Fondements de l'informatique
Logique, modèles, et calculs

Chapitre: Démonstrations

Cours CSC_41012_EP

de l'Ecole Polytechnique

Olivier Bournez

bournez@lix.polytechnique.fr

Version du 16 juillet 2024

Démonstrations

L'objectif de ce chapitre est de commencer à aborder la question fondamentale suivante : qu'est-ce qu'une démonstration ?

Pour cela, plus précisément, on va se focaliser dans ce chapitre sur le problème suivant : on se donne une formule propositionnelle F, et on veut déterminer si F est une tautologie. Une tautologie est aussi appelée un *théorème*. On dit encore que F est *valide*.

Cela va nous amener à décrire des algorithmes particuliers.

1 Introduction

Une première méthode pour résoudre ce problème, qui est celle que nous avons utilisée dans le chapitre précédent, est la suivante : si F est de la forme $F(p_1, \cdots, p_n)$, on teste pour chacune des 2^n valuations v, i.e. pour les 2^n fonctions de $\{1, 2, \ldots, n\}$ dans $\{0, 1\}$, si v est bien un modèle de F. Si c'est le cas, alors F est une tautologie. Dans tout autre cas, F n'est pas une tautologie. Il est facile de programmer une telle méthode dans son langage de programmation favori.

La bonne nouvelle est que cette méthode existe : le problème de déterminer si une formule est une tautologie, est *décidable*, en utilisant la terminologie que nous verrons dans les chapitres suivants.

Remarque 1 *Cette observation peut sembler étrange et quelque part se contenter de peu, mais nous verrons que lorsque l'on considère des logiques plus générales, même très simples, cela devient problématique : il n'existe pas toujours d'algorithme pour déterminer si une formule F est une tautologie.*

Cependant, cette méthode est particulièrement inefficace. Elle a l'inconvénient majeur de garantir que lorsque F est une tautologie, on fera 2^n fois un test du type "la valuation v est-elle un modèle de F?" Lorsque n est grand, 2^n explose très vite : si l'on peut effectivement programmer la méthode, le programme obtenu sera en pratique inutilisable, car prenant un temps énorme, dès que l'on considèrera des formules F avec un grand nombre de variables.

Revenons alors à notre problème : on peut se dire que dans le raisonnement classique en mathématique, la méthode usuelle pour prouver qu'une assertion est un théorème est de la *démontrer*.

Si l'on veut faire mieux que la méthode exhaustive précédente, il y a essentiellement deux angles d'attaque. Le premier angle d'attaque est de chercher à s'approcher de la notion de *démonstration* dans le raisonnement usuel : des méthodes de preuve dans l'esprit de la section suivante apparaissent. Le deuxième angle d'attaque est de chercher à produire des algorithmes le plus efficace possible : des méthodes comme les *preuves par résolution* ou *méthodes par tableaux* plus algorithmiques apparaissent alors.

En général, on s'attend à ce qu'une méthode de preuve soit toujours *valide* : elle ne produit que des déductions correctes. Dans tous les cas se pose la question de la *complétude* : est-ce que tous les théorèmes (tautologies) sont prouvables ?

Dans ce qui suit, on présentera quatre systèmes de déduction valides et complets : les méthodes *à la Hilbert*, la *déduction naturelle*, la *méthode par résolution* et la *méthode des tableaux*. On ne prouvera la validité et la complétude que de la méthode des tableaux. A chaque fois nous noterons par $\vdash$ la notion de preuve sous-jacente : $T \vdash F$ signifie que la formule F se prouve à partir de l'ensemble de formules propositionnelles T. On note $\vdash T$ si $\emptyset \vdash T$.

A priori, il faudrait un symbole différent $\vdash$ pour chaque notion de démonstration.

Cependant, les théorèmes de complétude et de validité qui suivent prouvent qu'en fait à chaque fois, ce qui est prouvable pour chaque notion de démonstration est exactement la même chose, c'est-à-dire exactement les tautologies du calcul propositionnel.

En résumé : le symbole $\models$ et le symbole $\vdash$ désignent exactement la même notion : pour chacune des variantes de $\vdash$ évoquées dans ce qui suit, on a $\vdash F$ si et seulement si $\models F$, c'est-à-dire si et seulement si F est une tautologie.

2 Démonstrations à la Frege et Hilbert

Dans ce système de déduction, on part d'un ensemble d'axiomes, qui sont des tautologies, et on utilise une unique règle de déduction, le *modus ponens*, aussi appelé *coupure*, qui vise à capturer un type de raisonnement tout à fait naturel en mathématique.

La règle du modus ponens dit qu'à partir de la formule F et d'une formule $F \Rightarrow G$, on déduit G.

Graphiquement :

$$\frac{F \quad (F \Rightarrow G)}{G}$$

Exemple 1 *Par exemple, à partir de* $(A \wedge B)$ *et de* $(A \wedge B) \Rightarrow C$ *on déduit* C.

On considère alors un ensemble d'axiomes, qui sont en fait des instances d'un nombre fini d'axiomes.

