Contribution à l'analyse de la methode des tableaux

Michel Levy

THESE

présentée par

Michel LEVY

pour obtenir le titre de DOCTEUR
DE L'UNIVERSITE JOSEPH FOURIER - GRENOBLE 1
(arrêté ministériel du 5 juillet 1984)

SPECIALITE : INFORMATIQUE

CONTRIBUTION A L'ANALYSE DE LA METHODE DES TABLEAUX

Thèse soutenue le 15 mars 1991

Composition du jury:

Président	Pierre Claude SCHOLL
Rapporteurs	Pierre SIEGEL
	Joseph SIFAKIS
Examinateurs	Pierre BERLIOUX
	Laurent TRILLING

Thèse préparée au sein du Laboratoire de Génie Informatique

Titre : Contribution à l'analyse de la méthode des tableaux

Résumé : L'intérêt de la méthode des tableaux réside dans le fait que, contrairement aux méthodes usuelles de démonstration automatique par résolution, elle n'exige aucun prétraitement des formules.
Initialement créée pour des formules fermées, cette méthode a été généralisée par M.Fitting pour permettre l'utilisation de variables libres et d'algorithmes d'unification. Ceci l'a conduit à une reformulation des règles d'élimination des quantificateurs universels (gamma-règle) et des quantificateurs existentiels (delta-règle). La delta-règle décrite par M.Fitting apparait à mon avis insuffisamment prouvée. Aussi j'ai entrepris de l'analyser en présentant la méthode des tableaux comme un algorithme de construction de listes de buts à satisfaire.

Grâce à cette nouvelle présentation, j'ai pu :
(1) établir que la delta-règle de Fitting est en fait une skolémisation pour un quantificateur existentiel (resp. universel) ayant une occurrence positive (resp. négative) dans la liste des buts considérée comme une disjonction de conjonctions.
(2) améliorer la delta-règle de Fitting afin que le nouveau symbole de fonction ajouté par la règle ait le moins possible d'arguments.
(3) décrire concisément et précisément des classes d'algorithmes de démonstration automatique pour la méthode des tableaux avec ou sans variables libres.

Mots-clés : logique du premier ordre, démonstration automatique, méthode des tableaux

Title : A contribution to the analysis of a tableaux based decision method

Abstract : The interest of the tableaux based decision method stems from the fact that formulas need not to be put into some particular form, contrary to the usual methods of automatic demonstration by resolution. This method has been first formulated for closed formulas and has been extended by M.Fitting for formulas with free variables and for the application of unification algorithms. To this aim, he reformulated the rules for elimination of universal quantifiers (gamma-rule) and of existential quantifiers (delta-rule). The delta-rule given by M.Fitting turned out to be insufficiently proven. The subject of this thesis is the analysis of this rule by means of a presentation of the tableaux based decision method as an algorithm of construction of list of goals to be satisfied.

Due to this original presentation, it has been possible
(1) to establish that the delta-rule of M.Fitting is in fact a skolemisation for an existential (resp. universal) quantifier with a positive (resp. negative) occurrence in the goal list considered here as a disjunction of conjunctions.
(2) to improve the delta-rule of M.Fitting such that the new function symbol added by the rule has the minimal number of arguments.
(3) to describe concisely classes of algorithms of automatic demonstration by tableaux based methods for formulas with or without free variables.

Keywords : first-order logic, automated theorem proving, tableaux method

Je tiens à remercier tous les membres du jury,

Monsieur Pierre-Claude Scholl, Professeur à l'Université Joseph Fourier, pour l'honneur qu'il me fait en présidant le jury de cette thèse et pour tous les efforts qu'il a réalisés en tant que directeur de l'U.F.R. I.M.A. afin de m'aménager du temps libre pour que je puisse me consacrer à la recherche,

Monsieur Pierre Siegel, Professeur à l'Université de Provence, qui a bien voulu juger ce travail,

Monsieur Joseph Sifakis, Directeur de Recherches au C.N.R.S., qui a accepté la tâche de rapporteur et qui m'a témoigné de sa confiance et de ses encouragements,

Monsieur Pierre Berlioux, Maitre de conférences à l'I.N.P.G., qui a relu avec patience plusieurs versions de cette thèse, qui a contribué par ses discussions et ses critiques à clarifier mes idées et qui, grâce à sa collaboration depuis de nombreuses années, a soutenu mon intérêt pour la logique et l'a transformé en un réel enthousiasme,

Monsieur Laurent Trilling, Professeur à l'Université Joseph Fourier, pour la confiance qu'il a montrée dans l'aboutissement de ce travail et pour toute l'aide qu'il a m'a consacrée avec une générosité qui m'a profondément touchée.

Je tiens également à remercier,

Anne Rasse et Philippe Bizard, qui ont relu des parties de cette thèse, m'ont beaucoup aidé matériellement pendant la préparation de ce texte et qui ont su m'encourager, quand j'étais prêt à tout abandonner. Sans eux, je n'aurais pas pu terminer à temps ce travail.

Je remercie enfin mes filles Florence et Clara et ma femme Silvia, pour toute l'affection qu'elles m'ont offerte dans les moments difficiles de la rédaction de cet ouvrage.

Table des matières

Introduction

Intérêt de la méthode

La logique du premier ordre est enseignée avec trois types de systèmes formels :
(1) les systèmes à la Hilbert qui n'utilisent qu'une ou deux règles de déduction (le modus ponens et la généralisation),
(2) les systèmes de Gentzen (calcul des séquents et déduction naturelle) où il y a deux règles de déduction par connective,
(3) la méthode des tableaux où il y a aussi deux règles de déduction par connective, qui présente l'avantage sur le calcul des séquents d'éviter des copies de formules.
Il est beaucoup plus facile de faire des preuves avec les systèmes de Gentzen ou la méthode des tableaux qu'avec un système à la Hilbert. Aussi dans l'enseignement de la logique, il serait préfèrable d'utiliser les systèmes de Gentzen ou la méthode des tableaux.

La démonstration automatique en logique du premier ordre, est souvent fondée sur la méthode de résolution, qui peut être vue comme un quatrième type de système formel, avec une seule règle indiquant comment obtenir un résolvant de deux clauses. Cette méthode présente certains inconvénients : une preuve obtenue par la méthode de résolution est peu compréhensible et cette méthode exige un prétraitement des formules qui les rend peu lisibles.

Au contraire, la méthode des tableaux n'exige aucun prétraitement des formules à prouver. Aussi est-elle particulièrement intéressante comme base d'algorithmes de démonstration automatique (de préférence aux algorithmes de démonstration automatique fondés sur la méthode de résolution) dans les trois cas suivants :
(1) pour réaliser des démonstrateurs semi-automatiques (interactifs), l'absence de prétraitement des formules permettant une interaction plus naturelle entre un démonstrateur et un utilisateur,
(2) quand on veut obtenir d'un démonstrateur non pas la réponse «cette formule est prouvée» mais une preuve compréhensible de la formule prouvée,
(3) pour permettre une extension de la méthode aux logiques modales pour lesquelles certains prétraitements sont impossibles.

Historique

La méthode des tableaux sémantiques a été introduite par [E.W.Beth 59] et [K.J.J.Hintikka 55] . Nous utilisons la présentation précise qui en faite par [P.B.Andrews 86] où un tableau est en fait un arbre de formules, dans lequel tout sommet ayant deux successeurs correspond à une disjonction et toute branche de l'arbre à une conjonction de formules. Cette méthode est un système formel de déduction correct et complet, beaucoup plus naturel que les « systèmes à la Hilbert », donc plus adapté à l'enseignement de la logique. Par contre l'utilisation de cette méthode comme

base d'un algorithme de démonstration automatique est moins évidente, parce qu'il est difficile d'indiquer quand une formule n'est plus nécessaire (dans une stratégie complète) pour la suite de la démonstration. Prenons un exemple.

Pour prouver la validité de la formule $\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x)$ on procède par réfutation, en supposant que l'on peut satisfaire $\neg(\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x))$.
On construit alors l'arbre ci-dessous :

(ε) $\neg(\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x))$

(1) $\forall x(p(x) \wedge q(x))$ *de (ε)*

(11) $\neg(\forall x p(x) \wedge \forall x q(x))$ *de (ε)*

(111) $\neg\forall x p(x)$ *de (11)*

(1111) $\neg p(a)$ *de (111), a constante*

(11111) $p(a) \wedge q(a)$ *de (1)*

(111111) $p(a)$ *de (11111)*

(1111111) $q(a)$ *de (11111)*

branche fermée par $p(a), \neg p(a)$

(112) $\neg\forall x q(x)$ *de (11)*

(1121) $\neg q(a)$ *de (112), a constante*

(11211) $p(a) \wedge q(a)$ *de (1)*

(112111) $p(a)$ *de (11211)*

(1121111) $q(a)$ *de (11211)*

branche fermée par $q(a), \neg q(a)$

On note que la formule $\forall x(p(x) \wedge q(x))$ a été instanciée une fois sur chaque branche, dans des sommets qui ne sont pas des successeurs immédiats du sommet (1). Pour garder un aspect **local** à chaque étape élémentaire de la preuve, nous préférons une présentation très proche du calcul des séquents, qui fait correspondre un sommet à chaque application de règle.

Présentation de la méthode des tableaux sans variables avec règles locales.
Reprenons l'exemple précédent. Dans l'arbre ci-dessous, chaque sommet correspond à l'application d'une règle. On peut noter que la formule $\forall x(p(x) \wedge q(x))$ apparait au sommet (111) et qu'elle est recopiée avec son instance $p(a) \wedge q(a)$ au sommet (1111).

(ε) $\neg(\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x))$

(1) $\forall x(p(x) \wedge q(x)), \neg(\forall x p(x) \wedge \forall x q(x))$

(11) $\forall x(p(x) \wedge q(x)), \neg\forall x p(x)$

(111) $\forall x(p(x) \wedge q(x)), \neg p(a)$

(1111) $\forall x(p(x) \wedge q(x)), p(a) \wedge q(a), \neg p(a)$

(11111) $\forall x(p(x) \wedge q(x)), p(a), q(a), \neg p(a)$

feuille fermée

(12) $\forall x(p(x) \wedge q(x)), \neg\forall x q(x)$

(121) $\forall x(p(x) \wedge q(x)), \neg q(a)$

(1211) $\forall x(p(x) \wedge q(x)), p(a) \wedge q(a), \neg q(a)$

(12111) $\forall x(p(x) \wedge q(x)), p(a), q(a), \neg q(a)$

feuille fermée

Appelons but une liste de formules. Les règles définissent comment réaliser un but en réalisant un ou deux sous-buts. Ces règles sont données au paragraphe 1.4.2.1. On peut voir sur l'exemple que les règles sont écrites de telle sorte qu'un but est insatisfaisable **si et seulement si** tous ses sous-buts sont insatisfaisables.
La preuve de $\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x)$ est terminée, quand tous les buts obtenus sont insatisfaisables. La formule $\neg(\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x))$ est alors insatisfaisable, donc $\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x)$ est valide.

Dérivation et arbre de dérivation

J'ai essayé de séparer clairement les règles logiques utilisées par un démonstrateur automatique et le contrôle qu'il faut exercer sur l'application de ces règles pour obtenir un démonstrateur complet.

Pour toute formule fermée A, les algorithmes de démonstration automatique du chapitre 1 construisent une dérivation commençant par $\neg A$ en utilisant les règles données au paragraphe 1.4.2.1.
Par exemple pour la formule $\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x)$, il est possible qu'une exécution d'un tel algorithme produise la dérivation :

$[[\neg(\forall x(p(x) \wedge q(x)) \Rightarrow \forall x p(x) \wedge \forall x q(x))]^0]$

$[[\forall x(p(x) \wedge q(x)), \neg(\forall x p(x) \wedge \forall x q(x))]^0]$

$[[\forall x(p(x) \wedge q(x)), \neg\forall x p(x)]^0, \quad [\forall x(p(x) \wedge q(x)), \neg\forall x q(x)]^0]$

$[[\forall x(p(x) \wedge q(x)), \neg p(c_1)]^1, \ [\forall x(p(x) \wedge q(x)), \neg\forall x q(x)]^0]$

$[[\ \forall x(p(x) \land q(x)), \neg p(c_1)\ ]^1,\ [\ \forall x(p(x) \land q(x)), \neg q(c_1)\ ]^1\ ]$

$[[\ \forall x(p(x) \land q(x)), p(c_1) \land q(c_1), \neg p(c_1)\ ]^1,\ [\ \forall x(p(x) \land q(x)), \neg q(c_1)\ ]^1\ ]$

$[[\ \forall x(p(x) \land q(x)), p(c_1) \land q(c_1), \neg p(c_1)\ ]^1,\ [\ \forall x(p(x) \land q(x)), p(c_1) \land q(c_1), \neg q(c_1)\ ]^1\ ]$

$[[\ \forall x(p(x) \land q(x)), p(c_1), q(c_1), \neg p(c_1)\ ]^1,\ [\ \forall x(p(x) \land q(x)), p(c_1) \land q(c_1), \neg q(c_1)\ ]^1\ ]$

$[[\ \forall x(p(x) \land q(x)), p(c_1), q(c_1), \neg p(c_1)\ ]^1,\ [\ \forall x(p(x) \land q(x)), p(c_1) \land q(c_1), \neg q(c_1)\ ]^1\ ]$

$[[\ \forall x(p(x) \land q(x)), p(c_1), q(c_1), \neg p(c_1)\ ]^1,\ [\ \forall x(p(x) \land q(x)), p(c_1), q(c_1), \neg q(c_1)\ ]^1\ ]$

A une dérivation, on associe un arbre de dérivation de la même manière que l'on peut associer un arbre de dérivation à une dérivation dans une grammaire hors-contexte.
A la dérivation ci-dessus correspond l'arbre

(ε) $[\ \neg(\forall x(p(x) \land q(x)) \Rightarrow \forall x p(x) \land \forall x q(x)\ )\ ]^0$

(1) $[\ \forall x(p(x) \land q(x)), \neg(\forall x p(x) \land \forall x q(x)\ )\ ]^0$

(11) $[\ \forall x(p(x) \land q(x)), \neg \forall x p(x)\ ]^0$

(111) $[\ \forall x(p(x) \land q(x)), \neg p(c_1)\ ]^1$

(1111) $[\ \forall x(p(x) \land q(x)), p(c_1) \land q(c_1), \neg p(c_1)\ ]^1$

(11111) $[\ \forall x(p(x) \land q(x)), p(c_1), q(c_1), \neg p(c_1)\ ]^1$

feuille fermée

(12) $[\ \forall x(p(x) \land q(x)), \neg \forall x q(x)\ ]^0$

(121) $[\ \forall x(p(x) \land q(x)), \neg q(c_1)\ ]^1$

(1211) $[\ \forall x(p(x) \land q(x)), p(c_1) \land q(c_1), \neg q(c_1)\ ]^1\ ]$

(12111) $[\ \forall x(p(x) \land q(x)), p(c_1), q(c_1), \neg q(c_1)\ ]^1$

feuille fermée

Un algorithme de démonstration automatique est complet si pour toute formule fermée A, si A est valide il se termine en annonçant que A est valide et si A n'est pas valide il se termine en annonçant que A n'est pas valide ou il ne termine pas.
Une dérivation qui ne se termine pas est équitable si, dans toute branche infinie b de l'arbre qui est associé à la dérivation, il n'y a pas de sommet comportant une formule et sa négation et si toutes les formules de la branche b ont été dérivées. Nous avons montré qu'un algorithme est complet, si toutes ses dérivations infinies sont équitables.

