Logique et fondements de l'Informatique

Logique du 1^{er} ordre, calculabilité et λ-calcul

Richard Lassaigne
Université de Paris VII

Michel de Rougemont
Université de Paris-II, Panthéon-Assas

HERMES 1993

To order this book, contact Claudine BEMBARON
email : 100442.3133@compuserve.com
FAX: (1) 42 29 15 56, TEL: (1) 42 29 44 66.

Table des matières

Introduction

Le développement de la Logique mathématique et celui de l'Informatique dans la dernière décennie ont mis en lumière un certain nombre d'interactions entre preuve et calcul, modèles de machines et fonctions calculables, pouvoir d'expression d'un langage et complexité des problèmes exprimés dans ce même langage. L'objectif de cet ouvrage est de présenter les concepts et méthodes logiques de base qui permettront d'approfondir l'étude de ces diverses interactions, chacune d'elles pouvant faire l'objet d'un travail à part entière.

L'origine de la Logique mathématique contemporaine en tant qu'outil de formalisation, est marquée par les publications de G.Frege (1879 et 1892) sur le langage des formules, sur le sens et la dénotation. L'utilisation d'un langage formel dans un cadre mathématique, informatique ou linguistique, soulève une question fondamentale : quelle sémantique peut-on associer à un tel langage ? Un énoncé est représenté par une référence (ou dénotation), mais possède également un sens plus général.

Pour les langages de la Logique du premier ordre, la dénotation est donnée en termes de valeurs de vérité. La notion syntaxique de preuve est définie à l'aide d'un système d'axiomes et de règles de déduction. Dans sa thèse, K. Gödel démontra en 1930 la propriété de complétude pour un tel système, c'est-à-dire l'adéquation entre aspect sémantique et aspect syntaxique. L'année suivante, il publiait ses deux théorèmes d'incomplétude et obligeait ainsi la Logique, en tant que métamathématique, à redéfinir ses objectifs : il montrait l'inexistence d'une formalisation complète de l'arithmétique et apportait ainsi la preuve de l'impossibilité de résoudre l'un des problèmes énoncés dans le fameux programme de Hilbert. Ce dernier voulait réduire les mathématiques aux déductions logiques à partir de simples axiomes.

A la même époque, se développaient les bases de la théorie des fonctions récursives. Cette théorie étudie les fonctions pour lesquelles il existe une méthode effective de calcul. Elle a pris son essor à partir des travaux de Church, Kleene, Markov, Post et Turing : les différentes classes de fonctions qu'ils définissent sont constituées des fonctions calculables, elles incluent toutes les fonctions calculables connues et coïncident toutes. Cette constatation amena Church à proposer en 1936 sa fameuse thèse : *toute fonction calculable est récursive.* A la même époque, A.Turing définit une machine abstraite [**?**], qui porte maintenant son nom et montre que les fonctions calculables sur cette machine sont exactement les fonctions récursives.

L'impulsion donnée par Gödel à la Logique se retrouve également dans la théorie de la démonstration développée à partir des travaux de Herbrand et Gentzen. Le théorème de Herbrand (1929), qui précise le caractère semi-décidable de la Logique du premier ordre, est le fondement des méthodes de démonstration automatique. Les travaux de Gentzen (1935) ont ensuite permis de développer de nouvelles méthodes applicables à un système de règles de déduction, sans axiomes, appelé calcul des séquents. Ce genre de système a été utilisé récemment par J.Y.Girard pour construire la Logique linéaire, dont l'un des objectifs est de permettre l'expression des problèmes de ressources et de communication dans les systèmes informatiques.

Les systèmes de types sont des systèmes récents de déduction naturelle pour lesquels il est possible d'établir une correspondance entre preuves et programmes fonctionnels, ainsi qu'entre formules et types : la construction d'un terme fonctionnel typé associé à un programme d'un certain type, est la trace de la preuve de la formule exprimant le type considéré. Ce domaine de recherche est l'un des plus actifs actuellement en Logique.

