', \mathsf{P}'),\ \rho''] \in G(\mathsf{P})\}$
  $\cup \{[(\rho, \mathsf{while}(\mathsf{b}, \mathsf{P})), \rho] \ /\ \rho(\mathrm{b}) = \mathrm{faux}\}$

ASCÈSE 8.2 *Soit un sommet $s \in \mathcal{S}$ du graphe $G(P)$ tel que $s \in \mathfrak{E}$. Montrer qu'il n'existe aucun sommet $s' \in \mathcal{S}$ tel que $(s, s') \in \mathcal{A}$.*

Nous avons la proposition suivante :

PROPOSITION 8.1.1 *Si $s = (\rho', P')$, alors il existe un et un seul sommet $s' \in \mathcal{S}$ tel que $(s, s') \in \mathcal{A}$.*

ASCÈSE 8.3 *Soit le programme $P = \{\mathsf{while}\,(x \neq 0,\ x \leftarrow x-1)\}$. Notons par ρ un environnement quelconque et notons pour tout entier $k \in \mathbb{Z}$: $\rho_k = \rho\,(\mathrm{y}/\mathrm{x}, \mathrm{k})$. En commençant par l'environnement ρ_3, montrer que l'unique successeur de (ρ_3, P) est ρ_0.*

Que se passe-t-il si on prend comme environnement initial ρ_{-1} ?

8.1.2 Forme dénotationnelle d'un programme

La *sémantique dénotationnelle* d'un programme P est la fonction partielle $\mathsf{P} : \mathbf{Env} \rightarrow \mathfrak{E}$ qui à l'environnement initial ρ associe, s'il existe, l'environnement final $\mathsf{P}(\rho)$.

Considérons le graphe $G(\mathsf{P}) = (\mathcal{S}, \mathcal{A})$ du programme P et soit $\mathcal{R}(\mathsf{P}) = \left\{(\rho, \rho') \in \mathfrak{E} \times \mathfrak{E} \ /\ ((\rho, \mathsf{P}), \rho') \in \mathcal{A}\right\}$ l'ensemble des couples (ρ, ρ') dont le second membre est la conséquence du premier associé au programme P. En fait cet ensemble est au plus un singleton car pour tout environnement $\rho \in \mathfrak{E}$ il existe au plus un environemment $\rho' \in \mathfrak{E}$ tel que $(\rho, \rho') \in \mathcal{R}(\mathsf{P})$, qui sera noté $\mathsf{P}(\rho)$.

Si on note par $\mathcal{R}$ l'ensemble des applications partielles f de $\mathfrak{E}$ dans $\mathfrak{E}$, on peut définir dans cet ensemble un ordre noté $\sqsubseteq$ et défini comme suit : $\mathsf{f} \sqsubseteq \mathsf{f}'$ si pour tout environnement $\rho \in \mathfrak{E}$ pour lequel $\mathsf{f}(\rho)$ est définie, alors $\mathsf{f}'(\rho)$ est définie aussi et elle est égale à $\mathsf{f}(\rho)$. On peut étendre cette définition en considérant une partie E de $\mathfrak{E}$.

On notera par $f|_E$ la restriction de f à E, c'est-à-dire $f|_E$ est la fonction de E dans $\mathfrak{E}$ qui a la même valeur que f sur E.

La proposition suivante fournit pour la sémantique dénotationnelle la signification de différentes structures du langage simplifié de la programmation.

PROPOSITION 8.1.2 *La sémantique dénotationnelle a les propriétés suivantes :*

(1) $\texttt{skip}$ *est l'identité, c'st-à-dire* $\forall \rho \in \mathfrak{E} : \texttt{skip}(\rho) = \rho$.

(2) $x \leftarrow e(\rho) = \rho\,(y/x, \rho\,(e))$

(3) $seq(P_1, P_2)(\rho) = seq(P_1) \circ seq(P_2)$ *(composition de fonctions partielles)*

(4) $cond(b, P_1, P_2)(\rho) = \begin{cases} P_1(\rho) & si\ \rho(b) = vrai \\ P_2(\rho) & si\ \rho(b) = faux \end{cases}$

(5)
$$while(b, P)(\rho) = \inf\Big\{ f : \mathfrak{E} \to \mathfrak{E} \;/\; \forall f' : \mathfrak{E} \to \mathfrak{E}, f \sqsubseteq f', \; f|_{\overline{\mathcal{M}(b)}} = Id : \overline{\mathcal{M}(b)} \to \overline{\mathcal{M}(b)}, (f \circ P)\,|_{\mathcal{M}(b)} = f|_{\mathcal{M}(b)} \Big\}$$

8.2 La logique de Hoare

Nous avons maintenant à notre disposition l'outillage mathématique indispensable pour la vérification des programmes écrits dans le langage simplifié. En règle générale il est composé d'instructions dont chacune est appliquée à un environnement donné et à la fin de son exécution elle crée un nouvel environnement. Ce triplet prendra le nom d'*assertion*. Plus formellement considérons deux parties Q et R de $\mathfrak{E}$ et un programme P. Une assertion est le triplet $\{Q\}\mathsf{P}\{R\}$. Q est appelée la *précondition* et définit l'ensemble des états constituant le domaine de définition du programme. R est appelée la *postcondition* et elle exprime les propriétés de l'état qu'il soit possible d'être vérifiées à l'issue de l'exécution du programme.

La liaison avec la logique des prédicats est la suivante : L'assertion est valide lorsque pour tout environnement $\rho \in \mathfrak{E}$ tel que $\rho \vDash Q$ et si $\mathsf{P}(\rho) \neq \perp$, alors nécessairement $\mathsf{P}(\rho) = R$. Nous avons noté $\mathsf{P}(\rho)$ l'environnement qu'on obtient après l'exécution du programme P sur l'environnement ρ.

ASCÈSE 8.4 *Soit le programme*

```
debut
    r <-- 0
    tant que n >= (r+1)^2 faire
    debut
        r <-- r+1
    fin tant que
fin
```

qui calcule l'approximation entière de la racine carrée d'un naturel.

Écrire le même programme en utilisant les pré et post conditions.

ASCÈSE 8.5 *Écrire le programme qui calcule la plus grande valeur des variables x et y en utilisant les pré et post conditions. On considère comme univers du discours les nombres naturels.*

Une assertion $\{Q\}\mathsf{P}\{R\}$ est partiellement correcte si pour tout $\rho \in Q$ tel que $\mathsf{P}(\rho)$ est défini, $\mathsf{P}(\rho) \in R$. Une assertion est totalement correcte si elle partiellement correcte et si de plus, $\mathsf{P}(\rho)$ est défini pour tout $\rho \in Q$. Donc dans ce cas l'assertion il est sûr qu'elle termine, tandis que dans le cas d'une assertion partiellement correcte la fin du programme n'est pas assurée.

Assurer la terminaison d'un programme est une tâche difficile qui depasse le cadre du présent cours. Pour nos besoins ici on considérera par la suite que toutes les assertions se terminent, c'est-à-dire qu'elles sont totalement correctes.

8.2.1 Formules de la logique de Hoare

Les formules de la logique de Hoare sont soit des assertions, soit des inclusions $Q \subseteq R$ entre parties de $\mathfrak{E}$.

Cette logique a deux axiomes et possède quatre règles d'inférence.