Définition 1 (Instance) *On dit qu'une formule F est une instance d'une formule G si F s'obtient en substituant certaines variables propositionnelles de G par des*

formules F_i.

Exemple 2 *La formule $((C \Rightarrow D) \Rightarrow (\neg A \Rightarrow (C \Rightarrow D)))$ est une instance de $(A \Rightarrow (B \Rightarrow A))$, en prenant $(C \Rightarrow D)$ pour A, et $\neg A$ pour B.*

Définition 2 (Axiomes de la logique booléenne) *Un* axiome de la logique booléenne *est n'importe quelle instance d'une des formules suivantes :*

1. $(X_1 \Rightarrow (X_2 \Rightarrow X_1))$ *(axiome 1 pour l'implication) ;*
2. $((X_1 \Rightarrow (X_2 \Rightarrow X_3)) \Rightarrow ((X_1 \Rightarrow X_2) \Rightarrow (X_1 \Rightarrow X_3)))$ *(axiome 2 pour l'implication) ;*
3. $(X_1 \Rightarrow \neg\neg X_1)$ *(axiome 1 pour la négation) ;*
4. $(\neg\neg X_1 \Rightarrow X_1)$ *(axiome 2 pour la négation) ;*
5. $((X_1 \Rightarrow X_2) \Rightarrow (\neg X_2 \Rightarrow \neg X_1))$ *(axiome 3 pour la négation) ;*
6. $(X_1 \Rightarrow (X_2 \Rightarrow (X_1 \wedge X_2)))$ *(axiome 1 pour la conjonction) ;*
7. $((X_1 \wedge X_2) \Rightarrow X_1)$ *(axiome 2 pour la conjonction) ;*
8. $((X_1 \wedge X_2) \Rightarrow X_2)$ *(axiome 3 pour la conjonction) ;*
9. $(X_1 \Rightarrow (X_1 \vee X_2))$ *(axiome 1 pour la disjonction) ;*
10. $(X_2 \Rightarrow (X_1 \vee X_2))$ *(axiome 2 pour la disjonction) ;*
11. $((((X_1 \vee X_2) \wedge (X_1 \Rightarrow C)) \wedge (X_2 \Rightarrow C)) \Rightarrow C)$ *(axiome 3 pour la disjonction).*

On obtient une notion de démonstration.

Définition 3 (Démonstration par modus ponens) *Soit T un ensemble de formules propositionnelles, et F une formule propositionnelle. Une* preuve (par modus ponens) de F à partir de T *est une suite finie $F_1, F_2, \cdots, F_n$ de formules propositionnelles telle que F_n est égale à F, et pour tout i, ou bien F_i est dans T, ou bien F_i est un axiome de la logique booléenne, ou bien F_i s'obtient par modus ponens à partir de deux formules F_j, F_k avec $j < i$ et $k < i$.*

On note $T \vdash F$ si F est prouvable (par modus ponens) à partir de T. On note $\vdash F$ si $\emptyset \vdash F$, et on dit que F est *prouvable (par modus ponens).*

Exemple 3 *Soit F, G, H trois formules propositionnelles. Voici une preuve de $(F \Rightarrow H)$ à partir de $\{(F \Rightarrow G), (G \Rightarrow H)\}$:*

- $F_1 : (G \Rightarrow H)$ *(hypothèse) ;*
- $F_2 : ((G \Rightarrow H) \Rightarrow (F \Rightarrow (G \Rightarrow H)))$ *(instance de l'axiome 1.) ;*
- $F_3 : (F \Rightarrow (G \Rightarrow H))$ *(modus ponens à partir de F_1 et F_2) ;*
- $F_4 : ((F \Rightarrow (G \Rightarrow H)) \Rightarrow ((F \Rightarrow G) \Rightarrow (F \Rightarrow H)))$ *(instance de l'axiome 2.) ;*
- $F_5 : ((F \Rightarrow G) \Rightarrow (F \Rightarrow H))$ *(modus ponens à partir de F_3 et F_4) ;*
- $F_6 : (F \Rightarrow G)$ *(hypothèse) ;*

— $F_7 : (F \Rightarrow H)$ *(modus ponens à partir de F_6 et F_5).*

Exercice 1 *(corrigé page 236) Prouver* $(F \Rightarrow F)$.

Dans les exercices suivants, on pourra utiliser les exercices précédents pour résoudre chacune des questions.

Exercice 2 *(corrigé page 237) [Theorème de la déduction] Soit T une famille de formules propositionnelles, et F, G deux formules propositionnelles. Montrer que $T \vdash F \Rightarrow G$ est équivalent à $T \cup \{F\} \vdash G$.*

Exercice 3 *(corrigé page 237) Prouver les assertions suivantes :*
- *$T \cup \{F\} \vdash G$ est équivalent à $T \cup \{\neg G\} \vdash \neg F$.*
- *Si on a à la fois $T \vdash F$ et $T \vdash \neg F$, alors on a $T \vdash G$ pour toute formule G.*

Exercice 4 *(corrigé page 237) Prouver que $\{(\neg G \Rightarrow G)\} \vdash G$, pour toute formule G.*