Extraction d'un modèle

Nous avons étudié complètement le cas où un algorithme de démonstration automatique se termine

en annonçant que A n'est pas valide, pour montrer comme il était possible d'extraire un modèle de $\neg A$ de la dernière étape de la dérivation commencée par $\neg A$.
Nous présentons l'arbre de dérivation pour un tel exemple.

$(\varepsilon)\ [\ \neg(\forall x(p(x) \vee q(x)) \Rightarrow \forall x p(x)\ )\ ]^0$

$(1)\ [\ \forall x(p(x) \vee q(x)), \neg\forall x p(x)\ ]^0$

$(11)\ [\ \forall x(p(x) \vee q(x)), \neg p(c_1)\ ]^1$

$(111)\ [\ \forall_1 x(p(x) \vee q(x)), p(\ c_1) \vee q(c_1), \neg p(c_1)\ ]^1$

$(1111)\ [\ \forall_1 x(p(x) \vee q(x)), p(c_1), \neg p(c_1)\ ]^1$

branche fermée

$(1112)\ [\ \forall_1 x(p(x) \vee q(x)), q(c_1), \neg p(c_1)\ ]^1$

liste terminale

Le sommet (1112) est non dérivable (d'après les règles 1.4.2.1). Il définit le modèle suivant de la formule $\neg(\forall x(p(x) \vee q(x)) \Rightarrow \forall x p(x)\ )$:

domaine c_1
base $q(c_1), \neg p(c_1)$

Skolémisation

La skolémisation d'une formule est une transformation souvent décrite, mais presque toujours mal expliquée parce qu'elle est présentée comme une transformation qui change une formule A en une formule universelle B qui a un modèle si et seulement si A a un modèle.
Plutôt que de décrire la transformation complète, j'ai décrit seulement comment on peut enlever **une** occurrence positive (resp. négative) d'un quantificateur existentiel (resp. universel) et j'ai pris modèle sur la transformation étoile de [P.B.Andrews 86] pour obtenir des fonctions de Skolem ayant le nombre minimal d'arguments.
Prenons un exemple :

$\forall x[p(x) \wedge \exists y s(y) \wedge \neg\forall y(\ q(x, y) \vee \exists x\ r(x, y)\ )\ ]$

Puisqu'il n'y a pas de variable libre dans $\exists y s(y)$, on change $\exists y s(y)$ en $s(a)$, où a est une constante et on obtient :

$\forall x[p(x) \wedge s(a) \wedge \neg\forall y(\ q(x, y) \vee \exists x\ r(x, y)\ )\ ]$

Puisque x est une variable libre de $\neg\forall y(q(x, y) \vee \forall x\ r(x, y)\ )$, on substitue f(x) à y dans la formule $(q(x, y) \vee \forall x\ r(x, y))$, on renomme la variable liée de $\forall x\ r(x, y)$ au cours de la substitution et on obtient :

$\forall x[p(x) \wedge s(a) \wedge \neg(\ q(x, f(x)) \vee \exists z\ r(z, f(x))\ )\ ]$.

Méthode des tableaux avec variables libres

Il est possible d'étendre la méthode des tableaux sans variables libres, pour des formules comportant des symboles fonctionnels. Il suffit de disposer d'un algorithme pour énumérer les termes du domaine de Herbrand (appelons t_i le i-ème terme énuméré par cet algorithme) et de prévoir pour le quantificateur universel la règle consistant à remplacer la formule $\forall_k xA$ par les deux formules $\forall_{k+1} xA$, $A[x:=t_{k+1}]$, l'indice du quantificateur permettant de contrôler que l'on instancie une et une seule fois toute formule universelle sur tout le domaine de Herbrand. Il est clair que cette méthode de démonstration automatique est déraisonnable, et la méthode des tableaux avec variables libres a été introduite par [M.Fitting 88] pour éviter cette énumération systématique du domaine de Herbrand grâce à l'emploi d'algorithmes d'unification.

Cette méthode est exposée par Fitting avec les règles non locales traditionnelles et mon premier travail a été de changer cette présentation. Appelons but une liste de formules et tableau une liste de buts, les règles sont décrites comme un moyen de dériver un tableau d'un autre tableau. A un but on associe la conjonction de ses formules et à un tableau on associe la disjonction des formules associées à ses buts.

Avec cette nouvelle façon de voir la méthode des tableaux, il devient facile, contrairement à [M.Fitting 90] de prouver la correction de la delta-règle et aussi de l'améliorer : cette règle est simplement l'élimination (skolémisation) d'un quantificateur existentiel (resp. universel) d'occurrence positive (resp. négative) de la formule associée à un tableau.

Mes algorithmes de démonstration automatique construisent des dérivations, c'est-à-dire des suites de tableaux. Comme dans la méthode des tableaux sans variables, j'associe un arbre de dérivation à une dérivation. Une dérivation qui ne se termine pas est équitable si toute branche b de l'arbre qui est associé à la dérivation est complète (c'est-à-dire b est infinie ou l'extrémite de b est étiquetée par une liste de formules non dérivables). J'ai montré qu'un algorithme est complet si toutes ses dérivations sont équitables.

Chapitre 1
Méthode des tableaux sans variables libres

Dans ce chapitre, nous analysons la méthode bien connue des tableaux sans variables libres avec une présentation nouvelle.
Un tableau est pour nous une liste de buts à satisfaire (un but étant une liste de formules) et les règles permettent de remplacer un de ces buts par un ou deux nouveaux buts.
Pour établir la validité d'une formule A, nous partons du but $\neg A$ à satisfaire et nous arrêtons la construction des buts quand nous arrivons à une liste de buts manifestement insatisfaisables (chaque but comportant une formule et sa négation).

Au début de ce chapitre (cf 1.1, 1.2, 1.3) nous présentons un système formel, procédant par réfutation, dont nous prouvons la cohérence et la complétude. Ce système peut être vu comme un calcul des séquents, avec des séquents de conséquents vides et d'ailleurs toute réfutation dans ce système peut être traduite sans difficulté en une preuve dans le calcul des séquents.
Ce système formel comporte deux règles structurelles : la règle d'échange et la règle de contraction mais aucune règle d'affaiblissement. Nous prouvons que la règle d'affaiblissement est une règle dérivée du système.

Dans le paragraphe 1.4.1, nous remplaçons la règle de contraction et la règle d'introduction du $\forall$, par une règle (C$\forall$) regroupant une contraction(C) et une introduction du $\forall$.
La complétude de ce nouveau système résulte de la complétude (prouvée) des algorithmes de démonstration que nous présentons. Je suis persuadé que cette preuve de complétude ne peut être, contrairement à celle du système avec règle de contraction, que de nature constructive : il faut montrer comment contrôler l'emploi de cette nouvelle règle.

Dans le paragraphe 1.4, nous présentons des classes d'algorithmes de démonstration automatique corrects et complets (cf 1.4.2.8) plutôt qu'un algorithme particulier. Nous séparons le plus clairement possible les règles logiques (cf 1.4.2.1) utilisées par un algorithme et le contrôle qu'il faut exercer sur l'enchaînement de règles pour que l'algorithme soit complet.

Un algorithme est complet s'il construit des dérivations équitables. Pour décrire ce que nous entendons par dérivation équitable, nous associons à une dérivation (la liste de listes de buts construite par l'algorithme) un arbre de dérivation dont la construction est anologue à celle de

l'arbre de dérivation associé à une dérivation dans une grammaire hors-contexte.

L'arbre de dérivation peut être infini, et l'ensemble des dérivations équitables est défini par une condition portant sur les branches infinies de cet arbre (cf 1.4.2.13).
Nous donnons deux moyens effectifs de produire des dérivations équitables (ces moyens sont décrits en 1.4.2.10 et prouvés corrects en 1.4.2.19).

Lorsqu'un démonstrateur automatique a pour donnée une formule fermée A non valide, il est capable **s'il s'arrête** de produire un modèle de $\neg A$. Nous avons décrit de façon précise (cf 1.4.2.18) comment exhiber ce modèle à partir de l'état final du démonstrateur.

Ce travail est né de l'insatisfaction ressentie à la lecture du livre de [J.Gallier 86] où la séparation entre règles et contrôle des règles n'est pas faite, et du livre de [M.Fitting 90] où manque la rigueur dans les preuves.
Avant d'écrire ce chapitre, j'ai écrit un prototype de démonstrateur automatique, qui m'a servi pour vérifier que les formules dont je demandais aux étudiants de prouver la validité, étaient bien valides, et pour trouver (quand le démonstrateur avait la bonne idée de se terminer pour la donnée A non valide) un modèle de la formule $\neg A$.

J'estime qu'après l'étude de ce chapitre, il est possible de réaliser sans difficultés (par un étudiant en informatique du niveau DEA) des maquettes de démonstrateurs utiles pour l'enseignement de la logique avec les caractéristiques suivantes :
(1) paramètrage du contrôle des règles, tout en obtenant des algorithmes complets
(2) pour une donnée A qui est une formule valide, le démonstrateur fournit une réfutation de $\neg A$ dans le système formel 1.3, éventuellement traduit cette réfutation en une preuve de A dans le calcul des séquents, ou dans le calcul de la déduction naturelle [G.Gentzen 55]
(3) pour une donnée A qui n'est pas valide, le démonstrateur fournit, s'il se termine, un modèle de la formule $\neg A$.

Dans ce chapitre, nous avons utilisé les seules opérations $\wedge$, $\neg$, $\forall$ pour simplifier les démonstrations et éviter les notations peu intuitives de [M.Fitting 90]. Mais il est clair que pour rendre les démonstrateurs efficaces, il faut ajouter des règles pour toutes les opérations logiques usuelles ($\Leftrightarrow$, $\vee$, $\Rightarrow$, $\exists$) ainsi que des règles dérivées (cf 3.2).

Dans la méthode des tableaux sans variables libres, les buts sont **indépendants les uns des autres** et le parallèle que nous avons fait ci-dessus avec les grammaires hors-contexte peut nous inspirer aussi dans la programmation d'un démonstrateur récursif très simple dont le schéma est décrit en pseudo-Pascal par la fonction insatisfaisable qui a pour spécification :

insatisfaisable (X) = vrai si et seulement si le but X est insatisfaisable

insatisfaisable (X) = faux si X est satisfaisable et si le calcul de la fonction se termine

```
fonction insatisfaisable (X : but) : boolean ;
var Y, Z : but ;
debut
    si X est fermée alors insatisfaisable := vrai
    sinon si X est terminale alors satisfaisable := faux
    sinon
    debut
        si X produit Y (par une des règles ¬, ∧, C∀, ¬∀)
        alors insatisfaisable := insatisfaisable (Y)
        sinon si X produit Y et Z (par la règle ¬∧) alors
            si insatisfaisable(Y) alors insatisfaisable := insatisfaisable (Z)
    fin
fin
```

1.1 Sens des formules

Nous utilisons généralement les notations de [R.Lalement 90] avec quelques divergences mineures. Nous ferons de brefs rappels de ces notations et nous renvoyons à l'ouvrage cité pour tout complément.

Notation 1.1.1 (dénotation d'une formule logique)
Rappelons ce que nous entendons par le sens d'une formule logique.
Soit Σ une signature, c'est à dire un ensemble Σ_f de symboles de fonctions et un ensemble Σ_r de symboles de relations.
Soit $\mathfrak{A}$ une Σ-algèbre (autrement dit une interprétation des symboles de fonctions et relations) de domaine A.
Soit X un ensemble de variables.
$T_\Sigma[X]$ est l'ensemble des Σ-termes dont les variables sont éléments de X et dont les symboles de fonctions sont éléments de Σ_f.
$L_\Sigma[X]$ est l'ensemble des Σ-formules logiques dont les variables sont éléments de X et dont les symboles de fonctions et de prédicats sont éléments de Σ.
Soit t un terme. $\mathfrak{A}(t)$ est une application de A^X dans A qui est le sens du terme t.
Soit α une formule. $\mathfrak{A}(\alpha)$ est une application de A^X dans $\{0,1\}$ qui est le sens de la formule α.
Soit σ une application de X dans A. σ est appelée une valuation dans le cas général et une substitution quand $A = T_\Sigma[X]$.
La valeur du terme t dans l'algèbre $\mathfrak{A}$ et pour la valuation σ est notée $\mathfrak{A}(t)\sigma$: cette valeur est élément de A.
La valeur de la formule α dans l'algèbre $\mathfrak{A}$ et pour la valuation σ est notée $\mathfrak{A}(\alpha)\sigma$: cette valeur est élément de $\{0, 1\}$.
L'algèbre $\mathfrak{A}$ est un modèle de la formule α si pour toute valuation σ, on a $\mathfrak{A}(\alpha)\sigma = 1$.

Nous considérons des systèmes formels pour les formules dont les connectives sont $\wedge$ et $\neg$ et le quantificateur $\forall$. Toutes les autres connectives et le quantificateur $\exists$ sont considérés comme étant définis à partir de $\wedge$, $\neg$, $\forall$.

Dans la suite, il ne sera plus fait mention de l'ensemble X des variables, que l'on suppose une fois pour toute comme étant un ensemble infini dénombrable.

1.2 Le principe de Smullyan

Nous présentons un résultat de logique, le principe de Smullyan, qui généralise les méthodes utilisées dans les preuves de complétude de la logique du premier ordre.

Propriété de cohérence

Soit Σ une signature et Γ une propriété des ensembles de Σ-formules fermées, c'est-à-dire un ensemble d'ensembles de Σ-formules fermées.

Γ est un propriété de cohérence (pour Σ) si Γ est fermée pour les sous-ensembles, c'est-à-dire

si $S \in \Gamma$ et $T \subseteq S$ alors $T \in \Gamma$

et si pour tout $S \in \Gamma$ les conditions suivantes sont vérifiées par les formules A et B

(a) il n'y a pas de formule atomique A tel que $A \in S$ et $\neg A \in S$

(b) si $\neg\neg A \in S$ alors $S \cup \{A\} \in \Gamma$

(c) si $A \wedge B \in S$ alors $S \cup \{A, B\} \in \Gamma$

(d) si $\neg(A \wedge B) \in S$ alors $S \cup \{\neg A\} \in \Gamma$ ou $S \cup \{\neg B\} \in \Gamma$

(e) si $\forall x A \in S$ alors pour tout terme fermé t, $S \cup \{A[x:=t]\} \in \Gamma$

(f) si $\neg\forall x A \in S$ alors pour toute constante c, qui ne figure pas dans S, $S \cup \{\neg A[x:=c]\} \in \Gamma$

Le principe de Smullyan :

Soit Σ une signature, et C un ensemble disjoint de l'ensemble des symboles de Σ et de même cardinalité que l'ensemble des Σ-formules.

Soit Σ^C la signature obtenue en ajoutant à la signature Σ, les symboles de C pris comme constantes.

Soit Γ une propriété de cohérence pour la signature Σ^C.

Si S est un ensemble de formules fermées, ne comportant pas les nouvelles constantes et si $S \in \Gamma$ alors S a un modèle dont le domaine est l'ensemble des Σ^C-termes fermés.

preuve : cf [P.B.Andrews 86]

1.3 Un système formel pour les tableaux sans variables libres.

Nous décrivons un système formel, très analogue au système des tableaux présenté par [M.Fitting 90], qui permet de dériver une liste de formules **fermées** à partir d'un ensemble de listes de formules **fermées**.