AXIOMES

A-skip $\{Q\}\texttt{skip}\{Q\}\,;\forall Q \subseteq \mathfrak{E}$

A-affect $\{[\rho : y/x, e]\,Q\}\ x \leftarrow e\ \{Q\}\,;\ \forall Q \subseteq \mathfrak{E}$, où on a noté par $[\rho : y/x, e]\,Q$ l'ensemble des environnements $\{\rho\,/\rho\,(y/x,\rho\,(e)) \in Q\}$. Cet axiome indique que si on considère un environnement pour lequel la variable x fournit la valeur $\rho\,(x) = \rho\,(e)$ dans Q, alors l'affectation de e à x ne modifie pas Q mais toute formule qui contenait la variable x contient maintenant la valeur e à la place de x.

RÈGLES D'INFÉRENCE

RI-seq $\{Q_1\}\mathsf{P}_1\{Q_2\}, \{Q_2\}\mathsf{P}_2\{Q_3\} \longrightarrow \{Q_1\}\mathsf{seq}(\mathsf{P}_1, \mathsf{P}_2)\{Q_3\}$ ce qui signifie que pour démontrer $\{Q_1\}\mathsf{seq}(\mathsf{P}_1, \mathsf{P}_2)\{Q_3\}$ il faut trouver un état intermédiaire Q_2 qui est l'état final de Q_1et l'état initial de Q_3.

RI-cond $\{Q \cap \mathcal{M}(b)\}\mathsf{P}_1\{R\}, \left\{Q \cap \overline{\mathcal{M}(b)}\right\}\mathsf{P}_2\{R\} \longrightarrow \{Q\}\mathsf{cond}(b,\mathsf{P}_1,\mathsf{P}_2)\{R\}$

RI-while $\{Q \cap \mathcal{M}(b)\}\mathsf{P}\{Q\} \longrightarrow \{Q\}\mathsf{while}(b,\mathsf{P})\left\{Q \cap \overline{\mathcal{M}(b)}\right\}$

RI-log $(Q \to Q', \{Q'\}\,\mathsf{P}\,\{R'\}, R' \to R) \longrightarrow \{Q\}\mathsf{P}\{R\}$.

Cette règle, qui est issue de la logique, peut être décomposée en deux règles plus simples, à savoir :

RI-pré $(Q \to Q', \{Q'\}\,\mathsf{P}\,\{R\}) \longrightarrow \{Q\}\mathsf{P}\{R\}$. Il s'agit de la règle logique qui s'applique à la précondition. Elle permet d'obtenir une formule avec l'état attendu Q qui est une restriction de l'état Q' ($Q \subseteq Q'$) avec lequel on a démontré la formule.

RI-post $(\{Q\}\mathsf{P}\{R'\}, R' \to R) \longrightarrow \{Q\}\mathsf{P}\{R\}$. Il s'agit de la règle logique qui s'applique à la postcondition. Elle permet d'obtenir une formule avec l'état attendu R qui est une extension de l'état R' ($R' \subseteq R$) avec lequel on a démontré la formule.

Une *démonstration* dans cette logique est une suite de formules telle que :
- chaque inclusion $Q \subseteq R$ de cette suite doit être justifiée ;
- chaque assertion de cette suite est soit un axiome soit une conséquence d'une règle d'inférence appliquée aux éléments qui précedent cett assertion dans la suite.

On dira qu'une assertion *est prouvée* si elle est la dernière formule d'une démonstration.

ASCÈSE 8.6 *Écrire un programme qui donne à une variable x le plus petit entier pair qui lui est supérieur en utilisant les pré et post conditions. On considère comme univers du discours les nombres naturels.*

La méthode suivie pour la démonstration d'une assertion est fournit par la regle d'inférence *RI-log*. En effet toute assertion $\{Q\}\mathsf{P}\{R\}$ peut être considérée comme la conséquence de cette règle qui, à son tour, peut être appliquée soit parce qu'elle est un axiome, soit parce qu'elle la conséquence d'application de trois premières règles. Donc si on cherche à démontrer une assertion, on commence par la fin : on considère l'assertion à démontrer, on choisit la règle dont cette assertion est la conséquence et on cherche à prouver ses antécédents.

Plus précisement on peut utiliser les deux dernières règles *RI-pré* et *RI-post* qui introduisent dans la démarche de la méthode de Hoare des fragments de preuves purement logiques. La règle *RI-pré*, qui constitue un renforcement de la précondition, peut être utilisée quand on a établi $\{Q'\}\mathsf{P}\{R\}$ et que le but cherché est d'obtenir $\{Q\}\mathsf{P}\{R\}$. Dans ce cas il suffit de démontrer que $Q \to Q'$ qui est un fragment de preuve purement logique. De même la règle *RI-post*, qui constitue un affaiblissement de la postcondition, peut être utilisée quand on a établi que $\{Q\}\mathsf{P}\{R'\}$ et qu'on veut démontrer $\{Q\}\mathsf{P}\{R\}$. Dans ce cas il suffit de démontrer que $R' \to R$ qui est aussi un fragment de preuve purement logique.

L'importance de la démonstration d'une assertion par la logique de Hoare est donnée par la proposition suivante :

PROPOSITION 8.2.1 (COMPATIBILITÉ DE LA LOGIQUE DE HOARE) *Si une assertion est prouvable, alors elle est partiellement correcte.*

D'autre part la logique de Hoare est complète, c'est-à-dire son application permet de trouver toutes les assertions qui sont partiellement correctes.

PROPOSITION 8.2.2 (COMPLÉTUDE DE LA LOGIQUE DE HOARE) *Si une assertion est partiellement correcte, alors elle est prouvable par la logique de Hoare.*

ASCÈSE 8.7 *Soit le programme* `t <-- x; x <-- y; y <-- t` *qui permet à deux variables* `x` *et* `y` *d'échanger leurs valeurs.*

Montrer que ce programme réalise la spécification.

ASCÈSE 8.8 *Prouver le programme de l'ascèse 8.6.*

8.2.2 Plus faible précondition

Considérons l'assertion $\{Q\}\mathsf{P}\{R\}$ qui, comme nous l'avons indiqué, est totalement correcte par hypothèse. Supposons qu'on trouve un programme P' tel que

$$\forall P,Q : \big(\{Q\}\,\mathsf{P}\,\{R\} \longrightarrow \{Q\}\,\mathsf{P}'\,\{R\}\big)$$

On peut donc remplacer P par P′ tout en continuant de satisfaire à la spécification. On dit que P′ constitue un raffinement de P que l'on note $\mathsf{P}\sqsubseteq\mathsf{P}'$. Cette relation est réflexive, transitive et monotone. La transitivité permet de raffiner un programme par étapes successives. La monotonie permet de raffiner un programme par parties.

Supposons maintenant que pour un programme donné P et une postcondition fixé R on a un opérateur, appellé **wp** – **w**eakest **p**ostcondition – qui nous fournit la plus faible précondition $Q = \mathbf{wp}(\mathsf{P},R)$ telle que l'assertion $\{Q\}\mathsf{P}\{R\}$ se termine. Cet opérateur a été introduit par Dijkstra.

Ainsi le raffinement se définit à l'aide de **wp** comme suit :

$$\mathsf{P} \sqsubseteq \mathsf{P}' \longleftrightarrow \big(\forall Q \ : \mathbf{wp}\,(\mathsf{P},\mathsf{Q}) \rightarrow \mathbf{wp}\,\big(\mathsf{P}',\mathsf{Q}\big)\big)$$

8.2.3 Invariant

Une autre notion qui est utile pour la vérification est la notion de l'*invariant* dans un programme itératif, c'est-à-dire de la condition qui reste vraie après chaque itération. Par exemple si on prend l'itération while(b, P) d'après la règle RI-while Q reste vrai à chaque étape de l'itération. Q est donc un invariant pour while.