Enfin, la théorie des modèles s'est constituée vers 1950 à partir des travaux de A.Tarski. Les notions de définissabilité implicite et explicite, dans un langage du premier ordre, ont été introduites par Beth et se sont révélées fondamentales pour la théorie des modèles finis, particulièrement importante en Informatique. Cet ouvrage a une suite entièrement consacrée à la définissabilité logique et à la théorie de la complexité où ces aspects sont approfondis.

Le développement de l'Informatique est beaucoup plus récent et a connu un essor important à partir des années 60, grâce à des progrès technologiques continus. Ainsi, la puissance des machines est régulièrement doublée, alors que leur prix diminue. Cependant, ces progrès ne peuvent garantir de solution aux problèmes pratiques sans qu'une étude théorique soit faite sur les limitations des méthodes utilisées sur ces machines. On distingue alors les modèles de machines des langages qui permettent de programmer ces machines.

Les *programmes* sont les expressions données aux machines pour être exécutées. Ce sont bien des expressions d'un langage auxquelles Frege associerait un sens et une dénotation. La dénotation est classiquement la fonction qui associe à l'entrée du programme (x) la sortie ($y = f(x)$). Mais quel est son sens ? La complexité en temps et en espace est une des composantes du sens informatique : il est donc légitime d'introduire les notions fondamentales de complexité.

Un *langage de programmation* est un ensemble d'expressions, au même titre que l'anglais est un ensemble d'expressions grammaticalement correctes. Les premiers langages de programmation tels que FORTRAN étaient conçus comme des traducteurs du langage de programmation [1] en langage des instructions machines : il était donc nécessaire de pouvoir isoler une suite d'instructions à partir d'un programme. Dans ce livre, nous insisterons sur l'influence de la logique sur les langages de programmation, en particulier sur la sémantique des langages fonctionnels et de la programmation logique :

- Dans le cas d'un langage fonctionnel typé, la compilation d'un programme correspond à une vérification ou à une inférence de types. L'évaluation du programme repose sur une stratégie de réduction qui permet d'obtenir une forme normale du programme fonctionnel. L'un des objectifs de la recherche actuelle sur les systèmes logiques de types est la conception d'environnements de programmation permettant la preuve de programmes, c'est-à-dire leur correction.

- Dans le cas de la programmation logique, la stratégie d'un programme peut être considérée comme un cas particulier d'une méthode générale de démonstration automatique : la résolution. A chaque étape, on applique une règle de déduction, effectuée à l'aide

[1] FORTRAN est l'abréviation FORmulas TRANslators.

d'un algorithme d'unification sur les termes de la logique du 1er ordre.

Si la machine de Babbage est souvent reconnue comme étant une des premières machines informatiques, c'est Turing qui donne pour la première fois une définition précise d'un modèle de calcul et étudie les fonctions définissables à l'aide d'un tel modèle, appelées aussi fonctions *calculables au sens de Turing.* L'équivalence entre ces fonctions calculables au sens de Turing et les fonctions récursives partielles est une des premières interactions entre Logique et Informatique.

Turing reconnaissait implicitement l'importance du temps et de l'espace de calcul d'une machine de Turing car, dans son fameux *test de Turing*, il comparait une telle machine à un être humain, du point de vue à la fois des réponses et du temps de calcul : si la machine répond comme l'être humain, alors seulement elle lui sera comparable. L'étude de la complexité en temps et en espace est donc bien, au départ, une des questions fondamentales de l'Informatique. Si le modèle de Turing est souvent le plus classique, d'autres modèles existent tels que les RAM (Random Access Machines) ou machines à accès aléatoire plus proches des machines actuelles, pour lesquelles certaines notions de complexité sont différentes[2]. Enfin, la généralisation de ces modèles aux machines probabilistes est un sujet important qui sera abordé dans le second volume.