Exercice 5 *(corrigé page 237) Prouver que si l'on a à la fois $T \cup \{F\} \vdash G$ et $T \cup \{\neg F\} \vdash G$ alors on a $T \vdash G$.*

Exercice 6 *Prouver les assertions suivantes :*
- $\{F\} \vdash \neg\neg F$
- $\{F, G\} \vdash F \vee G$
- $\{\neg F\} \vdash \neg(F \wedge G)$
- $\{\neg G\} \vdash \neg(F \wedge G)$
- $\{F\} \vdash F \vee G$
- $\{G\} \vdash F \vee G$
- $\{\neg F, \neg G\} \vdash \neg(F \vee G)$
- $\{\neg F\} \vdash (F \Rightarrow G)$
- $\{G\} \vdash (F \Rightarrow G)$
- $\{F, \neg G\} \vdash \neg(F \Rightarrow G)$

Exercice 7 *Pour v une fonction partielle de $\{X_i\}$ dans $\{0,1\}$, on pose*

$$T_V = \{X_i | v(X_i) = 1\} \cup \{\neg X_i | v(X_i) = 0\}.$$

Montrer que pour toute formule H dont les variables sont parmi le domaine de V, la relation $v \models H$ entraîne $T_V \vdash H$ et la relation $v \not\models H$ entraîne $T_V \vdash \neg H$.

Cette méthode de preuve est valide : en vérifiant que les axiomes sont des tautologies, il est facile de se convaincre par récurrence sur la longueur n de la preuve du résultat suivant :

Théorème 1 (Validité) *Toute formule propositionnelle prouvable est une tautologie.*

Ce qui est moins trivial, et plus intéressant est la réciproque : toute tautologie admet une preuve de ce type.

Théorème 2 (Complétude) *Toute tautologie est prouvable (par modus ponens).*

Nous ne donnerons pas la preuve de ce résultat, en la faisant correspondre à l'exercice suivant :

Exercice 8 *Prouver ce résultat en utilisant les exercices précédents : la clé est d'utiliser la possibilité de raisonner par cas (exercice 5) et l'exercice 7 qui fait le lien entre sémantique et syntaxe.*

On vient de décrire un système de déduction qui est très proche de la notion usuelle de preuve en mathématique. Cependant, ce système n'est pas facilement exploitable pour construire un algorithme qui déterminerait si une formule F est une tautologie.

Il est facile de s'en convaincre en tentant de faire les exercices précédents.

3 Démonstration par déduction naturelle

3.1 Règles de la déduction naturelle

La notion de démonstration précédente est en pratique difficile à utiliser. En effet, dans le système précédent, on se contraint quelquepart de garder les hypothèses tout au long de la démonstration. On ne peut donc pas facilement traduire une forme de raisonnement pourtant courante : nous voulons démontrer que $A \Rightarrow B$, supposons A et démontrons B sous cette hypothèse. Cette remarque mène à introduire une notion de couple formé d'un ensemble fini d'hypothèses, et d'une conclusion. Un tel couple est appelé un *séquent*.

On considère dans cette section que les propositions incluent aussi $\bot$, interprété par faux, et $\top$ interprété par vrai.

Définition 4 (Séquent) *Un* séquent *est un couple* $\Gamma \vdash A$*, où* Γ *est un ensemble fini de propositions et* A *est une proposition.*

Les règles de déduction de la déduction naturelle sont alors les suivantes :

$$\frac{}{\Gamma \vdash A}\ \text{axiome pour chaque } A \in \Gamma$$

$$\frac{}{\Gamma \vdash \top}\ \top\text{-intro}$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot\text{-élim}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\text{-intro}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\text{-élim}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\text{-élim}$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\text{-intro}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\text{-intro}$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee\text{-élim}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \Rightarrow\text{-intro}$$

$$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \Rightarrow\text{-élim}$$

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg\text{-intro}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash \neg A}{\Gamma \vdash \bot}\ \neg\text{-élim}$$

$$\frac{}{\Gamma \vdash A \vee \neg A}\ \text{tiers exclu}$$

Les règles $\top$-intro, $\wedge$-intro, $\vee$-intro, $\Rightarrow$-intro, $\neg$-intro, $\forall$-intro et $\exists$-intro sont appelées des *règles d'introduction* et les règles $\bot$-élim, $\wedge$-élim, $\vee$-élim, $\Rightarrow$-élim, $\neg$-élim, $\forall$-élim et $\exists$-élim des *règles d'élimination*. Les règles de la déduction naturelle sont donc classées en quatre groupes : les règles d'introduction, les règles d'élimination, la règle *axiome* et la règle *tiers exclu*.

Une *démonstration* d'un séquent $\Gamma \vdash A$ est une dérivation de ce séquent, c'est-à-dire un arbre dont les nœuds sont étiquetés par des séquents dont la racine est étiquetée par $\Gamma \vdash A$, et tel que si un nœud est étiqueté par un séquent $\Delta \vdash B$, alors ses enfants sont étiquetés par des séquents $\Sigma_1 \vdash C_1, \ldots, \Sigma_n \vdash C_n$ tels qu'il existe une règle de déduction naturelle, qui permet de déduire $\Delta \vdash B$ de $\Sigma_1 \vdash C_1, \ldots, \Sigma_n \vdash C_n$.