À la notion d'invariant est associée celle de *variant* qui est une fonction définie sur un domaine (habituellement sur les nombres naturels) et qui permet de caractériser les propriétés d'une itération. Par exemple dans le cas d'une boucle `for` fournit la borne supérieure du nombre d'itérations. Un autre exemple est donné par le programme while(`x <> 0, x ← x - 2`). Ce programme ne se termine pas si x est initialement impair. On peut détecter cette situation en utilisant le variant v = x. D'après la condition nous avons au début que $\mathtt{x} \in \mathbb{N} = \{1,2,\ldots\}$. Après la première itération on $\mathtt{v} \in \{-1,0,1,2,\ldots\}$ et donc v n'est plus un naturel.

ASCÈSE 8.9 *Pour le programme de l'ascèse 8.4 trouver l'invariant.*

8.3 Exercices

EXERCICE 8.1 *En s'inspirant de l'ascèse 8.7, examiner la possibilit d'écrire un programme qui échange les valeurs de deux variables en utilisant deux instructions.*

EXERCICE 8.2 *Soit le programme*

```
debut
    x <-- 1
    y <-- b
    tant que y <> 0 faire
    debut
        x <-- x*a
        y <-- y-1
    fin tant que
fin
```

(1) Écrire le triplet de Hoare correspondant

(2) Démontrer que le triplet de Hoare est vrai.

9

LOGIQUE FLOUE

La logique floue est issue de la théorie des ensembles flous, dont la forme définitive est due à L. Zadeh (1965). Les ensembles flous sont définis par rapport à un univers du discours $\mathcal{U}$. Un ensemble flou $A \subset \mathcal{U}$ est déterminé par sa fonction d'appartenance $\mu_A : \mathcal{U} \rightarrow [0,1]$ qui exprime le degré d'appartenance de chaque élément de $\mathcal{U}$ à A. Rappelons qu'un ensemble classique A est aussi déterminé par sa fonction d'appartenance (fonction indicatrice) $\chi_A : \mathcal{U} \rightarrow \{0,1\}$. Donc on peut considérer les ensembles flous comme une extension des ensembles déterministes.

La logique floue a été développée selon deux axes distincts :

- Logique floue au sens large.- Il fut historiquement le premier axe développé. Il sert essentiellement pour des applications dans divers domaines, comme par exemple, la commande, la reconnaissance de formes, le traitement du langage naturel, etc.
- Logique floue au sens strict.- Axe plus récent. Il s'agit d'une extension de la logique symbolique dans le domaine de la notion comparative de vérité dont les valeurs se trouvent dans l'intervalle $[0,1]$. Elle est donc une branche de la logique multi-valuée.

La logique floue avec ce deux axes, permet d'établir des valeurs de vérité à

des constructions vagues comme la suivante :

L'hôtel et proche de la gare.

Le bistrot est très proche de la gare.

Alors l'hôtel n'est pas très éloigné du bistrot.

Avec la logique floue nous rentrons dans une de grandes controverses de la science. En effet le logicien allemand Frege soutenait que la présence des expressions vagues dans une phrase est le signe d'une incohérence intrinsèque du discours. À l'opposé, Wittgenstein, le logicien du cercle de Vienne, affirmait que l'imprécision est une composante essentielle du discours. À notre avis cette controverse n'a pas lieu d'être. Car s'il est vrai que l'opinion de Frege est juste concernant le langage logico-mathématique – et scientifique, en général – il est par ailleurs impossible, en intelligence artificielle, de faire un traitement des connaissances sans tenir compte des expressions vagues du discours et donc d'utiliser la logique floue pour s'en sortir.

La logique floue au sens strict a des similitudes dans sa démarche avec la logique classique. Néanmoins on peut considérer qu'elle est de nature différente. Ainsi le propriétés du tiers exclu et de la conradiction ne s'appliquent pas. Elle a aussi plusieurs opérateurs d'inclusion.

Ce chapitre présente la logique floue au sens strict. Les cours des algorithmes génétiques, des réseaux de neurones et des systèmes experts utilisent, à des degrés divers, cette logique.

9.1 Système logique gradué

Nous avons vu que l'appartenance d'un élément a à un ensemble flou E est donné par le degré d'appartenance $\mu_E(a) \in [0,1]$. Si on cherche à établir un système logique sur les ensembles flous, il est normal de prendre en compte le degré d'appartenance $\mu(a)$ pour chaque élément a et de considérer qu'il constitue en réalité la valeur de vérité de a. Nous allons, pour commencer, construire un *système logique gradué* qui est un cas particulier des systèmes logiques flous. La technique pour la construction d'un tel système logique est la suivante. On définit un ensemble $B \subset [0,1]$ des valeurs de vérité. On suppose que pour chaque valeur de vérité $a \in B$, correspond une constante $\mathfrak{a}$ qui sera appelé *constante logique*. On notera Ψ l'ensemble des constantes logiques.

On muni B d'une structure de treillis. On rappelle ici la définition du treillis.

DÉFINITION 9.1.1 *Un ensemble B est un treillis s'il est ordonné et si pour chaque couple d'éléments de B leur plus petite borne supérieure (sup) et leur plus grande borne inférieure (inf) existent et elles sont dans B.*

D'habitude on définit dans le treillis deux opérations, $\vee$ et $\wedge$, et on a

$$\sup(a,b) = a \vee b, \ \ \inf(a,b) = a \wedge b$$

Si on note par $\leq$ la relation d'ordre dans B, on a

$a \leq b$ ssi $a \vee b = b$, ou, de façon équivalente, $a \wedge b = a$

Le lemme suivant fournit les propriétés d'un treillis.

LEMME 9.1.1 *Soit B un treillis. Nous avons les propriétés suivantes :*

(1) Associativité : $a \vee (b \vee c) = (a \vee b) \vee c, \ a \wedge (b \wedge c) = (a \wedge b) \vee c$

(2) Commutativité : $a \vee b = b \vee a, \ a \wedge b = b \wedge a$

(3) Absorption : $a \vee (a \wedge b) = a, \ \ a \wedge (a \vee b) = a$

(4) Idempotence : $a \vee a = a, \ a \wedge a = a$

Des treillis importants sont les treillis distributifs et des treillis complémentés.

DÉFINITION 9.1.2 *Un treillis B est distributif si les deux identités suivantes sont satisfaites :*

- $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
- $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

DÉFINITION 9.1.3 *Un treillis B est complémenté s'il possède un opérateur unaire $\neg$ et un plus petit élément noté 0 (ou $\bot$) avec les propriétés :*

- *Double négation :* $\neg\neg a = a$
- *Lois de de Morgan :* $\neg(a \vee b) = \neg a \wedge \neg b, \ \neg(a \wedge b) = \neg a \vee \neg b$
- *Loi de la contradiction :* $a \wedge \neg a = 0$

Pour un treillis complémenté, on pose $1 = \neg 0$ ou $\top = \neg 0$. On peut montrer que 1 est plus grand élément de B

Nous pouvons sur ce treillis définir un système logique gradué comme suit

$$\Sigma_G = (B, \Psi, L, 0, 1) = (B, \Psi, L, \bot, \top)$$

avec

- $B \subset [0,1]$ treillis complémeté qui détermine le nombre de valeurs de vérité de la logique ;
- Ψ est un ensemble des constantes logiques $\mathfrak{a}$ dont l'évaluation logique (la valeur de vérité) est $a \in B$;
- $0, 1$ (resp. $\bot, \top$) représente la plus petite et la plus grande valeur de B ;
- $L = \{\vee, \wedge, \otimes, \rightarrow, \longleftrightarrow\}$, où on a noté par $\otimes$ l'opération de multiplication définie comme suit

$$a \otimes b = 0 \vee (a + b - 1); \ \ a, b \in B$$

et par $\rightarrow$ l'opération de la résiduation définie ci-après

$$a \rightarrow b = 1 \wedge (1 - a + b); \;\; a, b \in B \; ([1])$$

Pour faire le parallèle avec la logique des propositions, présentée au chapitre 3, nous nous plaçons dans l'univers du discours $\mathcal{U}$. Une constante $a \in \mathcal{U}$ a, pour un ensemble flou $E \subset \mathcal{U}$, un degré d'appartenance $\mu_E(a) \in B$. On peut donc assimiler la constante a dans E avec la valeur de la fonction indicatrice μ_E et noter, par abus de langage, $a \in B$. Dans la mesure où B est un sous-ensemble de $[0,1]$, il y a seulement certaines valeurs a de cet intervalle qui sont dans B, c'est-à-dire qui peuvent jouer le rôle des valeurs de vérité. Ces valeurs doivent correspondre à des constantes effectives du système logique. L'ensemble de ces constantes effectives est appellé ensemble des constantes logiques est il est noté par Ψ. Nous pouvons faire une remarque analogue concernant la logique des prédicats.