La théorie de la complexité a permis d'isoler certaines classes de problèmes qui étaient *aussi* difficiles à résoudre en prenant en compte les ressources de temps et d'espace, en particulier deux grandes classes P et NP. P est la classe des problèmes que l'on peut résoudre en *temps polynomial* , c'est-à-dire par des programmes dont le temps de calcul est borné par une fonction polynomiale de la taille de l'entrée. NP est la classe des problèmes que l'on peut *vérifier*[3] en temps polynomial. Une des observations fondamentales de Cook, Karp et Levin est le fait qu'il existe certains problèmes NP plus difficiles que tous les autres problèmes NP. Plus récemment, la théorie de la complexité s'est intéressée

[2]Pour des bornes polynomiales sur le temps de calcul, les modèles de Turing et les RAM sont équivalents ; ce n'est pas le cas pour une borne linéaire.

[3]Etant donné x, y on décide si $f(x) = y$.

à la classe IP généralisant[4] NP, en donnant au vérificateur le pouvoir d'utiliser le hasard. Dans ces développements, l'Informatique influence directement la logique, car la notion même de preuve se trouve fondamentalement modifiée.

Dans ce premier volume, nous décrirons les bases de la logique, puis la théorie des fonctions récursives et de la calculabilité et enfin le λ-calcul. Le deuxième volume sera consacré à la définissabilité et à la théorie de la complexité.

Mode d'emploi

Ce cours recouvre notre enseignement de la logique en Licence, Maîtrise, Ecoles d'Ingénieurs et DEA, divisé en trois parties :

1. *Introduction à la Logique :* le calcul propositionnel et la logique du 1^{er} ordre. L'aspect déductif et sa complétude sont montrés en suivant les méthodes de déduction naturelle et de résolution.

2. *Calculabilité* : la classe des fonctions récursives est introduite et l'on montre son équivalence avec la classe des fonctions Turing-calculables.

3. *λ-calcul* et systèmes de types.

Le dernier chapitre est constitué d'exercices pour trois logiciels d'enseignement qui illustrent les idées principales de ce livre. *Tarski 's World* est un logiciel qui permet la construction de modèles géométriques et de tester la validité de formules du 1^{er} ordre. Le *Tableau System* permet de déduire par résolution un théorème à partir d'axiomes. *Turing's World* donne la possibilité de construire des machines de Turing à 1 ruban et de simuler leur comportement.

Chaque chapitre est suivi d'exercices, dont certains sont corrigés à la fin du livre.

[4] IP est l'abréviation de *Interactive Proof*, c'est-à-dire des problèmes qui admettent une preuve interactive.

Remerciements

Marouan Ajlani, Daniel Andler, Gilles Amiot, Jean-Pierre Azra, Nathalie Barthes, Jean-Pierre Bénéjam, Chantal Berline, Claude-Laurent Bernard, Georges Blanc, Spéphane Boucheron, Elisabeth Bouscaren, Jean-Pierre Calais, Vincent Campagnie, Serena Cerrito, François Conduché, Jean Coret, René Cori, Vincent Danos, Juan Franciso Diaz-Frias, Max Dickmann, Patrick Dehornoy, Françoise Delon, Thomas Ehrhard, Marie-Christine Ferbus, Jean-Yves Girard, Serge Grigorieff, Nicolas Guelfi, Philippe Ithier, Bernard Jaulin, Jean-Louis Krivine, Ramez Labib-Sami, Daniel Lacombe, Zoé Lacroix, Daniel Lascar, Yves Legrand-Gérard, François Lucas, Kenneth Mac-Aloon, Janos Makowsky, Sophie Malecki, Jean Malifaud, Jean-Luc Marion, Catherine Muhlrad-Greif, Michel Parigot, Christophe Rafalli, Laurent Régnier, Jean-Pierre Ressayre, Carlos Rodriguez, Paul Rozière, Gabriel Sabbagh, Miklös Santha, Jacques Stern, Avy Sharell, Anne Strauss, Sovanna Tan Françoise Ville.