Nous désignons les formules par les lettres A, B, C et les listes de formules par les lettres X, Y, Z. A une liste de formules A_1, A_2, ...A_n on associe la formule $A_1 \wedge A_2 \wedge$ ...A_n et à la liste vide on associe la formule faux. Dans la suite, nous confondons une liste de formules et la formule qui est associée à la liste.

Les axiomes sont les listes de formules de la forme :

A, $\neg$A, X

Remarquons que les axiomes sont insatisfaisables, ce qui est normal, car le système procède par réfutation.
Les règles logiques permettent de dériver une liste de formules fermées à partir d'une ou deux listes de formules fermées :

$$\frac{A, X}{\neg\neg A, X}(\neg\neg) \qquad \frac{A, B, X}{A \wedge B, X}(\wedge) \qquad \frac{\neg A, X \quad \neg B, X}{\neg(A \wedge B), X}(\neg\wedge)$$

$$\frac{A[x:=t], X}{\forall x A, X}(\forall)\ t \text{ est un terme fermé} \qquad \frac{\neg A[x:=c], X}{\neg\forall x A, X}(\neg\forall)\ c \text{ est une constante qui ne doit pas figurer dans } \forall x A, X$$

Les règles structurelles permettent d'échanger deux formules et de fusionner deux formules identiques.

$$\frac{X, A, B, Y}{X, B, A, Y}(\text{échange}) \qquad \frac{A, A, X}{A, X}(\text{contraction})$$

On remarque qu'il n'y a pas de règle d'affaiblissement. Le lemme 1.3.2 montre que cette règle n'est pas indispensable dans ce système formel.

Soit Γ un ensemble de listes de formules et X une liste de formules.
X est réfutable à partir des hypothèses Γ, ce qu'on note $\Gamma \vdash X$, s'il y a une liste de listes de formules $Y_1, \ldots Y_n$ telle que $X = Y_n$ et dans laquelle toute liste Y_i de formules est un axiome, un liste élément de Γ ou une liste obtenue par application d'une règle à des listes Y_j où $j < i$.
Lorsque $\Gamma = \emptyset$, on dit que X est réfutable et on note cette situation par $\vdash X$.

Définition 1.3.1
Soient S et T deux listes de formules. S est contenu dans T ($S \subseteq T$) si toute formule de la liste S est une formule de la liste T.

Lemme 1.3.2
Supposons que la signature utilisée comporte une infinité dénombrable de constantes.
Si $\vdash S$ et si la liste S est contenue dans la liste T alors $\vdash T$.

preuve :

Supposons que S est réfutable et que S est contenu dans T.

Il existe une liste X, obtenue grâce à des règles de contraction et d'échange appliquées à S, qui comporte toutes les formules de S sans répétition.

Puisque S est réfutable, alors **X est réfutable.**

Puisque X est contenue dans T et que X est sans répétitions, il y a une liste U permutation de T et une liste V tels que:

U = X, V

Puisque T est obtenue par des règles d'échanges à partir de U, nous avons :

(1) $U \vdash T$

Pour obtenir à partir de la réfutation de X, une réfutation de U (donc de T), il ne suffit pas d'ajouter V à droite de toutes les listes de formules de la réfutation de X. En effet cette transformation ne préserve pas les conditions d'application de la règle $(\neg\forall)$.

Soit $X_1, \ldots X_n$ la réfutation de X, montrons que :

(2) il y a une réfutation de X dans laquelle les constantes c introduites par la règle $(\neg\forall)$ ne sont pas des constantes de V.

Appelons renommage des constantes, une application (non necessairement bijective) de l'ensemble des constantes dans lui-même.

Un renommage est étendu aux formules fermées et aux listes de formules fermées.

Le domaine d'un renommage s est l'ensemble des constantes c telles que $s(c) \neq c$.

Pour tout entier i ($i \geq 1$), s_i est un renommage des constantes.

Le renommage s_1 est l'identité sur l'ensemble des constantes.

Par récurrence on montre que :

(3) $s_i(X_1), \ldots s_i(X_i)$ est une réfutation de X_i, le domaine de s_i est inclus dans l'ensemble des constantes de $X_1, \ldots X_{i-1}$ et les constantes choisies à chaque application de la règle $(\neg\forall)$ dans la réfutation $s_i(X_1), \ldots s_i(X_i)$ ne sont pas des constantes de V.

La propriété (3) est vraie pour i = 1. Prouvons la pour i+1.

Si X_{i+1} est obtenu par une quelconque des règles sauf $(\neg\forall)$, on pose

$s_{i+1} = s_i$ et il est clair alors que la propriété (3) est vérifiée pour i+1.

Si $X_{i+1} = \neg\forall xA, Y$ et que X_{i+1} a été obtenu à partir de $X_j = \neg A[x:=c], Y$ avec $j \leq i$,

puisque par hypothèse de récurrence, s_i ne modifie pas les constantes de X_k (où $k \geq i$), et puisque A et Y apparaissent dans X_{i+1}, nous avons : $s_i(X_j) = \neg A[x:= s_i(c)], Y$.

Si la constante $s_i(c)$ ne figure ni dans V, ni dans X_{i+1} alors on pose

$s_{i+1} = s_i$ et il est clair alors que la propriété (3) est vérifiée pour i+1.

Si la constante $s_i(c)$ figure dans V ou dans X_{i+1}, on choisit une nouvelle constante d, qui ne figure ni dans V ni dans $s_i(X_1)$, ... $s_i(X_i)$ ni dans X_{i+1} et l'on pose $s_{i+1} = s_i[c:=d]$.

Puisque c figure dans X_j avec $j \leq i$ et pas dans X_{i+1} (vu la condition d'application de la règle $(\neg\forall)$, le domaine de s_{i+1} est inclus dans l'ensemble des constantes de $X_1,...X_i$.

Puisque d ne figure pas ni dans $s_i(X_1)$, ... $s_i(X_i)$, alors $s_{i+1}(X_1)$, ... $s_{i+1}(X_i)$ est une réfutation.

Puisque $s_{i+1}(X_j) = \neg A[x:= d], Y$ et que $s_{i+1}(X_{i+1}) = X_{i+1} = \neg\forall xA, Y$ il en résulte que $s_{i+1}(X_1)$, ... $s_{i+1}(X_{i+1})$ est une réfutation de X_{i+1}.

A cause du choix de la constante d, on peut aussi affirmer que les constantes c choisies à chaque application de la règle $(\neg\forall)$ dans la réfutation $s_{i+1}(X_1)$, ... $s_{i+1}(X_{i+1})$ ne sont pas des constantes de V.

Ce qui termine la preuve par récurrence de la propriété (3).

La propriété (2) est une conséquence immédiate de la propriété (3).

D'après la propriété (2), il y a une réfutation $Y_1,...Y_n$ de X dans laquelle les constantes c introduites par la règle $(\neg\forall)$ ne sont pas des constantes de V.

Posons $U_i = Y_i, V$, il est clair que $U_1, ...U_n$ est une réfutation de X,V = U.

Par suite U est réfutable et puisque d'après (1) $U \vdash T$ nous avons $\vdash T$.

On note que dès l'énoncé du théorème ci-dessous, on a identifié une liste de formules et la formule associée à la liste. Si l'on tient à éviter cette identification, il faudrait dire :
la formule associée à X est insatisfaisable si et seulement si X est réfutable.

Théorème 1.3.3 (complétude du système formel)

Soit X une liste de formules fermées.

X est insatisfaisable si et seulement si X est réfutable.

preuve :

(1) Si X est réfutable alors X est insatisfaisable

(1.1) Si une liste de formules en bas d'une règle est satisfaisable, il en est de même pour la liste en haut de la règle (pour les règles $\neg\neg$, $\wedge$, $\forall$, $\neg\forall$, échange, contraction) et pour l'une des deux listes en haut de la règle $\neg\wedge$.

Prenons l'exemple de la règle dont le tableau du bas est $\neg\forall xA, X$.

Supposons que l'algèbre $\mathfrak{A}$ et la valuation s satisfont toutes les formules de la liste $\neg\forall xA, X$.

Il y a un élément a du domaine de $\mathfrak{A}$ tel que $\mathfrak{A}(\neg A)s[x:=a] = 1$.

Soit $\mathfrak{A}'$ l'algèbre identique à $\mathfrak{A}$, sauf pour la constante c qui a dans $\mathfrak{A}'$ la valeur a.

Puisque c ne figure ni dans la formule A, ni dans Y, nous avons :

pour toute formule B de la liste Y, $\mathfrak{A}'(B)s = \mathfrak{A}(B)s = 1$

$\mathfrak{A}'(\neg A[x:=c])s = \mathfrak{A}'(\neg A)s[x:=a] = \mathfrak{A}(\neg A)s[x:=a] = 1$

Par suite $\mathfrak{A}'$ satisfait $\neg A[x:=c]$, Y.

Les axiomes étant insatisfaisables, on déduit de facilement de (1.1) que si X est réfutable alors X est insatisfaisable.

(2) Si X est insatisfaisable alors X est réfutable.

Sans perte de généralité, nous pouvons supposer que la signature Σ des formules de la liste X est finie.

Soit C un ensemble infini dénombrable disjoint de l'ensemble des symboles de Σ.

Soit Σ^C la signature obtenue en ajoutant à la signature Σ, les symboles de C pris comme constantes.

Soit Γ la classe d'ensembles de formules définie par :

$\Gamma = \{S \mid$ aucune liste de formules de S n'est réfutable$\}$

Pour prouver (2), il suffit de montrer que Γ est une propriété de cohérence (cf 1.2) pour Σ^C.

En effet supposons que X n'est pas réfutable.

L'ensemble des formules de X est élément de Γ. En effet, si une liste de formules de X était réfutable, d'après le lemme 1.3.2, X serait réfutable.

D'après le principe de Smullyan (cf 1.2), il en résulte que X a un modèle, donc que X est satisfaisable.

Montrons que Γ est une propriété de cohérence.

Du lemme 1.3.2, il résulte que Γ est fermée pour les sous ensembles.

La condition (a) est triviale.

Vérifions la condition (b).

Supposons que $\neg\neg A \in S$ et $S \in \Gamma$ mais que $S \cup \{A\} \notin \Gamma$.

Il y a une sous-liste X de $S \cup \{A\}$ qui est réfutable.

Soit A,Y une liste **sans répétition** composée des mêmes formules que A,X (donc les formules de la liste Y sont éléments de S).

D'après le lemme 1.3.2 la liste A,Y est réfutable donc la liste $\neg\neg A$, Yest réfutable grâce à la règle $(\neg\neg)$.

C'est impossible puisque $\neg\neg A$,Y est une liste de formule de S.

Vérifions la condition (c)

Supposons que $A \wedge B \in S$ et $S \in \Gamma$ mais que $S \cup \{A, B\} \notin \Gamma$.

Il y a une sous-liste X de $S \cup \{A, B\}$ qui est réfutable.

Soit A, B,Y une liste **sans répétition** composée des mêmes formules que A, B, X (donc les formules de la liste Y sont éléments de S).

D'après le lemme 1.3.2 la liste A, B,Y est réfutable donc la liste $A \wedge B$, Y est réfutable grâce à la règle ($\wedge$).

C'est impossible puisque $A \wedge B$,Y est une liste de formules de S.

Vérifions la condition (d)

Supposons que $\neg(A \wedge B) \in S$ et $S \in \Gamma$ mais que $S \cup \{\neg A\} \notin \Gamma$ et $S \cup \{\neg B\} \notin \Gamma$

Il y a une sous-liste X de $S \cup \{\neg A\}$ qui est réfutable et une sous-liste Y de $S \cup \{\neg B\}$ qui est réfutable.

Soit $\neg A$, X' une liste sans répétition composée des mêmes formules que $\neg A$, X (donc les formules de la liste X' sont éléments de S).

Soit $\neg B$, Y' une liste sans répétition composée des mêmes formules que $\neg B$, Y (donc les formules de la liste Y' sont éléments de S).

D'après le lemme 1.3.2 chacune des listes $\neg A$, X', Y' et $\neg B$, X', Y' est réfutable donc la liste $\neg(A \wedge B)$, X', Y' est réfutable grâce à la règle ($\neg\wedge$).

C'est impossible puisque $\neg(A \wedge B)$, X', Y' est une liste de formules de S.

On peut vérifier de façon analogue les conditions (e) et (f).

1.4 Démonstration automatique avec les tableaux sans variables libres

1.4.1 Système formel adapté à la démonstration automatique

Lorsqu'on fait une démonstration (automatique ou non), on part généralement de la formule à prouver et on trouve la preuve de la formule avec des règles ainsi formulées :

pour prouver A, il suffit de prouver les sous-buts B_1 et ....B_n

C'est ainsi que sont par exemple rédigées les tactiques de MacLogic [R.Dyckhoff 90] ou les règles du système Nuprl [R.L.Constable 86].

Cette manière de voir correspond à une lecture de bas en haut des règles précédentes (cf 1.3) où au lieu de prouver, on cherche des réfutations :

pour réfuter $\neg(A \wedge B)$, X il suffit de réfuter $\neg A$, X et $\neg B$, X.

On voit immédiatement qu'en retournant les règles, la règle de contraction pose un problème puisqu'elle « réduit » la réfutation de A, X à celle de A, A, X .

Aussi, la règle de contraction et la règle ($\forall$) sont **remplacées** par la règle dérivée obtenue par application d'une contraction et d'une règle ($\forall$).

$$\frac{A[x:=t],\ \forall x A,\ X}{\forall x A,\ X}\ (C\forall)$$ où t est un terme fermé

Le système formel obtenu avec ce remplacement est aussi complet [J.Gallier 86].

1.4.2 Démonstrateur automatique

Les formules que nous traitons sont des formules fermées, sans symboles fonctionnels (sauf des constantes).
Cette restriction a deux raisons :
(1) sans algorithme d'unification, il est impossible d'obtenir un démonstrateur efficace admettant des symboles fonctionnels (le chapitre 3 remédie à ce problème)
(2) à cause de l'absence de symboles fonctionnels, quand un démonstrateur se termine avec pour donnée une formule fermée A non valide, il est possible de produire un modèle fini de $\neg A$.

Les constantes sont toutes éléments de la liste c_i (où $i \in \mathbb{N} - \{0\}$).
Les formules universelles peuvent être accompagnées d'un indice entier :
$\forall_n xA$ où $n \in \mathbb{N} - \{0\}$ signifie qu'on a déjà appliqué n fois la règle (C$\forall$) présentée ci-dessus en créant les instances $A[x:=c_1]$, ...$A[x:=c_n]$.

Les listes de formules sont accompagnées d'un exposant entier : X^n où $n \in \mathbb{N}$.

Le démonstrateur est présenté sous forme de règles appliquées à des listes de listes de formules désignées par les lettres Γ, Δ, Π, Ψ.
Les listes de formules sont comme précédemment (cf 1.3) interprétées comme des conjonctions.