Si on veut se rapprocher de la logique définie sur des ensembles classiques, alors il faut prendre $B = \{0,1\}$ ce qui réduit B à la plus petite et à la plus grande valeur de $[0,1]$. Dans ce cas nous avons pour le système logique classique le schéma suivant

$$\Sigma = (B, \Xi, L, 0, 1) = (B, \Xi, L, \bot, \top)$$

avec $L = \{\vee, \wedge, \rightarrow, \longleftrightarrow\}$ et où on a noté par $\rightarrow$ l'opération de l'implication logique. Ξ représente ici l'ensemble des constantes de la logique. Remarquons encore une fois que la notation $a \in B$ signifie que $\chi_E(a) \in B$.

ASCÈSE 9.1 *Montrer que*

$$\forall a, b, c \in B : a \otimes b \leq c \;\; ssi \;\; a \leq b \rightarrow c$$

ASCÈSE 9.2 *Montrer que le système logique classique Σ devient un système gradué Σ_G si on définit la résiduation $a \rightarrow b = \neg a \vee b$.*

LEMME 9.1.2 *Soient Σ_G un système logique gradué et $a, b, c \in B$. Nous avons les propriétés suivantes :*

(1) $a \otimes b \leq a, \;\; a \otimes b \leq b, a \otimes b \leq a \wedge b$

(2) $b \leq a \rightarrow b$

(3) $a \otimes (a \rightarrow b) \leq b, \;\; b \leq a \rightarrow (a \otimes b)$

(4) Isotonicité du 2e argument : Si $a \leq b$, alors $c \rightarrow a \leq c \rightarrow b$

(5) Antitonicité du premier argument : Si $a \leq b$, alors $b \rightarrow c \leq a \rightarrow c$

(6) $a \otimes (a \rightarrow 0) = 0$

(7) $a \rightarrow (b \rightarrow c) = (a \otimes b) \rightarrow c$

(8) $a \leq b$ si et seulement si $a \rightarrow b = 1 \;, \;\; a = 1 \rightarrow a$

(9) $(a \vee b) \otimes c = (a \otimes b) \vee (b \otimes c)$

(10) $a \vee b \leq ((a \to b) \to b) \wedge ((b \to a) \to a)$

ASCÈSE 9.3 *Soit Σ_G un système logique gradué. Si on a*

$$\forall a \in B : 1 = a \vee (a \to 0)$$

alors Σ_G est un treillis booléen et $\otimes = \wedge$.

Le système logique graduée possède un certain nombre de propriétés dont on trouve l'équivalent en logique classique. Ainsi nous avons :

DÉFINITION 9.1.4 *Soient Σ_G un système logique gradué et $a, b \in B$. Nous avons les propriétés suivantes :*

(1) Négation : $\neg a = a \to 0$

(2) Birésiduation : $a \longleftrightarrow b = (a \to b) \wedge (b \to a)$

(3) Addition : $a \oplus b = \neg(\neg a \otimes \neg b) = 1 \wedge (a + b)$

(4) Multiplication n fois : $a^n = \underbrace{a \otimes \cdots \otimes a}_{n \text{ fois}}$

(5) Addition n fois : $na = \underbrace{a \oplus \cdots \oplus a}_{n \text{ fois}}$

LEMME 9.1.3 *La négation définie par $\neg a = a \to 0$ implique que $\neg a = 1 - a$.*

D'après le lemme précédent la négation est involutive, c'est-à-dire $\neg\neg a = a$.

ASCÈSE 9.4 *Démontrer le lemme précédent.*

ASCÈSE 9.5 *La négation inverse l'ordre :*

$$a \leq b \quad ssi \quad \neg b \leq \neg a$$

DÉFINITION 9.1.5 *Un système logique gradué est linéairement ordonné si $\forall a, b \in B$ on a soit $a \leq b$, soit $b \leq a$.*

Il est complet si le treillis B est complet.

LEMME 9.1.4 *Soient $\pm_G$ un système logique gradue flou complet et $a, b, a_i, b_i, x \in B$ pour $i \in I$. Nous avons les propriétés suivantes :*

(1) $a \to b = \bigvee \{x / a \otimes x \leq b\}$

(2) $a \otimes b = \bigwedge \{x / a \leq b \to x\}$

(3) $\left(\bigvee_{i \in I} a_i\right) \otimes b = \bigvee_{i \in I} (a_i \otimes b)$

(4) $\bigwedge_{i \in I} (a \to b_i) = a \to \left(\bigwedge_{i \in I} b_i\right)$

$$(5)\ \bigwedge_{i\in I}(a_i \to b) = \left(\bigvee_{i\in I} a_i\right) \to b$$

$$(6)\ \bigvee_{i\in I}(a \to b_i) = a \to \left(\bigvee_{i\in I} b_i\right)$$

$$(7)\ \bigvee_{i\in I}(a_i \to b) = \left(\bigwedge_{i\in I} a_i\right) \to b$$

9.1.1 Évaluation syntaxique - Démonstration

Considérons maintenat un langage formel $\mathcal{L}$ sur le système logique gradué Σ_G et soit $\mathbb{F}$ un ensemble des formules bien formées sur $\mathcal{L}$. On suppose que pour une valeur de vérité $a \in B$ nous avons dans le langage $\mathcal{L}$ une constante logique $\mathfrak{a}$ correspondante. Supposons aussi que dans $\mathcal{L}$ on trouve les symboles spécifiques $\bot, \top$.

DÉFINITION 9.1.6 *Une formule évaluée est le couple*

$$a/A$$

où $A \in \mathbb{F}$ et $a \in B$ est la valeur de vérité de son évaluation syntaxique.

Soit une suite des fbf dans $\mathbb{F}$ composée des fbf $A_1, \ldots, A_n$. Supposons que l'évaluation syntaxique de $A_1, \ldots, A_n$ est $a_1, \ldots, a_n$. On note par $r_{syn}(a_1, \ldots, a_n)$ une opération syntaxique sur la suite et $r_{ev}(A_1, \ldots, A_n)$ l'évaluation de l'opération syntaxique précédente. On admet qu'elle est une opération d'arité n sémi-continue inférieurement sur B, c'et-à-dire qu'elle préserve les valeurs maximales de tous les éléments. Nous pouvons maintenat définir pour la *règle d'inférence d'un système logique gradué (ISLG)* le schéma suivant :

$$\text{(ISLG)} \qquad \frac{a_1/A_1 \cdots a_n/A_n}{r_{ev}(a_1, \ldots, a_n)/r_{syn}(A_1, \ldots, A_n)}$$

Nous pouvons maintenant considérer que la fbf A appartient à un sous-ensemble flou $E \subset \mathbb{F}$. La démonstration de A est une démonstration au sens classique du terme couplée avec une évaluation du degré d'appartenance du résultat dans $\mathbb{F}$ comme il est précisé dans la définition suivante.