Un séquent $\Gamma \vdash A$ est donc démontrable s'il existe une démonstration de ce séquent.

3.2 Validité et complétude

On peut prouver les résultats suivants :

Théorème 3 (Validité) *Pour tout ensemble de formules* Γ *et pour toute formule* A*, si* $\Gamma \vdash A$ *est prouvable, alors* A *est* une conséquence *de* Γ.

Théorème 4 (Complétude) *Soit* Γ *un ensemble de formules. Soit* A *une formule qui est une* une conséquence *de* Γ.
Alors $\Gamma \vdash A$ *est prouvable.*

4 Démonstrations par résolution

Nous présentons brièvement la notion de preuve par résolution. Cette méthode de preuve est peut-être moins naturelle, mais est plus simple à implémenter informatiquement.

La résolution s'applique à une formule en forme normale conjonctive. Puisque toute formule propositionnelle peut se mettre sous forme normale conjonctive cela n'est pas restrictif.

Remarque 2 *Tout du moins en apparence. En effet, il faut être plus astucieux que dans le chapitre précédent pour transformer une formule en forme normale conjonctive, si l'on ne veut pas faire exploser la taille des formules et implémenter pratiquement la méthode.*

On appelle *clause* une disjonction de littéraux. Rappelons qu'un *littéral* est une variable propositionnelle ou sa négation. On convient de représenter une clause c par l'ensemble des littéraux sur lesquels porte la disjonction.

Exemple 4 *On écrit donc* $\{p, \neg q, r\}$ *plutôt que* $p \vee \neg q \vee r$.

Étant donné un littéral u, on note $\overline{u}$ pour le littéral équivalent à $\neg u$: autrement dit, si u est la variable propositionnelle p, $\overline{u}$ vaut $\neg p$, et si u est $\neg p$, $\overline{u}$ est p. Enfin on introduit une clause vide, notée $\square$, dont la valeur est 0 pour toute valuation.

Définition 5 (Résolvante) *Soient* C_1, C_2 *deux clauses. On dit que la clause* C *est une résolvante de* C_1 *et* C_2 *s'il existe un littéral* u *tel que :*
- $u \in C_1$;
- $\overline{u} \in C_2$;
- C *est donné par* $(C_1 \setminus \{u\}) \cup (C_2 \setminus \{\overline{u}\})$.

Exemple 5 *Les clauses* $\{p, q, r\}$ *et* $\{\neg r, s\}$ *admettent la résolvante* $\{p, q, s\}$.

Exemple 6 *Les clauses* $\{p, q\}$ *et* $\{\neg p, \neg q\}$ *admettent deux résolvantes, à savoir* $\{q, \neg q\}$ *et* $\{p, \neg p\}$. *Les clauses* $\{p\}$ *et* $\{\neg p\}$ *admettent la résolvante* $\square$.

Cela donne une notion de preuve.

Définition 6 (Preuve par résolution) *Soit* T *un ensemble de clauses. Une* preuve par résolution de T *est une suite finie* $F_1, F_2, \cdots, F_n$ *de clauses telle que* F_n *est égale à* $\square$, *et pour tout* i, *ou bien* F_i *est une clause dans* T, *ou bien* F_i *est une résolvante de deux clauses* F_j, F_k *avec* $j < i$ *et* $k < i$.

Remarque 3 *Le modus ponens, au coeur du système de preuve précédent, consiste à dire qu'à partir de la formule* F *et d'une formule* $(F \Rightarrow G)$, *on déduit* G. *Si l'on considère que la formule* $(F \Rightarrow G)$ *est équivalente à la formule* $(\neg F \vee G)$, *le modus ponens peut aussi se voir comme dire qu'à partir de la formule* F *et d'une formule* $(\neg F \vee G)$, *on déduit* G, *ce qui ressemble au concept de résolvante : la résolvante de* $\{f\}$ *et de* $\{\neg f, g\}$ *est* $\{g\}$.

D'une certaine façon, la résolvante est un modus ponens généralisé, même si cette analogie n'est qu'une analogie et une preuve dans un système de preuve ne peut pas se traduire directement dans l'autre.

Exercice 9 *Prouver par résolution*

$$T = \{\{\neg p, \neg q, r\}, \{\neg p, \neg q, s\}, \{p\}, \{\neg s\}, \{q\}, \{t\}\}.$$

Cette méthode de preuve est valide (sens facile).

Théorème 5 (Validité) *Toute clause apparaissant dans une preuve par résolution de* T *est une conséquence de* T.

En fait, pour prouver une formule, on raisonne en général dans cette méthode de preuve plutôt sur sa négation, et on cherche à prouver que sa négation est contradictoire avec les hypothèses. La validité se formule alors en général plutôt de cette façon.

Corollaire 1 (Validité) *Si un ensemble de clauses* T *admet une preuve par résolution, alors* T *est contradictoire.*

Elle s'avère complète (sens plus difficile).