Définition 1.4.2.1
Soient $i, k \in \mathbb{N}$.
Γ produit Δ dans les cas suivants :

($\neg\neg$) $\Gamma = \Pi, [X, \neg\neg A, Y]^i, \Psi$ $\Delta = \Pi, [X, A, Y]^i, \Psi$
$\neg\neg A$ est la formule dérivée par la règle et A est la formule produite par la règle

($\wedge$) $\Gamma = \Pi, [X, A \wedge B, Y]^i, \Psi$ $\Delta = \Pi, [X, A, B, Y]^i, \Psi$
$A \wedge B$ est la formule dérivée par la règle, A et B sont les formules produites par la règle

($\neg\wedge$) $\Gamma = \Pi, [X, \neg(A \wedge B), Y]^i, \Psi$ $\Delta = \Pi, [X, \neg A, Y]^i, [X, \neg B, Y]^i, \Psi$
$\neg(A \wedge B)$ est la formule dérivée par la règle, $\neg A$ et $\neg B$ sont les formules produites par la règle

($\neg\forall$) $\Gamma = \Pi, [X, \neg\forall xA, Y]^i, \Psi$ $\Delta = \Pi, [X, \neg A[x:=c_{i+1}], Y]^{i+1}, \Psi$
$\neg\forall xA$ est la formule dérivée par la règle, $\neg A[x:=c_{i+1}]$ est la formule produite par la règle

(C$\forall$) Si la formule universelle n'est pas indicée

$\Gamma = \Pi, [X, \forall xA, Y]^0, \Psi$ $\Delta = \Pi, [X, \forall_1 xA, A[x:=c_1], Y]^1, \Psi$

$\forall xA$ est la formule dérivée par la règle, $\forall_1 xA$, $A[x:=c_1]$ sont les formules produites par la règle

avec $i > 0$

$\Gamma = \Pi, [X, \forall xA, Y]^i, \Psi$ $\Delta = \Pi, [X, \forall_1 xA, A[x:=c_1], Y]^i, \Psi$

$\forall xA$ est la formule dérivée par la règle, $\forall_1 xA$, $A[x:=c_1]$ sont les formules produites par la règle

Si la formule universelle est déjà indicée et si $k < i$

$\Gamma = \Pi, [X, \forall_k xA, Y]^i, \Psi$ $\Delta = \Pi, [X, \forall_{k+1} xA, A[x:=c_{k+1}], Y]^i, \Psi$

$\forall_k xA$ est la formule dérivée par la règle, $\forall_{k+1} xA$, $A[x:=c_{k+1}]$ sont les formules produites par la règle

Soit $Z = [X, A, Y]^i$ une liste de formules et A une formule. La formule A n'est pas dérivable (pour la liste Z) si la formule A est un littéral ou une formule universelle d'indice i.

Définition 1.4.2.3

Une liste de formules est fermée si elle comprend une formule et sa négation.
Une liste de listes de formules est fermée si toutes ses listes de formules sont fermées.

Définition 1.4.2.4

Soit A une formule fermée sans symboles de fonctions et comportant les seules constantes $c_1, \ldots c_k$ où $k \geq 0$.
Un dérivation de A est une suite $\Gamma_0, \Gamma_1, \ldots$ de listes de listes de formules telle que

$\Gamma_0 = [[\neg A]^k]$

pour tout $i \in \mathbb{N}$, Γ_i produit Γ_{i+1}

Nous vérifions immédiatement qu'une dérivation de A, n'est autre qu'une réfutaion de $\neg A$ dans le système formel 1.3

Lemme 1.4.2.5

Soit A une formule fermée sans symboles de fonctions, non indicée et comportant les seules constantes $c_1, \ldots c_k$ où $k \geq 0$.
S'il y a une dérivation $\Gamma_0, \Gamma_1, \ldots$ de A telle qu'il existe i avec Γ_i fermée, alors $\neg A$ est réfutable dans le système 1.3 et A est valide.

preuve :

Par récurrence sur i on prouve que si la liste de formules X^k est élément de Γ_i toutes les constantes

apparaissant dans les formules de X^k sont dans l'ensemble de constantes $c_1, ...c_k$.
Par suite lorsqu'on applique la règle de dérivation

$(\neg\forall)$ $\qquad \Gamma_i = \Pi, [X, \neg\forall xA, Y]^k, \Psi \qquad \Gamma_{i+1} = \Pi, [X, \neg A[x:=c_{k+1}], Y]^{k+1}, \Psi$

la constante c_{k+1} n'apparait pas dans la liste de formules $[\neg\forall xA, X]^k$.

Puisque les règles (1.4.2.1) sont (à une permutation près des listes de formules) les règles du système formel 1.3 et que la contrainte de la règle $(\neg\forall)$ est respectée, il en résulte que si l'algorithme s'arrête avec Γ_k fermée (c'est à dire une liste d'axiomes du système formel 1.3), il y a une réfutation de la formule $\neg A$ dans le système formel 1.3.

D'après le théorème 1.3.3, il en résulte que $\neg A$ est insatisfaisable, donc que A est valide.

Exemple 1.4.2.6

Soit A la formule $\neg(\forall x(P(x) \wedge Q(x)) \wedge \neg (\forall xP(x) \wedge \forall xQ(x)))$.
Nous présentons une dérivation de A.

$[[\neg \neg(\forall x(P(x) \wedge Q(x)) \wedge \neg (\forall xP(x) \wedge \forall xQ(x)))]^0]$

$[[(\forall x(P(x) \wedge Q(x)) \wedge \neg (\forall xP(x) \wedge \forall xQ(x)))]^0]$

$[[\forall x(P(x) \wedge Q(x)), \neg (\forall xP(x) \wedge \forall xQ(x))]^0]$

$[[\forall x(P(x) \wedge Q(x)), \neg\forall xP(x)]^0, [\forall x(P(x) \wedge Q(x)), \neg\forall xQ(x)]^0]$

$[[\forall x(P(x) \wedge Q(x)), \neg P(c_1)]^1, [\forall x(P(x) \wedge Q(x)), \neg\forall xQ(x)]^0]$

$[[\forall_1 x(P(x) \wedge Q(x)), P(c_1) \wedge Q(c_1), \neg P(c_1)]^1, [\forall x(P(x) \wedge Q(x)), \neg\forall xQ(x)]^0]$

$[[\forall_1 x(P(x) \wedge Q(x)), P(c_1) , Q(c_1), \neg P(c_1)]^1, [\forall x(P(x) \wedge Q(x)), \neg\forall xQ(x)]^0]$

nous arrêtons ici avec la première liste de formules fermées, car il est évident qu'il est est possible de la même manière de fermer la deuxième liste.
Par conséquent la formule A est valide.

Définition 1.4.2.7

Une liste de formules est terminale si elle n'est pas fermée et si elle ne comprend que des formules non dérivables.
Une liste de listes de formules est terminale si au moins une de ses listes de formules est terminale.

Avant d'écrire des algorithmes de démonstration automatique, nous précisons les spécifications auxquelles ces algorithmes doivent obéir.
Nous avons repris à notre compte la terminologie de [G.S.Boolos, R.C.Jeffrey 74] p124.

Définition 1.4.2.8
Soit P un algorithme de démonstration automatique pour la logique du premier ordre.
L'algorithme P est correct si, pour toute formule fermée A, il vérifie :
lorsqu'il se termine en déclarant que A est valide, alors A est effectivement valide.
L'algorithme P est complet si, pour toute formule fermée A, il vérifie :
lorsque A valide, il se termine en déclarant que A est valide et lorsque A n'est pas valide, il se termine en déclarant que A n'est pas valide ou il ne se termine pas.

Plutôt que de définir un algorithme de démonstration automatique, nous présentons une **classe d'algorithmes**. Tout algorithme de cette classe construit une **dérivation équitable**. Informellement une dérivation équitable est une dérivation dans laquelle, toute formule élément d'une liste non fermée est dérivée.

Classe d'algorithmes 1.4.2.9
Soit A une formule fermée sans symboles de fonctions, non indicée et comportant les seules constantes $c_1, ...c_k$ où $k \geq 0$.
On construit une dérivation **équitable** $\Gamma_0, \Gamma_1, ...$ de la formule A et on s'arrête dès que la liste Γ_i est fermée ou terminale.
Si lors de l'arrêt Γ_i est fermée, on indique que A est valide (on peut alors fournir une réfutation de $\neg A$ dans le système 1.3) sinon lors de l'arrêt on indique que A n'est pas valide (on peut alors fournir un modèle satisfaisant $\neg A$).

Remarquons immédiatement que tout algorithme de cette classe est correct. En effet si un algorithme de cette classe se termine en indiquant que A est valide alors il y une dérivation de A qui termine par une liste Γ fermée de liste de formules.
Donc d'après le lemme 1.4.2.5, A est effectivement valide.

Avant de définir formellement ce qu'est une dérivation équitable, nous donnons un moyen de **construire** une dérivation équitable, pour montrer au lecteur qu'il est facile de fabriquer une telle dérivation (même si la définition de l'équitabilité n'est pas tout à fait simple) et pour essayer de le convaincre qu'il existe bien un algorithme dans la classe d'algorithmes ci-dessus.

Construction d'une dérivation équitable 1.4.2.10

On suppose que l'on peut marquer les formules.

Soit A une formule fermée sans symboles de fonctions, non indicée et comportant les seules constantes $c_1, ...c_k$ où $k \geq 0$.

Soit $\Gamma_0, ...\Gamma_i$ la partie déjà construite d'une dérivation de A. Supposons que Γ_i n'est ni fermée ni terminale. La liste Γ_{i+1} est construite en quatre étapes successives :

(1) Choisir dans Γ_i une liste Z non fermée de formules.

Puisque Γ_i n'est pas fermée, un tel choix est possible.

(2) Si toutes les formules de Z sont marquées ou ne sont pas dérivables alors enlever les marques.

(3) Choisir dans la liste Z une formule A qui est dérivable et non marquée.

Puisque Γ_i n'est pas terminale, Z comprend une formule dérivable, donc après l'étape (2) il y a dans Z une formule dérivable et non marquée et le choix de A est possible.

(4) Appliquer la règle de dérivation (cf 1.4.2.1) correspondant à A, et marquer les formules produites par la règle.

Remarque : Ce n'est évidemment pas la seule manière de produire de produire une dérivation équitable. Nous en indiquons immédiatement une autre, marquer seulement la formule universelle produite dans le cas de l'application de la règle (C∀) c'est-à-dire remplacer l'étape (4) par

(4') Appliquer la règle de dérivation (cf 1.4.2.1) correspondant à A, et marquer la formule universelle produite quand cette règle est la règle (C∀).

Définition 1.4.2.11 (arbre de dérivation)

Un domaine d'arbre est un sous ensemble D de $(\mathbb{N}-\{0\})^*$ fermé pour les préfixes et tel que si $u.(i+1) \in D$ et si $i > 0$ alors $u.i \in D$.

Un arbre de dérivation est une application d'un domaine d'arbre dans l'ensemble des listes de formules.

Soit T un arbre de dérivation et u un élément de son domaine : u est appelé un sommet de l'arbre et T(u) est l'étiquette du sommet u.

Soit A une formule fermée sans symboles de fonctions, non indicée et comportant les seules constantes $c_1, ...c_k$ où $k \geq 0$.

A une dérivation de A, on associe un arbre qui montre la structure de la dérivation.

Soit $\Gamma_0, \Gamma_1, ...$ une dérivation de A. On associe une suite d'arbres T_i à cette dérivation, cette association étant définie par récurrence sur i.

A $\Gamma_0 = [[\neg A]^k]$ on associe l'arbre T_0 ayant un seul sommet d'étiquette $[\neg A]^k$.

Supposons que T_i est un arbre associé à la dérivation $\Gamma_0, ...\Gamma_i$ et que la liste des étiquettes des

feuilles de T_i est Γ_i.
Si Γ_{i+1} est obtenu en remplaçant dans Γ_i la k-ième liste de Γ_i par la liste Y, alors l'arbre T_{i+1} est obtenu en ajoutant à la k-ième feuille u de T_i, le successeur u1 avec l'étiquette Y.
Si Γ_{i+1} est obtenu en remplaçant dans Γ_i la k-ième liste de Γ_i par les listes Y et Z, alors l'arbre T_{i+1} est obtenu en ajoutant à la k-ième feuille u de T_i, les deux successeurs u1 et u2 avec les étiquettes respectives Y et Z.

A la dérivation finie $\Gamma_0, \ldots \Gamma_n$ on associe l'arbre $T = T_n$.
A la dérivation infinie $\Gamma_0, \Gamma_1, \ldots$ on associe l'arbre $T = \cup\{ T_i \mid i \in \mathbb{N} \}$

Exemple 1.4.2.12
A la dérivation présentée à l'exemple 1.4.2.6 correspond l'arbre ci-dessous dont la structure est indiquée par la disposition et le nom des sommets.

$(\varepsilon)\ [\ \neg\neg(\forall x(\ P(x) \wedge Q(x)\ )\ \wedge \neg\ (\forall x P(x) \wedge \forall x Q(x)\ )\ )\ ]^0$
$(1)\ [\ (\forall x(\ P(x) \wedge Q(x)\ )\ \wedge \neg\ (\forall x P(x) \wedge \forall x Q(x)\ )\ )\ ]^0$
$(11)\ [\ \forall x(\ P(x) \wedge Q(x)\ ), \neg\ (\forall x P(x) \wedge \forall x Q(x)\ )\ ]^0$
$(111)\ [\ \forall x(\ P(x) \wedge Q(x)\ ), \neg\forall x P(x)\ ]^0$
$(1111)\ [\ \forall x(\ P(x) \wedge Q(x)\ ), \neg P(c_1)\ ]^1$
$(11111)\ [\ \forall_1 x(\ P(x) \wedge Q(x)\ ), P(c_1) \wedge Q(c_1), \neg P(c_1)\ ]^1$
$(112)\ [\ \forall x(\ P(x) \wedge Q(x)\ ), \neg\forall x Q(x)\ ]^0$

A l'aide de la définition d'un arbre de dérivation, il nous est possible de définir une dérivation équitable.

Définition 1.4.2.13 (dérivation équitable)
Soit T un arbre de dérivation.
Une **branche** est un chemin issu de la racine de l'arbre, soit ayant pour extrémité une feuille, soit infini.
L'ensemble des formules d'une branche est l'ensemble des formules qui sont éléments des listes étiquettes des sommets de la branche.
Une formule est sur une branche si elle est élément de l'ensemble des formules de la branche.

Une dérivation est **équitable** lorsque l'arbre T qui lui est associé vérifie les propriétés suivantes :
(1) Aucun sommet d'une branche infinie n'est étiquetée par une étiquette fermée
(2) L'ensemble H des formules de toute branche b infinie vérifie les conditions suivantes pour toutes formules A et B et toute variable x :
(a) si $\neg\neg A \in H$ alors $A \in H$

(b) si $A \wedge B \in H$ alors $A \in H$ et $B \in H$

(c) si $\neg(A \wedge B) \in H$ alors $\neg A \in H$ ou $\neg B \in H$

(e) si $\neg\forall xA \in H$ alors il existe un entier i tel que $0 < i$ et $\neg A[x:= c_i] \in H$

(f) si $\forall xA \in H$ alors pour tout entier i tel que $0 < i$ on a $A[x:= c_i] \in H$

Nous voyons dans la suite que l'ensemble des formules d'une branche infinie d'un arbre associé à une dérivation équitable est un ensemble d'Hintikka, donc a un modèle.

Définition 1.4.2.14 (ensemble d'Hintikka)

Soit Σ une signature. Un ensemble H de Σ-formules fermées est un ensemble d'Hintikka (relativement à la signature Σ), si H vérifie les propriétés (1) et (2) suivantes :

(1) Il n'existe pas de formule atomique A telle que $A \in H$ et $\neg A \in H$.