DÉFINITION 9.1.7 *Une démonstration évaluée d'une fbf A à partir de l'ensemble flou $E \subset \mathbb{F}$ est une suite finie de formules évaluées*

$$a_1/A_1 \cdots a_n/A_n$$

telle que $A = A_n$ et pour chaque $i \leq n$

- *soit $a_i/A_i = \mu_E(A_i)/A_i$,*
- *soit il existe une règle d'inférence telle que*

$$a_i/A_i = r_{ev}(a_{i_1}, \ldots, a_{i_n})/r_{syn}(A_{i_1}, \ldots, A_{i_n})\ ;\ \text{avec } i_1, \ldots, i_n < i$$

Pour l'évaluation de la fbf A on utilise la valeur a_n et on note $H(a_1/A_1 \cdots a_n/A_n) = a_n$ cette évaluation.

Comme exemple de la règle d'inférence nous pouvons envisager d'avoir $H(a_1/A_1 \cdots a_n/A_n) = a_1/A_1 \otimes \cdots \otimes a_n/A$.

ASCÈSE 9.6 *Soient les formules évaluées*

$$1/\forall X \forall Y \left(A(X \wedge A(Y) \rightarrow A(X,Y)\right), \;\; 0.9/A(c)$$

Calculer $H(A(c,c))$.

Un axiome dans un système logique gradué n'est pas juste une fbf valide toujours et sans démonstration mais, au contraire, un ensemble des formules évaluées. Comme ces évaluations sont de degrés d'appartenance dans un ensemble flou, on peut considérer que les axiomes sont des ensembles flous des formules. On distingue deux sortes d'axiomes : les axiomes logiques classiques qui forment l'ensemble $\mathcal{A}$ et les axiomes spécifiques du calcul flou qui forment l'ensemble A_{Fl}. Bien évidement, en fonction de ce qui vient d'être dit, $\mathcal{A}$ et $\mathcal{A}_{Fl}$ sont des sous-ensembles flous de $\mathbb{F}$. Notons aussi par R l'ensemble des règles d'inférence utilisées en logique floue. R contient donc la règle ISLG et, éventuellement, d'autres règles d'inférence.

DÉFINITION 9.1.8 *Une théorie floue $\mathcal{T}(\mathcal{L})$ sur le langage $\mathcal{L}$ est le triplet*

$$\mathcal{T}_{Fl}(\mathcal{L}) = (\mathcal{A}, \mathcal{A}_{Fl}, R)$$

On notera par $\mathcal{T}$ la théorie $\mathcal{T}_{Fl}(\mathcal{L})$ s'il n'y a pas risque d'ambigüité.

Si $w = a_1/A_1 \cdots a_n/A_n$ est une démonstration dans $\mathcal{T}$, alors $H_{\mathcal{T}}(w)$ est sa valeur de vérité. La définition suivante précise la notion du théorème dans une logique floue.

DÉFINITION 9.1.9 *$\mathbb{A} \in \mathbb{F}$ est un théorème de degré $a \in B$ dans la théorie floue $\mathcal{T}$, que l'on note $\mathcal{T} \vdash_a A$, si*

$$a = \bigvee \left\{ H(w) \,/ w \text{ est une démonstration de } A \text{ dans } \mathcal{A} \cup \mathcal{A}_{Fl} \right\}$$

Remarquons que, contrairement à la théorie classique où l'élaboartion d'une seule démonstration suffit pour établir que A est un théorème, en logique floue il faut trouver le maximum des degrés a pour le théorème, donc faire autant de démonstrations que nécessaire.

ASCÈSE 9.7 *Montrer que le principe d'induction n'est pas valide en logique floue.*

9.1.2 Interprétation sémantique - Modèles

Nous donnons la définition de l'interprétation floue et du modèle flou.

DÉFINITION 9.1.10 *Une interprétation floue I_{Fl} des fbf dans $\mathbb{F}$ est une fonction $I_{Fl} : \mathbb{F} \to \mathcal{L}$ avec la propriété suivante : soit $a \in B$ une valeur de vérité et supposons qu'il existe dans $\mathbb{F}$ une constante $\mathfrak{a}$ telle que $\mu_{\mathbb{F}}(\mathfrak{a}) = a$, alors $I_{Fl}(\mathfrak{a}) = a$.*

DÉFINITION 9.1.11 *L'interprétation floue I_{Fl} est un modèle pour la théorie floue $\mathcal{T}$, que l'on note $I_{Fl} \models \mathcal{T}$, si $\mathcal{A}_{Fl}(A) \leq I_{Fl}(A)$ pour toute formule $A \in \mathbb{F}$.*

Il faut bien se garder d'interpéter $\mathcal{A}_{Fl}(A)$ comm une valeur de vérité de A mais comme une restriction (contrainte) sur les valeurs possibles de vérité de A.

DÉFINITION 9.1.12 *Soit $\mathcal{T}$ une théorie f loue. On dit que A est $a-$valide (valide avec un degré a) dans la théorie floue $\mathcal{T}$, et l'on note $\mathcal{T} \models_a A$, si*

$$a = \bigwedge_{I_{\mathcal{F}\updownarrow} \models \mathcal{T}} I_{\mathcal{F}\updownarrow}(A)$$

A est une a-tautologie (tautologie avec degré a), et l'on note $\models_a A$, si

$$a = \bigwedge_{I_{\mathcal{F}\updownarrow}\ interp.floue} I_{\mathcal{F}\updownarrow}(A)$$

Si $a = 1$, alors A est une tautologie et on écrit $\models A$.

ASCÈSE 9.8 *Montrer que $I_{\mathcal{F}\updownarrow}(A \vee \neg A) \geq 0.5$.*

9.1.3 Équivalence entre modèles et théorie de démonstration

Comme dans la logique des prédicats, nous alors établir les conditions pour l'équivalence entre modèles et théorèmes.

DÉFINITION 9.1.13 *Un système logique gradué est adéquat si, pour théorie floue $\mathcal{T}$ et toute fbf $A \in \mathbb{F}$ nous avons*

$$\mathcal{T} \vdash_a A \ \ et \ \ \mathcal{T} \models_b A \ \ implique \ a \leq b$$

Remarquons que nous avons toujours $\mathcal{T} \vdash_a A$ et $\mathcal{T} \models_b A$ pour un $a, b \in B$.

DÉFINITION 9.1.14 *Un système logique gradué est complet si, pour théorie floue $\mathcal{T}$ et toute fbf $A \in \mathbb{F}$ nous avons*

$$\mathcal{T} \vdash_a A \ \ ssi \ \ \mathcal{T} \models_a A$$

THÉORÈME 9.1.1 *Un système logique gradué est complet si, pour tout sous-ensemble B' de B, les relations suivantes*

$$\bigvee_{b_i \in B'} (a \to b_i) = a \to \left(\bigvee_{b_i \in B'} b_i \right)$$

$$\bigvee_{a_i \in B'} (a_i \to b) = \left(\bigwedge_{a_i \in B'} a_i \right) \to b$$

$$\bigwedge_{b_i \in B'} (a \to b_i) = a \to \left(\bigwedge_{b_i \in B'} b_i \right)$$

$$\bigwedge_{a_i \in B'} (a_i \to b) = \left(\bigvee_{a_i \in B'} a_i \right) \to b$$

sont vérifiées.