Théorème 6 (Complétude) *Soit* T *un ensemble de clauses contradictoires. Il admet une preuve par résolution.*

5 Démonstrations par la méthode des tableaux

Nous avons jusque-là uniquement évoqué des systèmes de déduction valides et complets, parfois sans fournir aucune preuve de nos théorèmes. Nous allons étudier plus complètement la méthode des tableaux. Nous avons choisi de développer cette méthode, car elle est très algorithmique et basée sur la notion d'arbre, ce qui appuiera notre argumentaire récurrent sur le fait que la notion d'arbre est partout en informatique.

5.1 Principe

Pour simplifier la discussion, on considèrera que les formules propositionnelles ne sont écrites qu'avec les connecteurs $\neg, \wedge, \vee, \Rightarrow$. La formule $(F \Leftrightarrow G)$ sera considérée comme une abréviation de la formule $((F \Rightarrow G) \wedge (G \Rightarrow F))$.

Supposons que l'on veuille prouver qu'une formule F est une tautologie. Si la formule F est de la forme $(F_1 \wedge F_2)$, alors on peut chercher à prouver F_1 et F_2, et écrire F_1, F_2. Si la formule est de la forme $(F_1 \vee F_2)$, alors on peut explorer deux possibilités, l'une pour explorer le cas de F_1 et l'autre pour le cas de F_2.

On va ramener toutes les autres possibilités à ces deux configurations, en utilisant les lois de Morgan, et quelques équivalences : si la formule F est de la forme $(F_1 \Rightarrow F_2)$, on va la voir comme $(F_2 \vee \neg F_1)$ et appliquer la règle du $\vee$, et si F est de la forme $\neg(F_1 \Rightarrow F_2)$, comme $(F_1 \wedge \neg F_2)$ et appliquer la règle du $\wedge$. On traite chacun des autres cas à l'aide des lois de Morgan.

En faisant cela de façon systématique, on va construire un arbre, dont la racine est étiquetée par la négation de la formule F : autrement dit, pour prouver une formule F, la méthode commence par la négation de la formule F.

Partons par exemple de la formule F suivante que nous cherchons à prouver :

$$(((p \wedge q) \Rightarrow r) \Rightarrow ((p \Rightarrow r) \vee (q \Rightarrow r))).$$

On part donc de la formule $\neg F$, c'est-à-dire de

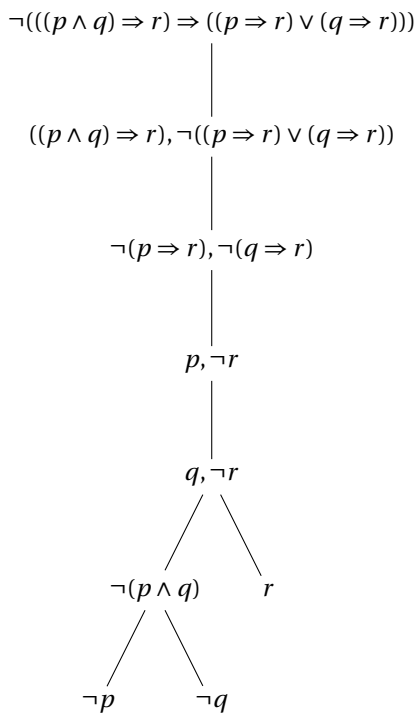
$$\neg(((p \wedge q) \Rightarrow r) \Rightarrow ((p \Rightarrow r) \vee (q \Rightarrow r))).$$

— en transformant l'implication $\neg(F_1 \Rightarrow F_2)$ en la formule équivalente $(F_1 \wedge \neg F_2)$, on obtient $((p \wedge q) \Rightarrow r) \wedge \neg((p \Rightarrow r) \vee (q \Rightarrow r))$ pour se ramener à la règle du $\wedge$.
— On applique alors la règle du $\wedge$: on considère les formules $((p \wedge q) \Rightarrow r)$ et $\neg((p \Rightarrow r) \vee (q \Rightarrow r))$.
— On considère à l'instant d'après la dernière formule, que l'on peut voir par les lois de Morgan comme $(\neg(p \Rightarrow r) \wedge \neg(q \Rightarrow r))$: on lui associe $\neg(p \Rightarrow r)$ et $\neg(q \Rightarrow r)$ par la règle du $\wedge$.
— On considère alors $\neg(p \Rightarrow r)$, ce qui donne les formules p et $\neg r$.
— On obtient alors q et $\neg r$ à partir de $\neg(q \Rightarrow r)$.
— Si l'on considère maintenant $((p \wedge q) \Rightarrow r)$, que l'on peut voir comme $(r \vee \neg(p \wedge q))$, on a le choix par la règle du $\vee$ entre la formule $\neg(p \wedge q)$ ou r.
— Le cas de r est exclus par l'étape précédente, où l'on avait $\neg r$.

— Dans le premier cas, on a encore le choix entre $\neg p$ ou $\neg q$. Les deux cas sont exclus parce que l'on avait avant p et q.

Puisque toutes les branches mènent à une contradiction, il n'y a aucune situation dans laquelle F pourrait être fausse : on sait alors que F est une tautologie.

Le calcul que l'on vient de faire se représente naturellement par un arbre.