(2) pour toutes les formules A et B et toute variable x :

(a) si $\neg\neg A \in H$ alors $A \in H$

(b) si $A \wedge B \in H$ alors $A \in H$ et $B \in H$

(c) si $\neg(A \wedge B) \in H$ alors $\neg A \in H$ ou $\neg B \in H$

(d) si $\neg\forall xA \in H$ alors il existe un Σ-terme fermé t tel que $\neg A[x:=t] \in H$

(e) si $\forall xA \in H$ alors pour tout Σ-terme fermé t, on a $A[x:=t] \in H$

Lemme 1.4.2.15 (lemme d'Hintikka)

Soit Σ une signature comportant au moins une constante (il y a donc au moins un terme fermé). Tout ensemble H de formules qui est un ensemble d'Hintikka pour cette signature a un modèle de Herbrand, c'est-à dire un modèle dont le domaine est l'ensemble des Σ-termes fermés et dont la base est l'ensemble des formules atomiques éléments de H.

preuve : cf [M.Fitting 90] p 112.

Lemme 1.4.2.16

Soit T un arbre de dérivation et b une branche finie dont le dernier sommet est étiqueté par la liste terminale de formules X^n. L'ensemble des formules de la branche b a un modèle ayant pour domaine $c_1, \ldots c_n$ (si $n > 0$) et c_1 (si $n = 0$) et pour base l'ensemble des formules atomiques de X^n.

preuve :

Soit H l'ensemble des formules de la branche b.

(1) L'ensemble des constantes apparaissant dans H est $c_1, \ldots c_n$.

En effet dans la preuve du lemme 1.4.2.5 nous avons déjà remarqué que :

(a) les seules constantes apparaissant dans les formules de X^n sont $c_1, \ldots c_n$.

Il est facile de vérifier que :

(b) si u est sommet d'une branche et v son successeur sur la branche et si les étiquettes respectives de u et v sont X^j et X^k alors $j \leq k$.

Des propriétés (a) et (b), il resulte que les seules constantes apparaissant dans les formules de la branche b sont $c_1, ...c_n$.

(2) L'ensemble H vérifie pour toute formule A que si $\neg\neg A \in H$ alors $A \in H$.

Supposons que $\neg\neg A \in H$.

Puisque $\neg\neg A$ figure sur la branche b, sans être élément de la liste X^n, c'est $\neg\neg A$ a été dérivée sur cette branche donc $A \in H$.

(3) L'ensemble H vérifie pour toutes formules A et B

si $A \wedge B \in H$ alors $A \in H$ et $B \in H$

si $\neg(A \wedge B) \in H$ alors $\neg A \in H$ ou $\neg B \in H$

La preuve de (3) est analogue à celle de (2).

(4) L'ensemble H vérifie pour toute formule A et toute variable x

si $\neg\forall xA \in H$ alors il existe un entier i tel que $0 < i \leq n$ et $\neg A[x:= c_i] \in H$

Supposons que $\neg\forall xA \in H$.

Puisque $\neg\forall xA$ figure sur la branche b, sans être élément de la liste X^n, c'est $\neg\forall xA$ a été dérivée sur cette branche donc $\neg A[x:= c_i] \in H$. D'après (1) nous avons $0 < i \leq n$.

(5) L'ensemble H vérifie pour toute formule A et toute variable x

si $\forall xA \in H$ alors pour tout entier i tel que $0 < i \leq n$, on a $A[x:= c_i] \in H$

Supposons que $\forall xA \in H$.

Il est clair que $\forall_n xA$ est élément de la liste X^n, car, si une formule universelle est élément de l'étiquette d'un sommet, elle est élément de toutes les étiquettes des descendants de ce sommet donc de X^n et d'autre part la liste X^n étant terminale, toute formule universelle qui apparait dans X^n est d'indice n.

Soit $Y_1, ...Y_p$ la suite des étiquettes de la branche b avec $Y_p = X^n$.

Par récurrence sur k, on prouve facilement que si $\forall_q xA$ est élément de Y_k alors $A[x:=c_1]$, ... $A[x:=c_q]$ sont des formules éléments des listes $Y_1, ...Y_k$.

De ce que $\forall_n xA$ est élément de la liste Y_p, on en déduit que $A[x:=c_1]$, ... $A[x:=c_n]$ sont des formules éléments des listes $Y_1, ...Y_p$ donc de l'ensemble H.

(6) Il n'y a pas dans H une formule atomique et sa négation.

Supposons le contraire. Il y a une formule atomique A telle que A et $\neg A$ sont éléments de H.

Puisque les litteraux ne sont pas dérivées, A et $\neg A$ sont éléments de la liste X^n.

C'est impossible, puisque la liste X^n est terminale donc non fermée.

Supposons que $n > 0$.
Les propriétés (1), ...(6) prouvent que H est un ensemble d'Hintikka pour une signature dont les symboles de fonction sont les constantes $c_1, ...c_n$.
Donc d'après le lemme d'Hintikka, H a un modèle ayant pour domaine $c_1, ...c_n$ et pour base l'ensemble des formules atomiques de H.
Puisque les formules atomiques ne sont pas dérivées, l'ensemble des formules atomiques de H est l'ensemble des formules de X^n.

Supposons que $n = 0$.
Il est évident que sur la branche, il ne peut y avoir de formules quantifiées. Par suite, l'ensemble H vérifie trivialement les propriétés (4) et (5).
Donc l'énoncé du lemme est encore vérifié pour $n = 0$.

Lemme 1.4.2.17
Soit T un arbre associé à une dérivation infinie et équitable. Pour toute branche infinie de l'arbre T, l'ensemble H des formules de la branche a un modèle de domaine $\{c_i \mid i \in \mathbb{N} - \{0\}\}$.

preuve :
Soit b une branche infinie de l'arbre T et H l'ensemble des formules de la branche b.
Soit Σ la signature des formules de H. Les seuls symboles de fonction de Σ sont les constantes c_i où $i \in \mathbb{N} - \{0\}$. Donc tout Σ-terme fermé est une de ces constantes.
Par suite pour vérifier que H est un ensemble d'Hintikka, il suffit de prouver qu'il n'y a pas dans H une formule atomique et sa négation.
Supposons au contraire qu'il y a une formule atomique A telle que $\neg A \in H$ et $A \in H$.
Il existe alors deux sommets u et v de la branche b tels que A soit dans la liste étiquette du sommet u et que $\neg A$ soit dans la liste étiquette du sommet v.
Soit w le sommet tel que si u est descendant de v alors $w = u$ sinon $w = v$.
Puisque $\neg A$ et A sont des littéraux donc des formules non dérivables de la branche b, il est immédiat de vérifier que A et $\neg A$ sont dans la liste étiquette du sommet w et par suite que la branche b ne vérifie pas la condition (1) de la définition 1.4.2.13, donc que contrairement aux hypothèses T n'est pas associé à une dérivation équitable.
Par suite H est un ensemble d'Hintikka, donc possède un modèle de domaine $\{c_i \mid i \in \mathbb{N} - \{0\}\}$.

Soit P un algorithme de la classe d'algorithmes 1.4.2.9.
Nous montrons maintenant deux propriétés :
(1) L'algorithme P est correct et complet
(2) Lorsque la formule A n'est pas valide et lorsque P se termine, il est possible de construire un modèle de $\neg A$.

Lemme 1.4.2.18

Tout algorithme de la classe 1.4.2.9 est correct et complet.

Si un algorithme de cette classe se termine en ayant construit une dérivation de A dont la dernière étape Γ est terminale et si X^n est une liste terminale de formules élément de Γ, alors $\neg A$ a un modèle qui a pour domaine $c_1, \ldots c_n$ (si $n > 0$) et c_1 (si $n = 0$) et pour base l'ensemble des formules atomiques de X^n.

preuve :

Soit P un algorithme de cette classe.

Nous avons déjà vérifié que l'algorithme P est correct.

Pour montrer que P est complet, il suffit de montrer que si l'algorithme P construit une dérivation qui ne se termine pas par une liste fermée de liste de formules, alors la formule A n'est pas valide.

Supposons donc que l'algorithme P construit une dérivation qui ne se termine pas par une liste fermée de liste de formules.

Soit T l'arbre associé à la dérivation équitable (de la formule A) construite par l'algorithme P.

Supposons que l'algorithme P se termine.

L'arbre T est alors fini.

Soit Γ la dernière liste de listes de formules de la dérivation de A.

Puisque l'algorithme P se termine et que Γ n'est pas fermée, il y a une liste terminale élément de Γ.

Puisque Γ est la liste des étiquettes des feuilles de T, il y a une feuille u de T qui est étiquetée par une liste X^n terminale.

Soit B la branche d'extremité u et H l'ensemble des formules de cette branche.

D'après le lemme 1.4.2.16, l'ensemble H possède un modèle qui a pour domaine $c_1, \ldots c_n$ (si $n > 0$) et c_1 (si $n = 0$) et pour base l'ensemble des formules atomiques de X^n.

Puisque $\neg A$ fait partie de toute branche, il en resulte que $\neg A$ est satisfaisable dans ce modèle, donc A n'est pas valide.

Supposons que l'algorithme P ne se termine pas.

L'arbre T est infini donc posséde une branche B infinie.

Soit H l'ensemble des formules de cette branche. D'après le lemme 1.4.2.17, l'ensemble H possède un modèle dont le domaine est $\{c_i \mid i \in \mathbb{N} - \{0\}\}$.

Puisque $\neg A$ fait partie de toute branche, il en resulte que $\neg A$ est satisfaisable, donc A n'est pas valide.

Pour donner au lecteur une idée sur la manière de construire une dérivation équitable, nous avions affirmé sans preuve, que la construction 1.4.2.10 permettait d'obtenir une dérivation équitable.

Nous terminons ce paragraphe en montrant la justesse de cette construction.

Lemme 1.4.2.19

Toute dérivation construite par la construction 1.4.2.10 est équitable.

preuve :

Soit $\Gamma_0, \Gamma_1, \ldots$ une dérivation infinie construite comme il est indiqué en 1.4.2.10.

Soit T l'arbre associé à cette dérivation. Soit b une branche infinie de l'arbre T et H l'ensemble des formules de cette branche.

(1) Aucun sommet de la branche b n'est étiqueté par une étiquette fermée.

En effet, supposons au contraire que le sommet u de la branche b est étiqueté par une liste fermée. D'après la construction 1.4.2.10, ce sommet n'a pas de successeur donc il ne peut pas être sur une branche infinie.

(2) S'il y a une formule A telle que $\neg\neg A \in H$ alors $A \in H$.

Supposons que la formule A vérifie $\neg\neg A \in H$.

Soit $X_0, X_1, \ldots$ la suite des étiquettes des sommets de la branche b. Puisque $\neg\neg A \in H$, il existe un entier i tel que $\neg\neg A$ est élément de la liste X_i. Nous prouvons en deux étapes que $A \in H$.

(21) Il existe j tel que la formule $\neg\neg A$ n'est pas marquée dans la liste X_j.

Si $\neg\neg A$ n'est pas marquée dans X_i, on prend $j = i$.

Supposons que $\neg\neg A$ est marquée dans X_i et soit k le nombre de formules dérivables et non marquées de X_i, alors dans X_{i+k} toutes les formules dérivables (parmi lesquelles il y a $\neg\neg A$) sont marquées et donc dans X_{i+k+1}, la formule $\neg\neg A$ n'est pas marquée.

(22) Il existe k tel que A soit élément de la liste X_k et par suite $A \in H$.

Supposons que $\neg\neg A$ n'est pas marquée dans la liste X_j et soit k le nombre de formules dérivables et non marquées de X_j, alors dans X_{j+k} toutes les formules sont marquées ou non dérivables. Par suite A est une formule marquée de la liste X_{j+k}.

(3) pour toutes formules A et B et toute variable x

si $A \wedge B \in H$ alors $A \in H$ et $B \in H$

si $\neg(A \wedge B) \in H$ alors $\neg A \in H$ ou $\neg B \in H$

si $\neg\forall x A \in H$ alors il existe une constante c_i telle que $\neg A[x:= c_i] \in H$

La preuve est analogue à celle de (2).

(4) Si $\forall x A \in H$ alors pour tout $i \in \mathbb{N} - \{0\}$, on a $A[x:= c_i] \in H$.

Par une preuve analogue à celle de (2), on vérifie que :

Si $\forall x A \in H$ alors $\forall_1 x A \in H$ et $A[x:=c_1] \in H$

et que pour tout $i \in \mathbb{N} - \{0\}$

Si $\forall_i xA \in H$ alors $\forall_{i+1} xA \in H$ et $A[x:=c_{i+1}] \in H$

La propriété (4) en résulte immédiatement.

Les propriétés (1), (2), (3), (4) nous prouvent que la dérivation est équitable.

Remarque: La preuve de l'équitabilité de l'autre marquage suggéré (cf 1.4.2.10 avec le remplacement de l'étape (4) par l'étape (4')) est presque identique à la preuve ci-dessus parce l'ensemble des formules que l'on peut produire sur une branche d'un arbre de dérivation sans utiliser la règle (C$\forall$) est un ensemble fini.

Chapitre 2
Transformation de formules (skolémisation par étapes)

La skolémisation d'une formule est le plus souvent décrite comme une transformation, qui change une formule A en une formule universelle B qui a un modèle si et seulement si A a un modèle. Cette transformation globale peut être décomposée en une suite d'opérations élémentaires, consistant à éliminer un quantificateur existentiel d'occurence positive (resp universel d'occurrence négative). Plutôt que de décrire la transformation globale de façon compliquée (comme le font presque tous les auteurs consultés), j'ai décidé d'étudier ces opérations élémentaires.
Pour obtenir la meilleure transformation possible, j'ai pris modèle sur la transformation étoile de [P.B.Andrews 86].

Nous prouvons essentiellement (en vue d'application au chapitre 3) que l'on peut éliminer une occurrence positive (resp. négative) d'un quantificateur existentiel (resp. universel) en préservant l'existence d'un modèle.

Nous utilisons les définitions de [R.Lalement 90], en particulier sa définition de la substitution avec renommage des variables liées, ainsi que ses notations avec une seule exception, le sens de la formule α dans l'algèbre $\mathfrak{A}$ est notée $\mathfrak{A}(\alpha)$ au lieu de $\alpha^{\mathfrak{A}}$.
Nous avons déjà rappelé ces notations en 1.1.1.

2.1 Le principe de la transformation

Lemme 2.1.1 :
Soit $\exists x\alpha$ une formule ayant pour variables libres $y_1, ...y_n$ (où $n \geq 0$). Soit f un symbole qui ne figure pas dans la formule $\exists x\alpha$.
(1) $\models \alpha[x:= f(y_1, ...y_n)] \Rightarrow \exists x\alpha$
(2) Soit $\mathfrak{A}$ une algèbre. Il existe une algèbre $\mathfrak{A}'$ ne différant de $\mathfrak{A}$ que dans l'interprétation du symbole f et vérifiant $\mathfrak{A}(\exists x\alpha) = \mathfrak{A}'(\alpha[x:= f(y_1, ...y_n)])$.

preuve : R.Lalement 90 , p167.

2.2 Occurrences dans une formule et signe des occurrences

Nous utilisons la notion d'occurrence positive et négative d'une formule de [R.Lalement 90] pour généraliser le lemme 2.2.1.
Rappelons brièvement la notion d'occurrence et de signe d'une occurrence.

Occurrences

On considère les formules comme des termes sur une signature formée notamment des connectives logiques et des $\forall x$, $\exists x$.

Prenons un exemple.

L'ensemble des occurrences des sous-formules de la formule

$\phi = \forall x(A(x) \Rightarrow \exists y\ B(x,y))$

est $\mathcal{O}(\phi) = \{\varepsilon, 1, 11, 12, 121\}$.

Soit u une occurrence d'une sous-formule de la formule ϕ : $\phi(u)$ est le symbole d'occurrence u, ϕ/u est la sous formule d'occurrence u.