9.2 Logique floue des prédicats

On généralisera le système logique gradué à un système logique flou. Il suffit pour cela de considérer que l'ensemble B est l'intervalle entier $[0,1]$. Les définitions, lemmes et théorèmes déjà établis pour les sstèmes gradués restent valebles pour les systèmes flous.

Plus précisement nous allons travailler avec un système logique flou des prédicats. Nous construisons d'abord un alphabet qui est composé par

- un ensemble au plus dénombrable **V** des variables $X, Y, \ldots$;
- un ensemble au plus dénombrable Ξ des constantes $\mathbf{u}, \mathbf{v}, \ldots$ [2] ;
- un ensemble au plus dénombrable F des foncteurs f, g,au plus dénombrable ;
- un ensemble au plus dénombrable **P** des prédicats p ;q,au plus dénombrable ;
- un ensemble $\Psi = \{a / a \in B\}$ des constantes logiques ;
- un ensemble des connecteurs flous $L = \{\vee, \wedge, \to, \longleftrightarrow\}$
- les quantificateurs $\forall, \exists$
- un connecteur d'implication binaire logique $\Rightarrow$
- des symboles extralogiques comme des parenthèses, des crochets, ...

Nous allons définir les termes et les formules bien formées (fbf) de manière analogue que dans la logique des prédicats.

DÉFINITION 9.2.1 *Un terme est défini de façon récursive comme suit :*

2. Notez bien que pour les constantes du calcul flou des prédicats, nous utilisons ici des caractères gras.

– *une constante est un terme ;*
– *une variable est un terme ;*
– *si f est un foncteur d'arité n et $t_1, \ldots, t_n$ sont des termes, alors $f(t_1, \ldots, t_n)$ est un terme.*

Tout autre terme est obtenu par application des règles précédentes un nombre fini de fois.

On notera par $\mathbb{T}$ l'ensemble des termes.

DÉFINITION 9.2.2 *L'ensemble des formules bien formées $\mathbb{F}$ est le plus petit ensemble tel que*

– *une constante logique $\mathfrak{a}$ est une fbf ;*
– *si p est prédicat d'arité n et si $t_1, \ldots, t_n$ sont des termes, alors $p(t_1, \ldots, t_n)$ est une fbf ;*
– *si $A, B \in \mathbb{F}$, alors $A \Rightarrow B \in \mathbb{F}$;*
– *si $X \in \mathbf{V}$ et $A \in \mathbb{F}$, alors $(\forall X)A \in \mathbb{F}$*

Toute autre fbf est obtenue par application des règles précédentes un nombre fini de fois.

Notons les notions de substitution, de portée des quantificateurs des formules filtrées ou libres sont identiques à celles de la logique des prédicats.

Nous avons les propriétés suivantes :

(1) Négation : $\neg A \equiv A \Rightarrow \bot$

(2) Disjonction : $A \vee B \equiv (B \Rightarrow A) \Rightarrow A$

(3) Conjonction : $A \wedge B \equiv \neg((B \Rightarrow A) \Rightarrow \neg B)$

(4) Conjonction floue (de Łukasiewicz) : $A \& B \equiv \neg(A \Rightarrow \neg B)$

(5) Disjonction floue (de Łukasiewicz) : $A \bigtriangledown B \equiv \neg(\neg A \Rightarrow \neg B)$

(6) Éqivalence : $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$

(7) Quantificateur existentiel : $(\exists X)A \equiv \neg(\forall X)\neg A$

9.2.1 Évaluation syntaxique

Nous avons les trois règles d'inférence suivantes :

R1 Modus ponens

$$r_{MP} \ : \ \frac{a/A, \ \ b/A \Rightarrow B}{a \otimes b/B}$$

R2 Généralisation

$$r_G \ : \ \frac{a/A}{a/(\forall X)A}$$

R3 Introduction de la constante logique

$$r_{CL} \ : \ \frac{a/A}{a \to a/\mathfrak{a} \Rightarrow A}$$

Remarquons que opération d'évaluation de la dernière règle est $H\left(r_{CL}\left(X\right)\right) = a \to X$. Ainsi, étant donnée une formule évaluée a/A la règle r_{CL} représente l'évaluation a. Donc on peut identifier la formule évaluée avec la formule $\mathfrak{a} \Rightarrow A$.

Le calcul flou des prédicats possède l'ensemble $\mathcal{A}$ des axiomes suivants :

A1 Introduction de l'implication binaire

$$A \Rightarrow (B \Rightarrow A)$$

A2 Transitivité de l'implication binaire

$$(A \Rightarrow B) \Rightarrow ((B \Rightarrow C) \Rightarrow (A \Rightarrow C))$$

A3 Réalisation de la contradiction binaire

$$(\neg B \Rightarrow \neg A) \Rightarrow (A \Rightarrow B)$$

A4 Inversion de l'implication binaire

$$((A \Rightarrow B) \Rightarrow B) \Rightarrow ((B \Rightarrow A) \Rightarrow A)$$

A5 Exemplification (instanciation) universelle

$$(\forall X) A \Rightarrow A_X\left[t\right] \ ; \ \text{pour terme substituable } t$$

A6 Généralisation universelle

$$(\forall X)\left(A \Rightarrow B\right) \Rightarrow \left(A \Rightarrow B\left(\forall X\right)\right) \ ; \ \text{si } X \text{ n'est pas libre dans } A$$

A7 Implication binaire des constantes logiques

$$(\mathfrak{a} \Rightarrow \mathfrak{b}) \Longleftrightarrow \left(\overline{\mathfrak{a} \to \mathfrak{b}}\right)$$

où $\overline{\mathfrak{a} \to \mathfrak{b}}$ indique la constante logique qui correspond à la valeur de vérité $a \to b$, quand a et b sont donnés.

L'ensemble des axiomes logiques du calcul flou des prédicats est donné par

$\mathbf{A}_{Fl}$1

$$\mathcal{A}_{Fl}\left(A\right) = \begin{cases} a \in B, & \text{si } A \equiv \mathfrak{a} \\ 1, & \text{si } A \text{ a une des formes (A1)-(A6)} \\ 0, & \text{sinon} \end{cases}$$

9.2.2 Interprétation sémantique

Pour effectuer une interprétation sémantique floue I_{Fl} que l'on notera ici, sans risque de confusion, I, il faut déterminer un univers du discours U dans lequel on doit trouver

- un ensemble D non vide, qu'on appelera domaine de l'interprétation ;
- pour chaque constante $\mathbf{u}$, une constante $\mathbf{u}_I = u \in D$ [3]
- pour chaque foncteur f d'arité n une fonction ordinaire f_I de même arité et avec des valeurs dans D ;
- pour chaque prédicat p d'arité n, une relations floue p_I de même arité et avec valeurs dans B.

Comme précédement une interprétation floue I_{Fl} des fbf dans $\mathbb{F}$ est une fonction $I_{Fl} : \mathbb{F} \to \mathcal{L}$ définie comme suit :

(i) Interprétation des termes clos :

- $I(\mathbf{u}) = u$; $\mathbf{u} \in \Xi$, $u \in D$
- $I(u) = u$; $u \in D$
- $I(f(t_1, \ldots, t_n)) = f_I(I(t_1), \ldots, I(t_n)) \in D$

(ii) Interprétation des fbf closes :

- $I(\mathfrak{a}) = a$ pour tout $a \in B$;
- $I(p(t_1, \ldots, t_n)) = p_I(I(t_1), \ldots, I(t_n)) \in B$;
- $I(A \Rightarrow B) = I(A) \Rightarrow I(B)$
- $I((\forall X)A) = \bigwedge_{u \in D} I(A_X[\mathbf{u}])$
- $I(A \wedge B) = I(A) \wedge I(B)$
- $I(A \& B) = I(A) \otimes I(B)$
- $I(A \vee B) = I(A) \vee I(B)$
- $I(A \veebar B) = I(A) \oplus I(B)$
- $I(A \Longleftrightarrow B) = I(A) \longleftrightarrow I(B)$
- $I((\exists X)A) = \bigvee_{u \in D} I(A_X[\mathbf{u}])$

Nous avons la définition suivante :

DÉFINITION 9.2.3 *L'interprétation floue I est un modèle pour D, que l'on note $I \models D$, si $\mathcal{A}_{Fl}(A) \leq I(A)$ pour toute formule A du domaine D.*

Le théorème suivant fournit une méthode pour vérifier si une fbf est une tautologie en logique floue.