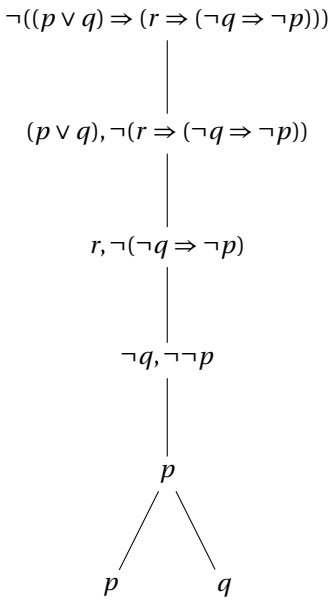


Chaque branche correspond à un scénario possible. Si une branche possède un sommet étiqueté par une formule A telle que $\neg A$ apparaît plus haut sur la même branche (ou l'opposé), alors on cesse de développer cette branche, et on dit que la branche est *close* : cela veut dire qu'il y a une contradiction. Si toutes les branches sont closes, alors on dit que *l'arbre est clos*, et on sait que tous les scénarios sont exclus.

Considérons maintenant l'exemple de la formule G donnée par

$$((p \vee q) \Rightarrow (r \Rightarrow (\neg q \Rightarrow \neg p))).$$

Avec la même méthode, on développe un arbre avec comme racine $\neg G$.



Cette fois l'arbre a deux branches. La branche de droite est close. La branche de gauche n'est pas close, et les variables propositionnelles sur cette branche sont $r, \neg q$ et p. Si l'on prend une valuation v telle que $v(r) = 1$, $v(q) = 0$, $v(p) = 1$, la valuation donne la valeur 1 à $\neg G$, et donc 0 à G. Autrement dit, G n'est pas une tautologie. On dit que l'arbre est *ouvert*.

5.2 Description de la méthode

Un *tableau* est un arbre binaire étiqueté dont les sommets sont des ensembles de formules propositionnelles et qui est construit récursivement à partir de la racine sommet par sommet en utilisant un nombre fini de fois deux types de règles : les règles α et les règles β.

On rappelle que pour simplifier la discussion, on considère que les formules propositionnelles écrites qu'avec les connecteurs $\neg, \wedge, \vee, \Rightarrow$. La formule $(F \Leftrightarrow G)$ est considérée comme une abréviation de la formule $((F \Rightarrow G) \wedge (G \Rightarrow F))$.

Les formules sont découpées en deux classes, la classe α et la classe β, et à chaque formule on associe inductivement deux autres formules par les règles suivantes :

— Les formules de type α sont les formules du type :

1. $\alpha = (A \wedge B)$: on lui associe $\alpha_1 = A$ et $\alpha_2 = B$.
2. $\alpha = \neg(A \vee B)$: on lui associe $\alpha_1 = \neg A$ et $\alpha_2 = \neg B$.
3. $\alpha = \neg(A \Rightarrow B)$: on lui associe $\alpha_1 = A$ et $\alpha_2 = \neg B$.
4. $\neg\neg A$: on lui associe $\alpha_1 = \alpha_2 = A$.

— Les formules du type β sont les formules du type :

1. $\beta = \neg(A \wedge B)$: on lui associe $\beta_1 = \neg A$, $\beta_2 = \neg B$.
2. $\beta = (A \vee B)$: on lui associe $\beta_1 = A$, $\beta_2 = B$.
3. $\beta = (A \Rightarrow B)$: on lui associe $\beta_1 = \neg A$, $\beta_2 = B$.

Si B est une branche d'un tableau, on note $\bigcup B$ pour l'ensemble des formules qui apparaissent sur un sommet de B.

Les deux règles de construction d'un tableau sont les suivantes :

1. une règle α consiste à prolonger une branche finie d'un tableau T par le sommet étiqueté $\{\alpha_1, \alpha_2\}$, où α est une formule de type α qui apparaît sur un sommet de B.
2. une règle β consiste à prolonger une branche finie d'un tableau T par deux fils étiquetés respectivement par $\{\beta_1\}$ et $\{\beta_2\}$, où β est une formule de type β qui apparaît sur un sommet de B.

Remarque 4 *Observons que ce n'est pas nécessairement le dernier sommet d'une branche B qui est développé à chaque étape, mais une formule quelque part sur la branche.*

On dit qu'une branche B est *close* s'il existe une formule A telle que A et $\neg A$ apparaissent sur la branche B. Dans le cas contraire la branche est dite *ouverte*.

Une branche B est *développée* si

1. pour toute formule de type α de $\bigcup B$, $\alpha_1 \in \bigcup B$ et $\alpha_2 \in \bigcup B$.
2. pour toute formule de type β de $\bigcup B$, $\beta_1 \in \bigcup B$ ou $\beta_2 \in \bigcup B$.

Un tableau est *développé* si toutes ses branches sont soit closes, soit développées. Un tableau est *clos* si toutes ses branches sont closes. Un tableau est *ouvert* s'il possède une branche ouverte.

Finalement, un tableau pour une formule A (respectivement pour un ensemble de formules Σ) est un tableau dont la racine est étiquetée par $\{A\}$ (respectivement par $\{A | A \in \Sigma\}$).