Soit α une formule, $\phi[\alpha]_u$ est la formule obtenue en remplaçant la sous-formule de ϕ d'occurrence u par la formule α.

C'est ainsi que :

$\phi(\varepsilon) = \forall x$, $\phi/\varepsilon = \phi$, $\phi(1) = \Rightarrow$, $\phi/1 = (A(x) \Rightarrow \exists y\ B(x,y))$, $\phi(11) = A$, $\phi/11 = A(x)$

$\phi(12) = \exists y$, $\phi/12 = \exists y\ B(x,y)$

$\phi[C(x) \wedge D(x)]_{11} = \forall x(\ (\ C(x) \wedge D(x)\ ) \Rightarrow \exists y\ B(x,y))$

Signe d'une occurrence

Une occurrence d'une formule est positive (resp. négative) si elle est dans la portée d'un nombre pair (resp. impair) de négations (explicitement ou implicitement dans le membre gauche d'une implication).

Plus précisément le signe des occurrences de ϕ est défini par une application $sg : \mathcal{O}(\phi) \rightarrow \{1, -1\}$:

$sg(\varepsilon) = 1$

si $\phi / u = \neg\alpha$ alors $sg(u1) = - sg(u)$

si $\phi / u = (\alpha \wedge \beta)$ ou $(\alpha \vee \beta)$ alors $sg(u1) = sg(u2) = sg(u)$

si $\phi / u = (\alpha \Rightarrow \beta)$ alors $sg(u1) = - sg(u)$ et $sg(u2) = sg(u)$

si $\phi / u = \forall x\alpha$ ou $\exists x\alpha$ alors $sg(u1) = sg(u)$

Une occurrence u est positive (resp. négative) si $sg(u) = 1$ (resp. -1).

2.3 Skolémisation en une occurrence

Lemme 2.3.1 :

Soit $\exists x\beta$ une formule ayant pour variables libres $y_1, ...y_n$ (où $n \geq 0$).

Soit α une formule et u une occurrence d'une sous-formule de α.

Soit f un symbole qui ne figure pas dans la formule $\alpha[\exists x\beta]_u$.

(1) Si l'occurrence u est positive alors $\models\ \alpha[\ \beta[\ x:= f(\ y_1, ...y_n)\ ]\ ]_u \Rightarrow\ \alpha[\exists x\beta]_u$

et si l'occurrence u est négative alors $\models\ \alpha[\exists x\beta]_u \Rightarrow \alpha[\ \beta[\ x:= f(\ y_1, ...y_n)\ ]\ ]_u$

(2) Soit $\mathfrak{A}$ une algèbre. Il existe une algèbre $\mathfrak{A}'$ ne différant de $\mathfrak{A}$ que dans l'interprétation du symbole f et vérifiant $\mathfrak{A}(\ \alpha[\exists x\beta]_u\ ) = \mathfrak{A}'(\ \alpha[\ \beta[\ x:= f(\ y_1, ...y_n)\ ]\ ]_u\ )$.

preuve : par récurrence sur l'ensemble des sous-formules de la formule α, en utilisant le lemme 2.1.1.

Lemme 2.3.2 :

Soit $\exists x\beta$ une formule ayant pour variables libres $y_1, y_2, ...y_n$ (où $n \geq 0$).

Soit α une formule et u une occurrence d'une sous-formule de α.

Soit f un symbole qui ne figure pas dans la formule $\alpha[\exists x\beta]_u$.

Si u est une occurrence positive alors

$\alpha[\exists x\beta]_u$ a un modèle si et seulement si $\alpha[\ \beta[\ x:= f(\ y_1, ...y_n)\ ]\ ]_u$ a un modèle.

Si u est une occurrence négative alors

$\alpha[\exists x\beta]_u$ est valide si et seulement si $\alpha[\ \beta[\ x:= f(\ y_1, ...y_n)\ ]\ ]_u$ est valide

preuve : conséquence immédiate du lemme 2.3.1.

Lemme 2.3.3 :

Soit $\forall x\beta$ une formule ayant pour variables libres $y_1, y_2, ...y_n$ (où $n \geq 0$).

Soit α une formule et u une occurrence d'une sous-formule de α.

Soit f un symbole qui ne figure pas dans la formule $\alpha[\forall x\beta]_u$.

Si u est une occurrence positive alors

$\alpha[\forall x\beta]_u$ est valide si et seulement si $\alpha[\ \beta[x:=f(\ y_1, y_2, ...y_n)]\ ]_u$ est valide.

Si u est une occurrence négative alors

$\alpha[\forall x\beta]_u$ a un modèle si et seulement si $\alpha[\ \beta[x:=f(\ y_1, y_2, ...y_n)]\ ]_u$ a un modèle.

preuve : conséquence du lemme 2.3.2 en utilisant l'équivalence entre $\forall x\beta$ et $\neg\exists x\neg\beta$.

Chapitre 3
Méthode des tableaux avec variables libres

La méthode des tableaux usuelle décrite par [J.Gallier 86] ou [P.B.Andrews 86] n'utilise pas l'algorithme d'unification. Aussi elle ne permet pas d'écrire des procédures de démonstration automatique «raisonnables» dès que l'on utilise des symboles de fonction. En m'aidant du livre de [M.Fitting 90], je présente une variante de la méthode des tableaux qui emploie un algorithme d'unification.

Pour sa méthode, M.Fitting propose une règle d'élimination des quantificateurs existentiels (la delta-règle) qui me semblait insuffisamment prouvée dans [M.Fitting 90]. Pour comprendre cette règle, j'ai exprimé l'état d'un tableau comme étant une liste de buts. A chaque but on associe une conjonction de ses formules et au tableau lui-même on associe une disjonction des formules associées aux buts.

Avec cette présentation, la delta-règle apparait comme étant l'élimination d'un quantificateur existentiel (resp. universel) dans une occurrence positive (resp. négative) de la formule qui est associée au tableau (cf chapitre 2). Cela nous permet de prouver la correction d'une nouvelle delta-règle et de remarquer que la règle de Fitting ajoute une fonction de Skolem qui peut avoir des arguments inutiles. Nous expliquons cette remarque par un exemple :

La skolémisation de la formule $A = \forall x(P(x) \vee \exists y Q(y))$ peut donner
soit la formule $B = \forall x(P(x) \vee Q(a))$ où a est une constante
soit la formule $C = \forall x(P(x) \vee Q(f(x)))$
La formule A a un modèle si et seulement si B a un modèle si et seulement si C a un modèle, mais la transformation de A en B (qui est décrite au chapitre 2) est celle que j'utilise, tandis que la transformation de A en C est celle implicitement utilisée par M.Fitting.

Comme au chapitre 1, nous présentons une classe d'algorithmes corrects et complets en séparant les règles logiques et le contrôle de l'enchaînement des règles.

Contrairement au chapitre 1, les buts d'un tableau **ne sont pas indépendants** les uns des autres. Si une variable libre apparait dans deux buts différents, elle doit avoir la même valeur dans les deux buts.
Soit T un arbre de dérivation associé à une dérivation équitable. Une branche de T est **complète**, si elle est infinie ou s'il est impossible d'appliquer une règle à l'étiquette de l'extrémité de la branche.
Pour obtenir un algorithme complet avec la méthode des tableaux sans variables libres, il faut veiller à ce que, dans le cas où l'algorithme a pour donnée une formule A fermée non valide, il «construise» au moins une branche complète. Par contre dans le même cas, avec les algorithmes de ce chapitre, il faut veiller à ce que toutes les branches soient complètes.

3.1 Les règles de la méthode

Dans la suite nous désignons les formules par les lettres A, B, C, les listes de formules par les lettres X, Y, Z, les listes de listes de formules par les lettres Γ, Δ, Π, Ψ.
A une liste de formules, on associe la conjonction des formules de la liste, la formule faux étant associée à la liste vide. A une liste de listes de formules, on associe la disjonction des formules associées aux listes de formules. Dans la suite nous confondons une liste de listes de formules et la formule qui est associée à la liste de listes de formules.

Une substitution est une application des variables dans les termes.
Une substitution est fermée si l'image de chaque variable est un terme sans variables.
Toute substitution s'étend naturellement aux termes, formules, listes de formules, listes de listes de formules.
Dans l'extension aux formules, nous supposons que les substitutions sont effectuées avec renommage des variables liées.
Rappelons le principe de ce renommage.
Supposons que l'ensemble des variables soit totalement ordonné. Soit s une substitution, $\exists x\alpha$ une formule de variables libres $y_1, ...y_n$ $(n \geq 0)$.

$s(\exists x\alpha) = \exists z\ s(\alpha')$

z est la plus petite variable qui ne figure pas dans les termes $s(y_1), ...s(y_n)$

α' est obtenue en remplaçant dans α toutes les occurrences libres de x par z

Définition 3.1.1

Γ **produit** Δ dans les cas suivants :

$(\neg\neg)$ $\quad \Gamma = \Pi, [X, \neg\neg A, Y], \Psi \qquad \Delta = \Pi, [X, A, Y], \Psi$

$(\wedge)$ $\quad \Gamma = \Pi, [X, A \wedge B, Y], \Psi \qquad \Delta = \Pi, [X, A, B, Y], \Psi$

$(\neg\wedge)$ $\Gamma = \Pi, [X, \neg(A \wedge B), Y], \Psi \quad \Delta = \Pi, [X, \neg A, Y], [X, \neg B, Y], \Psi$

$(\forall)$ $\quad \Gamma = \Pi, [X, \forall xA, Y], \Psi \qquad \Delta = \Pi, [X, \forall xA, A[x := y], Y], \Psi$
y est une variable

$(\neg\forall)$ $\quad \Gamma = \Pi, [X, \neg\forall xA, Y], \Psi \qquad \Delta = \Pi, [X, \neg A[x := f(y_1,...y_n)\ ], Y], \Psi$
$y_1,...y_n$ est une liste de toutes les variables libres de $\neg\forall xA$ et f est un nouveau symbole qui ne figure pas dans les formules de Γ.

Cette règle remplace la δ-règle de [M.Fitting90], pour laquelle $y_1,...y_n$ est la liste de toutes les variables libres de $[X, \neg\forall xA, Y]$.

Pour toutes les règles ci-dessus, la formule auquelle on applique la règle, est dérivée par la règle et les formules obtenues à partir de la formule dérivée sont produites par la règle.
Notons que les seules formules non dérivables sont les litteraux.

(substitution) $\Delta = s(\Gamma)$ s est une substitution.

(élimination des contradictions)
Dans Γ, il y a une occurrence d'une liste X de formules, qui comporte une formule et sa négation et Δ est obtenu en supprimant cette occurrence.

Lemme 3.1.2 (correction des règles)
Si Γ a un modèle et si Γ produit Δ alors Δ a un modèle.

preuve :
Pour les quatre règles ($\neg\neg$, $\wedge$, $\neg\wedge$, $\forall$) et pour l'élimination des contradictions, il est clair que Γ est équivalent à Δ.
La règle $\neg\forall$ est en fait l'élimination dans Γ, considérée comme une disjonction de conjonctions, d'une occurrence négative d'un quantificateur universel, donc, d'après le lemme 2.3.3, Γ a un modèle si et seulement si Δ a un modèle.
Pour la règle de substitution, tout modèle de Γ est un modèle de Δ.

Définition 3.1.3
Une dérivation de Γ en Δ est une suite Γ_i (telle que $0 \leq i \leq n$ et $0 \leq n$) vérifiant
$\Gamma_0 = \Gamma$, pour i de 0 à n - 1, Γ_i produit Γ_{i+1}, $\Gamma_n = \Delta$.

Γ **se dérive en** Δ si et seulement si il y a une dérivation de Γ en Δ.
Nous notons le fait que Γ se dérive en Δ par $\Gamma \vdash \Delta$

Lemme 3.1.4 (correction des dérivations)
Si Γ a un modèle et si Γ se dérive en Δ alors Δ a un modèle.

preuve : conséquence immédiate du lemme 3.1.2.

Lemme 3.1.5 (correction de la méthode)
Soit A une formule fermée. Si [[$\neg$A]] se dérive en [[]] alors la formule A est valide.

preuve :
Puisque A est une formule fermée, si elle n'était pas valide, $\neg A$ aurait un modèle.
si $\neg A$ avait un modèle, d'après le lemme 3.1.4, faux (la formule associée à la liste de liste de formules [[]]) aurait un modèle. Donc $\neg A$ n'a pas de modèle et par suite A est valide.

3.2 Les règles dérivées

Nous ajoutons aux règles précédentes des règles dérivées telles que si Γ produit Δ par une règle dérivée alors Γ se dérive en Δ par les règles non dérivées.
Les règles dérivées permettent de raccourcir les dérivations.

A cause de la diminution de la longueur des preuves, un démonstrateur automatique utilisant des règles dérivées peut être beaucoup plus rapide.
Un avantage essentiel de la méthode des tableaux est cette possibilité d'ajouter facilement de nouvelles règles (cf F.Oppacher and E.Suen 88).

3.2.1 Les règles simples

En utilisant les définitions des connectives $\vee$, $\exists$, $\Rightarrow$, $\Leftrightarrow$ à partir des connectives $\neg$, $\wedge$, $\forall$ on obtient facilement des règles dérivées pour les connectives $\vee$, $\exists$, $\Rightarrow$, $\Leftrightarrow$.
Dans les exemples de la suite, nous employons ces règles dérivées.

($\vee$)
Puisque $A \vee B = \neg(\neg A \wedge \neg B)$,
$\Pi, [X, A \vee B, Y], \Psi$ se dérive en $\Pi, [X, A, Y], [X, B, Y], \Psi$ grâce à la dérivation
$\Pi, [X, \neg(\neg A \wedge \neg B), Y], \Psi$
$\Pi, [X, \neg\neg A, Y], [X, \neg\neg B, Y], \Psi$
$\Pi, [X, A, Y], [X, \neg\neg B, Y], \Psi$
$\Pi, [X, A, Y], [X, B, Y], \Psi$

($\neg\vee$)
Puisque $\neg(A \vee B) = \neg A \wedge \neg B)$,
$\Pi, [X, \neg(A \vee B), Y], \Psi$ se dérive en $\Pi, [X, \neg A, \neg B, Y], \Psi$

($\Rightarrow$)
Puisque $(A \Rightarrow B) = \neg(A \wedge \neg B)$,
$\Pi, [X, (A \Rightarrow B), Y], \Psi$ se dérive en $\Pi, [X, \neg A, Y], [X, B, Y], \Psi$

($\neg\Rightarrow$)
Puisque $\neg (A \Rightarrow B) = (A \wedge \neg B)$,

Π, [X, $\neg$ (A $\Rightarrow$ B), Y], Ψ se dérive en Π, [X, A, $\neg$B, Y], Ψ

($\Leftrightarrow$)

Puisque (A $\Leftrightarrow$ B) = $\neg(A \wedge \neg B) \wedge \neg(B \wedge \neg A)$,

Π, [X, (A $\Leftrightarrow$ B), Y], Ψ se dérive en Π, [X, A, B, Y], [X, $\neg$A, $\neg$B, Y], Ψ grâce à la dérivation

Π, [X, $\neg(A \wedge \neg B) \wedge \neg(B \wedge \neg A)$, Y], Ψ

Π, [X, $\neg(A \wedge \neg B)$,$\neg(B \wedge \neg A)$, Y], Ψ

Π, [X, $\neg$A, $\neg(B \wedge \neg A)$, Y], [X, $\neg\neg$B,$\neg(B \wedge \neg A)$, Y], Ψ

Π, [X, $\neg$A, $\neg(B \wedge \neg A)$, Y], [X, B,$\neg(B \wedge \neg A)$, Y], Ψ

Π, [X, $\neg$A, $\neg$B, Y], [X, $\neg$A, $\neg\neg$A, Y], [X, B,$\neg(B \wedge \neg A)$, Y], Ψ

par élimination de la contradiction nous obtenons

Π, [X, $\neg$A, $\neg$B, Y], [X, B,$\neg(B \wedge \neg A)$, Y], Ψ

Π, [X, $\neg$A, $\neg$B, Y], [X, B, $\neg$B, Y], [X, B, $\neg\neg$A, Y], Ψ

par élimination de la contradiction nous obtenons

Π, [X, $\neg$A, $\neg$B, Y], [X, B, $\neg\neg$A, Y], Ψ

Π, [X, A, B, Y], [X, $\neg$A, $\neg$B, Y], Ψ

($\neg\Leftrightarrow$)

Puisque $\neg$(A $\Leftrightarrow$ B) = $\neg(\neg(A \wedge \neg B) \wedge \neg(B \wedge \neg A))$,

Π, [X, $\neg$(A $\Leftrightarrow$ B), Y], Ψ se dérive en Π,[X, A, $\neg$B, Y], [X, $\neg$A, B, Y], Ψ

($\exists$)

Puisque $\exists$xA = $\neg\forall$x$\neg$A,

Π, [X, $\exists$xA, Y], Ψ se dérive en Π, [X, A[x:= $f(y_1,...y_n)$], Y], Ψ

$y_1,...y_n$ est une liste de toutes les variables libres de $\exists$xA et f est un nouveau symbole qui ne figure pas dans les formules de Π, [X, $\exists$xA, Y], Ψ.