THÉORÈME 9.2.1 *Soient A, B deux fbf. Nous avons*

- $\models A \Rightarrow B$ *ssi* $I(A) \leq I(B)$ *pour toute interprétation I du domaine D.*
- $\models A \Longleftrightarrow B$ *ssi* $I(A) = I(B)$ *pour toute interprétation I du domaine D.*

9.3 Équivalence entre modèles et théorèmes

Nous avons les théorèmes suivants :

3. Notez bien que les éléments du domaine D sont désignés avec des caractères normaux, c'est-à-dire non gras.

THÉORÈME 9.3.1 (DE LA VALIDITÉ) *Soit T une théorie floue. Si pour toute fbf A on a $T \vdash_a A$ et $T \vdash_b A$, alors $a \leq b$.*

Une conséquence de ce théorème est que pour chaque constante logique $\mathfrak{a}$, avec $a \in B$, nous avons $T \vdash_a \mathfrak{a}$ et $T \vDash_a \mathfrak{a}$ dans toute théorie logique floue T.

THÉORÈME 9.3.2 *Soit T une théorie flou consistante et $T \vdash_a A$, $T \vdash_a B$. Alors*

(1) $T \vdash_c A \Rightarrow B$ implique $c \leq a \rightarrow b$

(2) $T \vdash_c A\&B$ implique $c \geq a \otimes b$

(3) Si $T \vdash_a (\forall X)A$, alors $a \leq \bigwedge \{b / T \vdash_b A_X[t], t \in \mathbb{T}\}$, avec $\mathbb{T}$ l'ensemble des termes de la théorie T.

(4) Si $T \vdash_c A \wedge B$, $T \vdash_a A$ et $T \vdash_b B$, alors $c = a \wedge b$.

DÉFINITION 9.3.1 *Une théorie floue est consistante s'il n'existe pas une fbf A et des démonstrations w_A et $w_{\neg A}$ de A et $\neg A$ respectivement, telles que*

$$H_T(w_A) \otimes H_T(w_{\neg A}) > 0$$

De cette définition on conclut que s'il existe une fbf A telle que $T \vdash_a A$, $T \vdash_b \neg A$ avec $a \otimes b > 0$, alors la théorie T est contradictoire.

Nous terminons cette section avec les théorèmes de complétude.

THÉORÈME 9.3.3 (DE COMPLÉTUDE - I) *Nous avons $T \vdash_a A$ si et seulement si $T \vDash_a A$ pour toute fbf A et toute théorie floue consistante T.*

THÉORÈME 9.3.4 (DE COMPLETUDE - II) *Une théorie floue est consistante si et seulement si a un modèle.*

9.4 Fonctions logiques et formes normales

Nous commençons par une généralisation de la fonction booléenne classique à deux valeurs.

DÉFINITION 9.4.1 *Soit $B = [0,1]$. Une fonction $f : B^n \rightarrow B$ est appelée fonction logique floue B-valuée ou, encore, FLF B-valuée. On notera l'ensemble des ces fonctions $\mathcal{F}_B$. Cete ensemble est l'ensemble des fonctions booléennes si $B = \{0,1\}$.*

On sait qu'une fonction booléenne peut être représentée par une formule logique adéquate du calcul propositionnel classique. On peut prouver que ce résultat n'est valide pour les FLF B-valuées que si ces fonctions sont linéaires par morceaux et avec des coefficients entiers.

Ce résultat pose un problème pour le traitement des règles floues du type "SI <premisses floues> ALORS <conclusion floue>". En effet ces règles sont transformées en FLF B-valuées. Il faut donc examiner quelles formes des fonctions peuvent être représentées par ce type des règles.

Soit une fonction booléenne $f(x_1,\ldots,x_n)$. On suppose que $f(x_1,\ldots,x_n)\not\equiv 0$. Afin de représenter la variable X et sa négation, on écrit

$$x^s = \begin{cases} \neg x, & \text{si } s=0 \\ x, & \text{si } s=1 \end{cases}$$

On a donc $x^s = 1$ si $s = 1$. Donc $f(x_1,\ldots,x_n)$ peut être représentée sous forme normale disjonctive (FND)

$$f(x_1,\ldots,x_n) = \bigvee_{f(s_1,\ldots,s_n)=1} \left(x_1^{s_1}\wedge\ldots\wedge x_n^{s_{nn}}\right) = \bigvee_{(s_1,\ldots,s_n)} \left(x_1^{s_1}\wedge\ldots\wedge x_n^{s_n}\wedge f(s_1,\ldots,s_n)\right)$$

ou sous forme normale conjonctive (FNC)

$$f(x_1,\ldots,x_n) = \bigwedge_{f(s_1,\ldots,s_n)=0} \left(x_1^{\bar{s}_1}\vee\ldots\vee x_n^{\bar{s}_n}\right) = \bigwedge_{(s_1,\ldots,s_n)} \left(x_1^{\bar{s}_1}\vee\ldots\vee x_n^{\bar{s}_n}\vee f(s_1,\ldots,s_n)\right)$$

où $\bar{s}$ signifie le complémentaire de s. La dernière partie de la forme normale conjonctive peut aussi s'écrire comme suit

$$f(x_1,\ldots,x_n) = \bigwedge_{(s_1,\ldots,s_n)} \left(x_1^{s_1}\vee\ldots\vee x_n^{s_n}\rightarrow f(s_1,\ldots,s_n)\right)$$

qui est de la forme des règles "SI <premisses floues> ALORS <conclusion floue>".

Faisons maintenant une extension au calcul des prédicats flous. Rappelons d'abord la définition d'une relation floue.

DÉFINITION 9.4.2 *Soit un domaine D non vide (sous-ensemble de l'univers du discours). Une relation logique floue (FL-relation) R d'arité n sur D est un sous-ensemble flou $R\subset D^n$ qui est donné par la fonction d'appartenance $R(x_1,\ldots,x_n)$ définie sur l'ensemble D^n et prenant ses valeurs dans B.*

Dans la suite on identifiera une FL-relation avec sa fonction d'appartenance.