5.3 Terminaison de la méthode

Observons tout d'abord que l'on peut toujours appliquer des règles α ou β jusqu'à obtenir un tableau développé.

Proposition 1 *Si Σ est un ensemble fini de formules, alors il existe un tableau développé (et fini) pour Σ.*

Démonstration: Cela se prouve par récurrence sur le nombre n d'éléments de Σ.

Pour le cas $n = 1$, observons que la longueur des formules α_1, α_2, β_1, et β_2 est toujours inférieure strictement à la longueur de α et β. Le processus d'extension des branches qui ne sont pas closes se termine donc nécessairement après un nombre fini d'étapes. Le tableau obtenu au final est développé, sinon il serait possible de l'étendre.

Pour le cas $n > 1$, écrivons $\Sigma = \{F_1, \cdots, F_n\}$. Considérons par hypothèse de récurrence un tableau développé pour $\Sigma = \{F_1, \cdots, F_{n-1}\}$. Si ce tableau est clos ou si F_n

est une variable propositionnelle, alors ce tableau est un tableau développé pour Σ. Sinon, on peut étendre toutes les branches ouvertes en appliquant les règles correspondantes à la formule F_n, et en développant les branches obtenues : le processus termine pour la même raison que pour $n = 1$. □

Bien entendu, à partir d'une racine donnée, il y a de nombreuses façons de construire un tableau développé.

5.4 Validité

La méthode précédente donne une méthode de preuve.

Définition 7 *On dira qu'une formule F est* prouvable par tableau *s'il existe un tableau clos avec la racine $\{\neg F\}$. On notera dans ce cas $\vdash F$.*

Exercice 10 *Prouver que A est une conséquence de $((A \vee \neg B) \wedge B)$ par la méthode des tableaux, i.e. $\vdash (((A \vee \neg B) \wedge B) \Rightarrow A)$.*

Exercice 11 *Prouver que $\neg C$ est une conséquence de $((H \wedge (P \vee C)) \Rightarrow A) \wedge H \wedge \neg A \wedge \neg P)$ par la méthode des tableaux.*

La méthode est valide.

Théorème 7 (Validité) *Toute formule prouvable est une tautologie.*

Démonstration: On dira qu'une branche B d'un tableau est *réalisable* s'il existe une valuation v telle que $v(A) = 1$ pour toute formule $A \in \bigcup B$ et $v(A) = 0$ si $\neg A \in \bigcup B$. Un tableau est dit *réalisable* s'il a une branche réalisable.

Il suffit de prouver le résultat suivant :

Lemme 1 *Soit T' une extension immédiate du tableau T : c'est-à-dire le tableau obtenu en appliquant une règle α ou une règle β à T. Si T est réalisable alors T' aussi.*

Ce lemme suffit à prouver le théorème : si F est prouvable, alors il y a un tableau clos avec $\neg F$ comme racine. Cela veut dire que pour toute branche, il y a une formule A telle que A et $\neg A$ apparaissent sur cette branche, et donc aucune des branches de T n'est réalisable. Par le lemme, cela veut dire que l'on est parti initialement d'un arbre réduit au sommet étiqueté par $\neg F$ qui n'était pas réalisable. Autrement dit, F est une tautologie.

Il reste à prouver le lemme. Soit B une branche réalisable de T, et soit B' la branche de T qui est étendue dans T'. Si $B \neq B'$, alors B reste une branche réalisable de T. Si $B = B'$, alors B est étendue dans T',

1. soit en une branche B_α par l'application d'une règle α ;

2. soit en deux branches B_{β_1} et B_{β_2} par l'application d'une règle β.

Dans le premier cas, soit α la formule utilisée par la règle, et v une valuation réalisant B : de $v(\alpha) = 1$, on déduit $v(\alpha_1) = 1$ et $v(\alpha_2) = 1$: donc v est une valuation qui réalise B_α et le tableau T' est réalisable.

Dans le second cas, soit β la formule utilisée par la règle : de $v(\beta) = 1$, on déduit qu'au moins une des valeurs $v(\beta_1)$ et $v(\beta_2)$ vaut 1 : donc v réalise l'une des branches B_{β_1} et B_{β_2}, et le tableau T' est réalisable. □

5.5 Complétude

La méthode est complète : en d'autres termes, la réciproque du théorème précédent est vraie.

Théorème 8 (Complétude) *Toute tautologie est prouvable.*

Corollaire 2 *Soit F une formule propositionnelle.*
F est une tautologie si et seulement si elle est prouvable.

Le reste de cette sous-section est consacré à prouver ce théorème.

Remarquons tout d'abord que si B est une branche à la fois développée et ouverte d'un tableau T, alors l'ensemble $\bigcup B$ de formules qui apparaissent dans B a les propriétés suivantes :

1. il n'y a aucune variable propositionnelle p telle que $p \in \bigcup B$ et telle que $\neg p \in \bigcup B$;
2. pour toute formule $\alpha \in \bigcup B$, $\alpha_1 \in \bigcup B$ et $\alpha_2 \in \bigcup B$;
3. pour toute formule $\beta \in \bigcup B$, $\beta_1 \in \bigcup B$ ou $\beta_2 \in \bigcup B$.