Cette règle est aussi un exemple de δ-règle modifiée.

($\neg\exists$)

Puisque $\neg\exists$xA = $\forall$x$\neg$A,

Π, [X, $\neg\exists$xA, Y], Ψ se dérive en Π, [X, $\neg\exists$xA, $\neg$A[x := y], Y], Ψ

où y est une variable

Exemple 3.2.1.1

Montrons que $\forall x(P(x) \Rightarrow Q) \Leftrightarrow (\exists xP(x) \Rightarrow Q)$ est valide.

Dans la dérivation, l'occurrence de la formule dérivée figure en caractères gras.

(1) [[$\neg(\forall \mathbf{x}(\mathbf{P(x)} \Rightarrow \mathbf{Q}) \Leftrightarrow (\exists \mathbf{x}\mathbf{P(x)} \Rightarrow \mathbf{Q})$)]]

par la règle $\neg\Leftrightarrow$

(2) [[$\forall x(P(x) \Rightarrow Q), \neg (\exists \mathbf{x}P(\mathbf{x}) \Rightarrow \mathbf{Q})$] , [$\neg\forall x(P(x) \Rightarrow Q), \exists xP(x) \Rightarrow Q$]]

par la règle $\neg\Rightarrow$

(3) [[$\forall x(P(x) \Rightarrow Q), \exists \mathbf{x}P(\mathbf{x}), \neg Q$] , [$\neg\forall x(P(x) \Rightarrow Q), \exists xP(x) \Rightarrow Q$]]

par la règle $\exists$, a étant une constante

(4) [[$\forall \mathbf{x}(\mathbf{P}(\mathbf{x}) \Rightarrow \mathbf{Q}), P(a), \neg Q$] , [$\neg\forall x(P(x) \Rightarrow Q), \exists xP(x) \Rightarrow Q$]]

par la règle $\forall$, y étant une variable

(5) [[$\forall x(P(x) \Rightarrow Q), \mathbf{P(y)} \Rightarrow \mathbf{Q}, P(a), \neg Q$] , [$\neg\forall x(P(x) \Rightarrow Q), \exists xP(x) \Rightarrow Q$]]

par la règle $\Rightarrow$

(6) [[$\forall x(P(x) \Rightarrow Q), \neg P(y), P(a), \neg Q$] , [$\forall x(P(x) \Rightarrow Q), Q, P(a), \neg Q$],
[$\neg\forall x(P(x) \Rightarrow Q), \exists xP(x) \Rightarrow Q$]]

par la règle de substitution, en remplaçant y par a

(7) [[$\forall x(P(x) \Rightarrow Q), \neg P(a), P(a), \neg Q$] , [$\forall x(P(x) \Rightarrow Q), Q, P(a), \neg Q$],
[$\neg\forall x(P(x) \Rightarrow Q), \exists xP(x) \Rightarrow Q$]]

par deux applications de la règle de contradiction

(8) [[$\neg\forall \mathbf{x}(\mathbf{P}(\mathbf{x}) \Rightarrow \mathbf{Q}), \exists xP(x) \Rightarrow Q$]]

par la règle $\neg\forall$, a étant une constante

(9) [[$\neg(\mathbf{P(a)} \Rightarrow \mathbf{Q}), \exists xP(x) \Rightarrow Q$]]

par la règle $\neg\Rightarrow$

(10) [[$P(a), \neg Q, \exists \mathbf{x}P(\mathbf{x}) \Rightarrow \mathbf{Q}$]]

par la règle $\Rightarrow$

(11) [[$P(a), \neg Q, \neg\exists xP(x)$], [$P(a), \neg Q, Q$]]

par la règle de contradiction

(12) [[$P(a), \neg Q, \neg\exists \mathbf{x}P(\mathbf{x})$]]

par la règle $\neg\exists$, y étant une variable libre

(13) [[$P(a), \neg Q, \neg\exists xP(x), \neg P(y)$]]

par la règle de substitution, en remplaçant y par a

(14) [[$P(a), \neg Q, \neg\exists xP(x), \neg P(a)$]]

par la règle de contradiction

(15) faux

3.2.1 règles composées

Les règles composées utilisent plusieurs formules. Nous en donnons deux exemples.

Π, [X, A, Y, $A \Rightarrow B$, Z], Ψ se dérive en Π, [X, A, Y, B, Z], Ψ (modus ponens)

Π, [X, $\neg$A, Y, $B \Rightarrow A$, Z], Ψ se dérive en Π, [X, $\neg$A, Y, $\neg$B, Z], Ψ (modus tollens)

3.3 Complétude de la méthode

Pour éviter d'ennuyeuses complications, nous n'utilisons dans ce paragraphe que les règles non

dérivées.

Définition 3.3.1

La liste de liste de formules Γ est contradictoire si et seulement si il y a une substitution s telle que toutes les listes de formules de $s(\Gamma)$ comportent une formule atomique et sa négation.

Exemple

[[P(x), ¬P(f(y)), Q(a, g(z))], [R(y), ¬R(a)]] est contradictoire.

Le test « Γ est contradictoire » peut être effectué en utilisant l'algorithme d'unification de Herbrand.

Une classe d'algorithmes 3.3.2

Soit A une formule fermée.

On construit une dérivation **équitable** Γ_i (où $i \in \mathbb{N}$) telle que

$\Gamma_0 = [[\neg A]]$

pour tout i, Γ_i produit Γ_{i+1} par une règle logique ($\neg\neg$, $\wedge$, $\neg\wedge$, $\forall$, $\neg\forall$)

et on arrête cette construction dès que Γ_i est contradictoire ou que Γ_i n'est plus dérivable par une règle logique.

Si lors de l'arrêt Γ_i est contradictoire, on indique que A est valide sinon on indique que A n'est pas valide.

Il est clair que tout algorithme de cette classe est correct (au sens défini en 1.4.2.8). En effet s'il termine en vérifiant la condition « Γ_i est contradictoire », il y a une dérivation de [[¬A]] en [[]], donc d'après le lemme 3.1.5 la formule A est valide.

Informellement une dérivation équitable est une dérivation, dans laquelle aucune formule dérivable (autrement dit qui n'est pas un littéral) n'est définitivement oubliée.

Pour préciser cette notion, nous associons à une dérivation (finie ou infinie) obtenue grâce aux règles logiques ($\neg\neg$, $\wedge$, $\neg\wedge$, $\forall$, $\neg\forall$) un arbre de dérivation défini exactement comme en 1.4.2.11. L'ensemble des dérivations équitables est défini par une condition portant sur les branches infinies de cet arbre.

Exemple

Aux six premières étapes de la dérivation de $[[\neg(\forall x(P(x) \Rightarrow Q) \Leftrightarrow (\exists x P(x) \Rightarrow Q))]]$ figurant dans l'exemple 3.2.1.1 , on associe l'arbre suivant dont la structure est indiquée par la disposition et la numérotation des sommets :

(ε) $[\neg(\forall x(P(x) \Rightarrow Q) \Leftrightarrow (\exists xP(x) \Rightarrow Q))]$

(1) $[\forall x(P(x) \Rightarrow Q), \neg (\exists xP(x) \Rightarrow Q)]$

(11) $[\forall x(P(x) \Rightarrow Q), \exists xP(x), \neg Q]$

(111) $[\forall x(P(x) \Rightarrow Q), P(a), \neg Q]$

(1111) $[\forall x(P(x) \Rightarrow Q), P(y) \Rightarrow Q, P(a), \neg Q]$

(11111) $[\forall x(P(x) \Rightarrow Q), \neg P(y), P(a), \neg Q]$

(11112) $[\forall x(P(x) \Rightarrow Q), Q, P(a), \neg Q]$

(2) $[\neg\forall x(P(x) \Rightarrow Q), \exists xP(x) \Rightarrow Q]$

Définition 3.3.3 (dérivation équitable)

Une dérivation est équitable si lorsque l'arbre T qui est associé à la dérivation est **infini**, il vérifie que :

pour toute branche b, l'ensemble E des formules de la branche b est tel que pour toutes formules A et B et toute variable x :

(a) si $\neg\neg A \in E$ alors $A \in E$

(b) si $A \wedge B \in E$ alors $A \in E$ et $B \in E$

(c) si $\neg(A \wedge B) \in E$ alors $\neg A \in E$ ou $\neg B \in E$

(d) si $\neg\forall xA \in E$ alors il y a un terme t tel que $\neg A[x:=t] \in E$

(f) si $\forall xA \in E$ alors il y a une infinité de variables y telles que $A[x:=y] \in E$

Notons qu'avec cette définition, si une branche est finie dans un arbre infini construit équitablement, alors il ne peut pas y avoir de formules universelles sur la branche.

Avant de prouver que tout algorithme de la classe 3.3.2 est complet, il faut déjà montrer qu'il y a un algorithme dans cette classe, autrement dit qu'il est possible de construire une dérivation équitable.

Construction d'une dérivation équitable 3.3.4

On suppose que l'on peut marquer les formules.

Soit $\Gamma_0, \ldots \Gamma_i$ la partie déjà construite d'une dérivation. Supposons que Γ_i comporte au moins une formule qui n'est pas un littéral.

Γ_{i+1} est construite en quatre étapes successives :

(1) Si toutes les formules de Γ_i sont marquées ou sont des litteraux, alors on enlève les marques.

(2) On choisit dans Γ_i une liste X de formules comportant au moins une formule non littérale et non marquée.

(3) On choisit dans la liste X une formule A qui est non littérale et non marquée.

(4) On applique la règle correspondant à A. Si A est la formule $\forall xB$ alors A est remplacée par $\forall xB$, $B[x:=y]$, la formule $\forall xB$ est marquée et y doit être une variable nouvelle, c'est-à-dire qui n'est pas libre dans $\Gamma_0, \ldots \Gamma_i$.

On peut prouver (de façon analogue à la preuve du lemme 1.2.4.19) que cet algorithme construit une dérivation équitable.
Il y a bien d'autres manières de construire des dérivations équitables, donc plusieurs réalisations d'algorithmes corrects et complets avec des comportements très différents.

Lemme 3.3.5

Soit T un arbre de dérivation et b une branche finie dont le dernier sommet est étiqueté par une liste de litteraux. L'ensemble E des formules de la branche b a la propriété suivante :
Il n'y a pas de formules universelles (c'est-à-dire de la forme $\forall xA$) dans E et pour toutes formules A et B et toute variable x :

(a) si $\neg\neg A \in E$ alors $A \in E$

(b) si $A \wedge B \in E$ alors $A \in E$ et $B \in E$

(c) si $\neg(A \wedge B) \in E$ alors $\neg A \in E$ ou $\neg B \in E$

(d) si $\neg\forall xA \in E$ alors il y a une constante c telle que $\neg A[x:=c] \in E$

preuve :
Nous prouvons d'abord qu'il n'y a pas de formules universelles dans E.
Supposons le contraire. Une formule universelle présente sur une branche est élément de la liste du dernier sommet de la branche. Par suite la liste étiquette du dernier sommet de la branche b comporterait une formule qui n'est pas un littéral. Ce qui est contraire à l'hypothèse.

Vérifions la condition d (la vérification des autres conditions est analogue et plus simple).
Puisque $\neg\forall xA \in E$ la formule a été dérivée sur la branche, et puisqu'il n'y a de formules universelles sur la branche, il n'y a pas de variables libres dans les formules de E, donc il existe une constante c telle que $\neg A[x:=c] \in E$.

Lemme 3.3.6

Tout algorithme de la classe d'algorithmes 3.3.2 est complet.

preuve :
Soit P un algorithme de cette classe.
Pour montrer que P est complet, il suffit de montrer que si l'algorithme construit une dérivation qui ne se termine pas par une liste contradictoire alors la formule A n'est pas valide.

Supposons donc que l'algorithme P construit une dérivation qui ne se termine pas par une liste contradictoire.
Soit T l'arbre associé à la dérivation équitable (de la formule A) construite par l'algorithme P.

Supposons que l'algorithme P se termine.
L'arbre T est alors fini.
Soit Γ la dernière étape de la dérivation de A. Puisque Γ n'est pas contradictoire et que l'algorithme s'est terminé, d'après le test d'arrêt, Γ ne comprend que des litteraux.

Soit Σ la signature utilisée au cours de la dérivation. Si Σ ne comporte pas de constantes, on lui en ajoute une pour que l'ensemble des Σ-termes fermées ne soit pas vide.

(1) il y a une branche de T qui ne comprend pas une formule atomique et sa négation.

Supposons le contraire. Puisque les litteraux d'une branche se retrouvent dans l'étiquette de l'extremité de la branche, il en résulte que Γ qui est la suite des étiquettes des feuilles de T serait fermée donc que Γ serait contradictoire contrairement aux hypothèses.

Il existe donc une branche b de T qui ne comprend pas de formule atomique et sa négation.
Soit E l'ensemble des formules de cette branche. D'après le lemme 3.3.5, E est un ensemble d'Hintikka (cf définition en 1.4.2.14) donc E a un modèle.

Puisque $\neg A$ fait partie de toute branche, il en résulte que $\neg A$ est satisfaisable donc A n'est pas valide.

Supposons que l'algorithme P ne se termine pas.
L'arbre T est infini et l'algorithme a construit une dérivation $\Gamma_0, \Gamma_1, \ldots$ équitable qui ne termine pas. Soit $T_0, T_1, \ldots$la suite infinie d'arbres où T_i est associé à la dérivation $\Gamma_0, \ldots \Gamma_i$.
Puisque l'algorithme ne se termine pas :

(1) pour tout i, Γ_i n'est pas contradictoire.

Nous montrons que sous ces hypothèses, la formule $\neg A$ est satisfaisable.