Considérons les prédicats $p_1,\ldots,p_k$, qu'on suppose pour simplifier les notations d'arité 1, et des fbf $q_{i_1\ldots i_n}$ avec $1\le i_j\le k, 1\le j\le n, n\ge 1$. On peut montrer (cf [3]) que les deux formes normales ci-dessus donnent dans ce cas

– la FND-MV (forme normale disjonctive multi-valuée)

$$f(x_1,\ldots,x_n) = \bigvee_{i_1=1}^{k}\cdots\bigvee_{i_n=1}^{k}\left(p_{i_1}(x_1)\&\cdots\&p_{i_n}(x_n)\&q_{i_1\ldots i_n}\right)$$

– La FNC-MV (forme normale conjonctive multi-valuée)

$$g(x_1,\ldots,x_n) = \bigwedge_{i_1=1}^{k}\cdots\bigwedge_{i_n=1}^{k}\left(p_{i_1}(x_1)\bigtriangledown\cdots\bigtriangledown p_{i_n}(x_n)\bigtriangledown q_{i_1\ldots i_n}\right)$$

Comme précédement cette dernière forme peut s'écrire aussi

$$g(x_1,\ldots,x_n)=\bigwedge_{i_1=1}^{k}\cdots\bigwedge_{i_n=1}^{k}(p_{i_1}(x_1)\&\cdots\&p_{i_n}(x_n)\Rightarrow q_{i_1\ldots i_n})$$

Nous allons étudier brièvement la topologie de l'espace des FL-relations. On suppose que sur D nous avons défini une pseudo-métrique $\delta : D\times D\to\mathbb{R}_+$ qui est uniformément continue sur $D\times D$. Nous avons deux définitions :

DÉFINITION 9.4.3 *Une FL-relation $R\subset D^n$ d'arité n est uniformément continue sur D^n si pour tout ε, avec $0<\varepsilon<1$, il existe un réel $r>0$ tel que si $\delta(x_i,y_i)<r$; $i=1,\ldots,n$, alors $\mid R(x_1,\ldots,x_n)-R(y_1,\ldots,y_n)\mid<\varepsilon$.*

DÉFINITION 9.4.4 *Soient deux FL relations $R,S\subset D^n$. On dit que R ε-approxime S si*

$$\mid R(x_1,\ldots,x_n)-S(x_1,\ldots,x_n)\mid\leq\varepsilon\,;\ \forall(x_1,\ldots,x_n)\in D^n$$

Le théorème suivant précise les conditions d'une FL-relation R par une FND ou FNC.

THÉORÈME 9.4.1 *Soit une FL-relation $R\subset D^n$ qui est uniformément continue sur D^n. Alors pour tout ε avec $0<\varepsilon<1$, existe une FND $f(x_1,\ldots,x_n)$ (resp. une FNC $g(x_1,\ldots,x_n)$) qui ε-approxime $R(x_1,\ldots,x_n)$..*

9.5 Règles de composition

Une *règle de composition* est une procédure qui fournit un résultat qui dépend d'une relation donnée dont ses valeurs indépendantes sont contraintes par des relations floues ou déterministes. Les règles du type "SI <premisses floues> ALORS <conclusion floue>".sont des cas particuliers des règles de composition. L'objectif est d'en extraire une fonction normale qui décrit la FL-relation de la règle. On utilise pour ce faire une procédure de defuzzification qui est une opération floue sur un ensemble d'éléments.

DÉFINITION 9.5.1 *Une defuzzification d'un ensemble flou $X\subset U$ qui est donné par sa fonction d'appartenance $\mu_X : U\to[0,1]$ est une application*

$$\Theta : \mathcal{F}(U)\to U \text{ telle que } \mu_X(\Theta(X))>0$$

où $\mathcal{F}(U)$ représente l'ensemble de tous les sous-ensembles flous de U.

Il y a plusieurs fonctions pour le calcul de la defuzzification. Nous donnons par la suite une liste non exhaustive.

Min des max $\Theta(X)=\inf\left\{x\Big/\mu_X(x)=\sup_{u\in U}\mu_X(u)\right\}$

Moyenne des max $\Theta(X) = \dfrac{x_L + x_G}{2}$; avec $x_L = \inf\left\{ x \Big/ \mu_X(x) = \sup\limits_{u \in U} \mu_X(u) \right\}$ et $x_G = \inf\left\{ x \Big/ \mu_X(x) = \sup\limits_{u \in U} \mu_X(u) \right\}$

Centre de gravité $\Theta(X) = \dfrac{\int_X x\mu_X(x)\,dx}{\int_X \mu_X(x)\,d}$ dans le cas continu

$\Theta(X) = \dfrac{\sum\limits_{i=1}^{n} x_i \mu_X(x)}{\sum\limits_{i=1}^{n} x\mu_X(x)}$ dans le cas discret, avec $U = \{x_1, , x_n\}$

Nous verrons maintenant qu'étant donnée une fonction de defuzzification on peut faire correspondre une fonction à une FL-relation $R(x_1, \ldots, x_n, y)$ définie sur l'ensemble $X_1 \times \ldots \times X_n \times Y$. Pour unr fonction de defuzzification Θ on définit la fonction $f_{R,\Theta}(x_1, \ldots, x_n) : X_1 \times \ldots \times X_n \to Y$ en posant

$$f_{R,\Theta}(x_1, \ldots, x_n) = \Theta(R(x_1, \ldots, x_n, y))$$

On peut montrer que la fonction $f_{R,\Theta}$ ε-approxime f, c'est-à-dire pour tout $(x_1, \ldots, x_n) \in D^n$ on a $| f(x_1, \ldots, x_n) - f_{R,\Theta}(x_1, \ldots, x_n) | < \varepsilon$ où $\varepsilon > 0$.

DÉFINITION 9.5.2 *Soit $f : [a,b] \to \mathbb{R}$ fonction continue et soit G un ensembe des fonctions continues d'approximation définies sur le même intervalle. Notons par d la distance entre deux fonctions réelles définies sur $[a,b]$. On dit que $g^* \in G$ est la meilleure approximation de f si*

$$d(f, g^*) \leq d(f, g) \quad \forall g \in G$$

On va maintenant établir la meilleure approximation de f par de FL-relations. Soient les prédicats d'arité 1 p_i, q_i ; $1 \leq i \leq n$. Considérons une fonction continue $f : [a,b] \to [c,d]$ et les FL-relations représentées par les formes normales :

$$\text{FND} : f(x,y) = \bigvee_{i=1}^{n} (p_i(x) \,\&\, q_i(y))$$

et

$$\text{FNC} : \; g(x,y) = \bigwedge_{i=1}^{n} (p_i(x) \Rightarrow q_i(y))$$

Il faut construire un ensemble D qui permet l'interprétation de deux formes normales ci-dessus et tel que les FL-relations $f(x,y)$ et $g(x,y)$ sont égales à $R(x,y)$ où

$$R(x,y) = \begin{cases} 1, \text{ si } (\exists i)\left(1 \leq i \leq n \text{ et } x \in I_i, y \in \overline{f}[I_i]\right) \\ 0, \text{ sinon} \end{cases}$$

avec $I_1, \ldots, I_n$ intervalles ouverts qui sont un recouvrement de D, et $\overline{f}[I_i] = \left\{ y \in [a,b] \,/\, f(x,y) = 1 \text{ pour } x \in I_i \right\}$.

L'ensemble des approximations pour f est donné par

$$\mathcal{G}_f = \left\{ g(x) = f_{R,\Theta}(x) = \Theta(R(x,y)) \,/\, \Theta \text{ fonction de defuzzification} \right\}$$

Notons $g_{MOM}(x) \in \mathcal{G}_f$ la fonction spécifiée par la defuzzification moyenne des max. Considérons les intervalles fermés $\bar{I}_i = [x_i, x_{i+1}], 1 \leq i < n$. La restriction continue de g_{MOM} dans $\bar{I}_i$ est la meilleure approximation dans $\mathcal{G}_f$ de la restriction de f dans $\bar{I}_i$ par rapport à la distance d.

9.6 Références

Pour la redaction de ce chapitre les livres suivants ont été consultés.

[1] S. Gottwald : *A treatise on many-valued logic*, Res. Stud. Press, 2001

[2] P. Hájek : *Metamathematics of fuzzy logic*, Kluwwer, 1998

[3] V. Novák et al. : *Mathematical principles of fuzzy logic*, Kluwer, 2000

[4] E. Turunen : *Mathematics behind fuzzy logic*, Physica Verlag, 1999

Table des matières