Lemme 2 *Toute branche développée et ouverte d'un tableau est réalisable.*

Démonstration: Soit B une branche développée et ouverte d'un tableau T. On définit une valuation v par :

1. si $p \in \bigcup B$, $v(p) = 1$;
2. si $\neg p \in \bigcup B$, $v(p) = 0$;
3. si $p \notin \bigcup B$ et $\neg p \notin \bigcup B$, on pose (arbitrairement) $v(p) = 1$.

On montre par induction structurelle sur A que : si $A \in \bigcup B$, alors $v(A) = 1$, et si $\neg A \in \bigcup B$, alors $v(A) = 0$.

En effet, c'est vrai pour les variables propositionnelles.

Si A est une formule α, alors $\alpha_1 \in \bigcup B$ et $\alpha_2 \in \bigcup B$: par hypothèse d'induction, $v(\alpha_1) = 1$, $v(\alpha_2) = 1$, et donc $v(\alpha) = 1$.

Si A est une formule β, alors $\beta_1 \in \bigcup B$ ou $\beta_2 \in \bigcup B$: par hypothèse d'induction, $v(\beta_1) = 1$ ou $v(\beta_2) = 1$, et donc $v(\beta) = 1$. □

Proposition 2 *S'il existe un tableau clos avec $\neg A$ comme racine, alors tout tableau développé avec la racine $\neg A$ est clos.*

Démonstration: Par l'absurde : soit T un tableau ouvert et développé avec la racine $\neg A$, et B une branche ouverte de T. Par le lemme précédent, B est réalisable, et puisque $\neg A$ est dans B, $\neg A$ est satisfiable. A n'est donc pas une tautologie, et donc n'est pas prouvable par tableau : il n'y a donc pas de tableau clos avec la racine $\neg A$. □

On a enfin tous les ingrédients pour prouver le théorème 8.

Supposons que A ne soit pas prouvable par tableau, et soit T un tableau développé avec la racine $\neg A$: T n'est pas clos. Comme dans la preuve précédente, si B est une branche ouverte de T, alors B est réalisable, et donc $\neg A$ est satisfiable : autrement dit, A n'est pas une tautologie.

5.6 Une conséquence du théorème de compacité

Définition 8 *On dira qu'un ensemble Σ de formules est* réfutable par tableau *s'il existe un tableau clos avec la racine Σ.*

Corollaire 3 *Tout ensemble Σ de formules qui n'est pas satisfiable est réfutable par tableau.*

Démonstration: Par le théorème de compacité, un ensemble de formules Σ qui n'est pas satisfiable possède un sous-ensemble fini Σ_0 qui n'est pas satisfiable. Cet ensemble fini de formules a une réfutation par tableau, i.e. il y a un tableau clos avec la racine Σ_0. Ce tableau donne aussi un tableau clos avec la racine Σ. □

6 Notes bibliographiques

Lectures conseillées Pour aller plus loin sur les notions évoquées dans ce chapitre, nous suggérons la lecture de [Cori & Lascar, 1993], de [Mendelson, 1987] pour les méthodes de preuve basées sur le modus ponens, de [Stern, 1994] pour une présentation simple des méthodes de preuve basées sur la résolution, et pour la méthode des tableaux nous renvoyons à [Lassaigne & de Rougemont, 2004] et [Nerode & Shore, 1997].

Bibliographie Ce chapitre a été rédigé en s'inspirant essentiellement du livre [Cori & Lascar, 1993], de [Dehornoy, 2006] pour la partie sur les méthodes preuves basées sur le modus ponens, de [Stern, 1994] pour la présentation de la preuve par résolution. La partie sur la déduction naturelle est prise de [Dowek, 2008]. La section sur la méthode des tableaux est reprise de [Lassaigne & de Rougemont, 2004].

Index

Bibliographie

[Cori & Lascar, 1993] Cori, R. & Lascar, D. (1993). *Logique mathématique. Volume I*. Masson.

[Dehornoy, 2006] Dehornoy, P. (2006). *Logique et théorie des ensembles*. Notes de cours.

[Dowek, 2008] Dowek, G. (2008). *Les démonstrations et les algorithmes*. Polycopié du cours de l'Ecole Polytechnique.

[Lassaigne & de Rougemont, 2004] Lassaigne, R. & de Rougemont, M. (2004). *Logic and complexity*. Discrete Mathematics and Theoretical Computer Science. Springer. https://doi.org/10.1007/978-0-85729-392-3

[Mendelson, 1987] Mendelson, E. (1987). *Introduction to mathematical logic (3. ed.)*. Chapman and Hall.

[Nerode & Shore, 1997] Nerode, A. & Shore, R. A. (1997). *Logic for Applications, Second Edition*. Graduate Texts in Computer Science. Springer. https://doi.org/10.1007/978-1-4612-0649-1

[Stern, 1994] Stern, J. (1994). Fondements mathématiques de l'informatique. *Ediscience International, Paris*.