Soit Σ la signature utilisée au cours de la dérivation.
Si Σ ne comporte pas de constantes, on lui en ajoute une pour que l'ensemble D des Σ-termes fermés ne soit pas vide.
Puisque la signature Σ est finie ou dénombrable, l'ensemble D est dénombrable : il existe donc une énumération t_i où $i \in \mathbb{N}$ des termes fermés de D.

Soit V l'ensemble des variables libres de la dérivation.
D'après la condition (f) vérifiée par tout arbre infini associé à une dérivation équitable, il y a sur chaque branche et pour chaque formule $\forall x A$ de la branche une infinité de variables y telle que A[x:=y] figure aussi sur la branche. La condition (f) peut donc être traduite par :

(2) il existe une fonction surjective f de V dans $\mathbb{N}$ telle que pour toute branche , toute formule $\forall xA$ de la branche et tout $i \in \mathbb{N}$, il existe une variable y telle que $y \in f^{-1}(i)$ et A[x:=y] figure aussi sur la branche.

On définit une substitution $s : V \rightarrow D$ par

pour tout $y \in V$, $s(y) = t_{f(y)}$

Soit s(T) l'arbre obtenu en appliquant la substitution s à chaque formule de T.
Nous prouvons tout d'abord que :

(3) il y a une branche de l'arbre s(T) qui ne comporte pas une formule atomique et sa négation.

Supposons au contraire que toute branche de s(T) comporte une formule atomique et sa négation.
Puisque toute branche de s(T) comporte une formule atomique et sa négation, il existe un arbre $s(T_i)$ pour lequel toute branche comporte une formule atomique et sa négation.
Si un littéral figure dans T_i , il figure aussi dans les feuilles de T_i , puisqu'un littéral n'est pas dérivé.
Il en est évidemment de même pour l'arbre $s(T_i)$. Par suite chaque étiquette d'une feuille de $s(T_i)$ comporte une formule atomique et sa négation et par conséquent $s(\Gamma_i)$ qui est la liste des étiquettes des feuilles de $s(T_i)$ est telle que chaque liste de formules de $s(\Gamma_i)$ contient une formule atomique et sa négation.
Par conséquent Γ_i est contradictoire, ce qui est contraire à la propriété (1).

Il existe donc une branche b de l'arbre s(T) qui ne comprend pas une formule atomique et sa négation.
Nous montrons que l'ensemble F des formules de cette branche est un ensemble d'Hintikka (définition en 1.4.2.14).

Soit E, l'ensemble des formules de cette même branche b dans l'arbre T.
(a) si $\neg\neg B \in F$ alors il existe $C \in E$ tel que $\neg\neg B = s(C)$.

Par définition des substitutions il existe D tel que $C = \neg\neg D$ et $B = s(D)$.

La dérivation étant équitable, on a $D \in E$ et donc $B \in F$.

De même on prouve que
(b) si $B \wedge C \in F$ alors $B \in F$ et $C \in F$
(c) si $\neg(B \wedge C) \in F$ alors $B \in F$ et $C \in F$
(d) si $\neg\forall xB \in F$ alors il y a un terme $t \in D$ tel que $\neg B[x:=t] \in F$
Prouvons que
(f) si $\forall xB \in F$ alors pour tout terme $t \in D$ on a $B[x:=t] \in F$

Si $\forall xB \in F$ alors il existe $C \in E$ tel que $\forall xB = s(C)$.

Par définition des substitutions il existe D tel que $C = \forall xD$ et $B = s'(D)$ où s'et s sont

identiques sauf pour x où s'(x) = x.

La dérivation étant équitable, d'après (2) pour tout $i \in \mathbb{N}$ il y a une variable $y \in f^{-1}(i)$ telle que $D[x:=y] \in E$

Par définition de s, $s(y) = t_i$ et par suite $s(D[x:=y]) = B[x:=t_i]$

Donc pour tout $t \in D$ on a $B[x:=t] \in F$.

D'après les conditions (a)-(f) et le fait que dans F il n'y a pas une formule et sa négation, F est un ensemble d'Hintikka, donc possède un modèle de domaine D.

Puisque $\neg A$ est une formule fermée, on a $\neg A = s(\neg A)$.

Par suite puisque $\neg A$ est sur toute branche, $\neg A \in F$ donc a un modèle dénombrable, donc est satisfaisable (c.q.f.d.).

Remarque

La preuve de la correction de l'algorithme nous fournit (péniblement il est vrai) deux résultats supplémentaires :

(1) les règles de 3.1 sont complètes : car si A est valide, l'algorithme s'arrête en ayant construit une dérivation terminée par un Γ_i contradictoire, ce qui constitue une preuve avec les règles du premier paragraphe

(2) si A a un modèle, alors A a un modèle dénombrable.

Conclusion

Bilan

Ce travail est né d'une intention pédagogique:
- clarifier la présentation par [J.Gallier 86] d'algorithmes de démonstration automatique où il était difficile de séparer le système formel utilisé et le contrôle sur les règles, nécessaire pour rendre complet ces algorithmes,
- expliquer la méthode des tableaux avec variables libres exposée par [M.Fitting 90].

J'ai accompli cette tâche de formalisation avec les resultats suivants :
(1) preuve de la correction et de la complétude du système formel à la base de la méthode des tableaux,
(2) présentation des règles de la méthode des tableaux sans variables sous la forme de dérivation de sous-buts à partir d'un but avec l'interprétation simple qu'un but est insatisfaisable si et seulement si tous ses sous-buts sont insatisfaisables,
(3) séparation dans les algorithmes de démonstration automatique entre les règles et le contrôle des règles,
(4) analyse des règles utilisées par [M.Fitting 90] dans la méthode des tableaux avec variables libres qui a permis de montrer la correction de la delta-règle et de l'améliorer.

Au delà des résultats techniques de cette thèse, j'ai voulu contribuer au débat sur la méthodologie des démonstrateurs automatiques.

Perspectives

Il est clair que ce travail est un prélude à d'autres réalisations, dans le domaine de l'enseignement et de la recherche.

(1) Pour faciliter l'enseignement de la logique, il est intéressant de programmer des démonstrateurs semi-automatiques produisant des preuves lisibles (dans le calcul des séquents ou la déduction naturelle) des formules valides et quand c'est possible, des modèles de la négation des formules non valides.

(2) Il faut étendre les résultats obtenus aux démonstrateurs automatiques en logique modale, car la méthode des tableaux se prête sans difficultés à cette extension.

(3) La difficulté de la recherche en démonstration automatique est qu'elle mêle la logique (les règles), les algorithmes (le contrôle des règles) et les structures de données (la représentation des formules). Dans la méthode des matrices [L.A.Wallen 90] les trois aspects sont mélangés, je me propose d'analyser ce mélange pour comparer cette méthode et la méthode très voisine des tableaux avec variables libres.

Bibliographie

[P.B.Andrews 81] Theorem Proving via General Matings
Journal of the Association for Computing Machinery, Vol 28, N°2, April 1981, pp 193-214

[P.B.Andrews 86] An Introduction to mathematical logic and type theory : to truth through proof
Academic Press 1986

[P.B.Andrews 89] On Connections and Higher-Order Logic
Journal of Automated Reasoning **5**: 257-291, 1989

[A.Avron 87] Simple Consequence Relations
Technical Report (July 1, 1987), Laboratory for the foundations of computer Science, Edinburgh University, 1987

[E.Audureau, P.Enjalbert, L.Fariñas del Cerro 90]
Logique Temporelle
Sémantique et validation de programmes parallèles
Masson-1990

[Y.Auffray, P.Enjalbert 88]
Modal Theorem Proving using equational methods.
Les cahiers du Laboratoire d'Informatique de l'Université de Caen
Novembre 1988

[E.W.Beth 59] The Foundations of Mathematics
North-Holland (1959)

[W.W.Bledsoe 77] Non-resolution Theorem Proving.
Artificial Intelligence **9** (1977), 1-35

[G.S.Boolos, R.C.Jeffrey 74]
Computability and Logic
Cambridge University Press 1974

[R.S.Boyer, J.S.Moore 88]
A Computational Logic Handbook
Academic Press 1988

[R.Caferra 82] Abstraction, partage de structure et retour arriere non aveugle dans la méthode de réduction matricielle en démonstration automatique de théorèmes.
Thèse 1982
Université scientifique et médicale de Grenoble

[C.L.Chang, R.C.T.Lee 73]
Symbolic Logic and Mechanical Theorem Proving
Academic Press 1973

[R.L.Constable 86] Implementing Mathematics with the Nuprl Proof Development System
Prentice Hall 1986

[F.Delbart, P.Enjalbert, M.Lescot 90]
Multi-Modal Logic Programming using equational and order-sorted Logic
Les cahiers du Laboratoire d'Informatique de l'Université de Caen
Mars 1990

[S.Demri 90] Extension de démonstrateurs pour la logique classique à des logiques non classiques
Mémoire de D.E.A.
Institut national polytechnique de Grenoble 1990

[R.Dyckhoff 91] Contraction-free sequent calculi for intuitionistic logic
St Andrews University, St Andrews, Scotland
Draft.

[R.Dyckhoff 90] MacLogic
A proof assistant for first-logic on the Apple Macintosh
MALT project, St Andrews University, St Andrews, Scotland

[H.B.Enderton 72] A Mathematical Introduction to Logic
Academic Press 1972

[M.Fitting 83] Proof methods for modal and intuitionistic logics.
Volume 169 of Synthese library, D.Reidel, Dordrecht, Holland, 1983

[M.Fitting 88] First-Order Modal Tableaux.
Journal of Automated Reasoning **4** (1988) 191-214

[M.Fitting 90] First-Order logic and automated theorem proving
Springer Verlag 1990

[J.Gallier 86] Logic for Computer Science
Harper and Row 1986

[G.Gentzen 55] Recherches sur la déduction logique.
Presses Universitaires de France 1955

[J.Y.Girard, Y.Lafont, P.Taylor 90]
Proofs and Types
Cambridge University Press 1990

[M.J.C.Gordon 88] Programming Language Theory and its Implementation
Prentice Hall 1988

[J.Herbrand 68] Ecrits logiques
Presses universitaires de France 1968

[K.J.J.Hintikka 55] Form and Content in Quantification Theory,
Acta Philosophica Fennica, vol 8, pp 7-55 (1955)

[J.Hsiang 85] Refutational Theorem Proving using Term-Rewriting Systems
Artificial Intelligence **25** (1985) 255-300

[G.Huet 80] Confluent Reductions : Abstract Properties and Applications to Term Rewriting Systems
Journal of the Association for Computing Machinery, Vol 27, N° 4, October 1980 pp 797-821

[G.Huet, D.C.Oppen 80]
Equations and Rewrite Rules. A Survey
Formal Language Theory : Perspectives and Open Problems, ed. R.V.Book
Academic Press 1980

[S.Jeannicot, L.Oxusoff, A.Rauzy 88]
Evaluation sémantique : une propriété de coupure pour rendre efficace la procédure de Davis et Putman
Revue d'Intelligence Artificielle, Volume 3 - n°1/1988, p. 41-60

[T.Kaufl, N.Zabel 90] Cooperation of Decision Procedures in a Tableau-Based Theorem Prover
Revue d'Intelligence Artificielle, Volume 4 - n°3/1990, 99-125

[S.C.Kleene 64] Introduction to Metamathematics
North-Holland 1964

[R.Kowalski 75] A Proof Procedure Using Connection Graphs
Journal of the Association for Computing Machinery, Vol 22 - N°4, October 1975, pp. 441-449

[G.Kreisel, J.L.Krivine 67]
Eléments de logique mathématique. Théorie des modèles
Dunod 1967

[C.Kreitz 90] Towards a formal Theory of Program Construction
Revue d'Intelligence Artificielle, Volume 4 - n°3/1990, 53-79

[J.L.Krivine 90] Lambda-calcul. Types et modèles
Masson-1990

[R.Lalement 90] Logique réduction résolution
Masson-1990

[M.Levy, G.Vivier 69] Démonstration automatique en logique modale
Projet de fin d'études ENSIMAG 1969

[D.W.Loveland 78] Automated Theorem Proving : A Logical Basis
North-Holland 1978

[J.D.Monk 76] Mathematical Logic
Springer-Verlag 1976

[N.V.Murray 82] Completely Non-Clausal Theorem Proving
Artificial Intelligence **18** (1982) 67-85

[F.Oppacher, E.Suen 88]
HARP : A Tableau-Based Theorem Prover
Journal of Automated Reasoning **4** (1988) 69-100

[D.A.Plaisted 82] A Simplified Problem Reduction Format
Artificial Intelligence **18** (1982) 227-261

[D.A.Plaisted 88] Non-Horn Clause Logic Programming Without Contrapositives
Journal of Automated Reasoning **4** (1988) 287-325

[D.A.Plaisted 90] A Sequent-Style Model Elimination Strategy and a Positive Refinement
Journal of Automated Reasoning **6** (1990) 389-402

[S.V.Reeves 87] Adding Equality to Semantic Tableaux
Journal of Automated Reasoning **3** (1987) 225-246

[M.Rusinowitch 89] Démonstration automatique. Techniques de réécriture
InterEditions 1989

[H.Saya 75] Définition et utilisation des S-L graphes en démonstration automatique
Thèse
Institut national polytechnique de Grenoble 1975

[J.R.Shoenfield 67] Mathematical Logic
Addison-Wesley 1967

[M.E.Stickel 85] Automated Deduction by Theory Resolution
Journal of Automated Reasoning **1** (1985) 333-355

[M.E.Stickel 88] A Prolog Technology Theorem Prover : Implementation by an Extended Prolog Compiler.
Journal of Automated Reasoning **4** (1988) 353-380

[P.B.Thistlewaite, M.A.McRobbie, R.K.Meyer 88]
Automated Theorem-Proving in Non-Classical Logics
Pitman 1988

[D.van Dalen 80] Logic and Structure
Springer-Verlag 1983

[L.A.Wallen 90] Automated deduction in Nonclassical Logics
MIT Press 1990

[C.Walther 87] A Many-Sorted Calculus Based on Resolution and Paramodulation
Research Notes in Artificial Intelligence
Pitman 1987

[L.Wos, R.Overbeek, E.Lusk, J.Boyle 84]
Automated Reasoning. Introduction and Applications
Prentice Hall 1984

Titre : Contribution à l'analyse de la méthode des tableaux

Résumé : L'intérêt de la méthode des tableaux réside dans le fait que, contrairement aux méthodes usuelles de démonstration automatique par résolution, elle n'exige aucun prétraitement des formules.
Initialement créée pour des formules fermées, cette méthode a été généralisée par M.Fitting pour permettre l'utilisation de variables libres et d'algorithmes d'unification. Ceci l'a conduit à une reformulation des règles d'élimination des quantificateurs universels (gamma-règle) et des quantificateurs existentiels (delta-règle). La delta-règle décrite par M.Fitting apparait à mon avis insuffisamment prouvée. Aussi j'ai entrepris de l'analyser en présentant la méthode des tableaux comme un algorithme de construction de listes de buts à satisfaire.

Grâce à cette nouvelle présentation, j'ai pu :
(1) établir que la delta-règle de Fitting est en fait une skolémisation pour un quantificateur existentiel (resp. universel) ayant une occurrence positive (resp. négative) dans la liste des buts considérée comme une disjonction de conjonctions.
(2) améliorer la delta-règle de Fitting afin que le nouveau symbole de fonction ajouté par la règle ait le moins possible d'arguments.
(3) décrire concisément et précisément des classes d'algorithmes de démonstration automatique pour la méthode des tableaux avec ou sans variables libres.

Mots-clés : logique du premier ordre, démonstration automatique, méthode des tableaux