Logique : notes de cours

Jean-Pierre Jouannaud

12 février 2005

Table des matières

Chapitre 1

Logique du premier ordre

1.1 Syntaxe

1.1.1 Vocabulaire

Un langage du premier ordre est constitué à partir d'un vocabulaire comprenant

1. un ensemble (supposé infini sauf mention express du contraire) de *symboles de variables* $\mathcal{X} = \{x, y, z, \ldots\}$,
2. un ensemble de *symboles de fonctions* $\mathcal{F} = \{f, g, h, \ldots, a, b, c, \ldots\}$ munis d'*arités* tel que $\mathcal{F}_n$ est le sous-ensemble des symboles de fonction d'arité n,
3. un ensemble de *symboles de prédicats* $\mathcal{P} = \{P, Q, R, \ldots, p, q, r, \ldots\}$ munis d'arités tel que $\mathcal{P}_n$ est le sous-ensemble des symboles de prédicats d'arité n,
4. des *connecteurs logiques* usuels : $\wedge, \vee, \neg, \Rightarrow$, des constantes $0, 1$, et des quantificateurs $\exists,\ \forall$. On utilisera également des symboles auxiliaires de désambiguation comme $(,), [,], \{$ et $\}$.

1.1.2 Formules logiques

Le langage logique est formé de quatre entités syntaxiques, les symboles de variables (ou *variables*), les *termes*, les *atomes* et les *formules*, construites successivement comme suit :

1. l'ensemble $T(\mathcal{F}, \mathcal{X})$ des termes finis est défini comme le plus ensemble tel que
   - toute variable est un terme ;
   - si f est un symbole de fonction d'arité n et $t_1, \ldots t_n$ sont des termes, alors $f(t_1, \ldots, t_n)$ est un terme.

   Exercice 1.1 *Montrer que l'ensemble des termes représentés comme des mots sur l'alphabet $\mathcal{X} \cup \mathcal{F}$ est un langage hors-contexte lorsque $\mathcal{F}$ et $\mathcal{X}$ sont finis.*

   Un terme est dit clos s'il ne contient pas de variable. On notera par $T(\mathcal{F})$ l'ensemble des termes clos sur le vocabulaire $\mathcal{F}$. On représentera sovent les termes par des arbres étiquettés. La *profondeur* d'un terme ($t \in T(\mathcal{F}, \mathcal{X})$) est la longueur de sa plus longue branche. Sa *taille* est celle de l'arbre associé. Si $t \in T(\mathcal{F}, \mathcal{X})$, $Var(t)$ désigne l'ensemble des variables de t. Enfin, l'égalité "syntaxique" entre termes est notée $\equiv$.

Les mathématiciens ont pour habitude de remplacer les parenthèses par des règles d'association et de priorité qui permettent de désambigüer les formules, donc de faire leur analyse syntaxique. On supposera ainsi que tous les symboles associent à gauche, et que leur ordre de priorité décroissant est donné par $\neg > \wedge > \vee > \Rightarrow$.

2. L'ensemble des atomes est l'ensemble des expressions de la forme $P(t_1, \ldots, t_n)$ où P est un symbole de prédicat et $t_1, \ldots, t_n$ sont des termes.
3. Le langage $\mathcal{F}(\mathcal{R}, \mathcal{F}, \mathcal{X})$ des *formules du premier ordre* bati sur le vocabulaire $\mathcal{R}, \mathcal{F}, \mathcal{X}$ est le plus petit ensemble contenant les constantes logiques 0 et 1 et les atomes, et clos par les opérations qui

   - à la formule A associent sa négation $\neg A$,

   - aux formules A, B associent leur conjonction $A \wedge B$, leur disjonction $A \vee B$, et l'implication $A \Rightarrow B$,

   - à la formule A et à la variable x associent la formule *existenciellement quantifiée* $\exists x A$, et la formule *universellement quantifiée* $\forall x A$.

Le langage des *formules propositionnelles* est obtenu pour $\mathcal{X} = \emptyset, \mathcal{F} = \emptyset, \mathcal{P} = \mathcal{P}_0$.

Un *littéral* est un atome (auquel cas il est dit *positif*) ou la négation d'un atome (auquel cas il est dit *négatif*). Une formule sans variable est dite *close*. Une formule dont les quantificateurs apparaissent tous en tête est dite *prénexe*. Une formule *universelle* (resp. *existentielle*) est une formule prénexe close dont les seuls quantificateurs sont universels (resp. existentiels).

On appelle *clause* toute formule universelle dont les sous formules non quantifiées sont des littéraux ou des disjonctions de littéraux. Une clause a donc la forme générale

$$\forall x_1 \ldots \forall x_n A_1 \vee \ldots \vee A_p \vee \neg B_1 \vee \ldots \vee \neg B_q$$

où $A_1, \ldots, A_p, B_1, \ldots, B_q$ désignent des atomes.

Exercice 1.2 *Montrer que l'ensemble des formules et des formules prénexes représentés comme des mots sur l'alphabet $\mathcal{X} \cup \mathcal{F} \cup \mathcal{P}$ est un langage hors-contexte lorsque $\mathcal{F}, \mathcal{P}$ et X sont finis.*

Qu'en est-il de l'ensemble des clauses ?

1.1.3 Variables libres et variables liées

Les quantificateurs *lient* les variables sur lesquelles ils portent. On définit donc les variables *libres* $\mathcal{V}ar()$ et les variables *liées* $\mathcal{BV}ar()$ d'une formule par récurrence sur la structure de la formule :

Variables $\mathcal{V}ar(x) = \{x\}$ si x est une variable
$\mathcal{BV}ar(x) = \emptyset$ si x est une variable
Termes $\mathcal{V}ar(f(t_1, \ldots, t_n)) = \mathcal{V}ar(t_1) \cup \ldots \cup \mathcal{V}ar(t_n)$
$\mathcal{BV}ar(f(t_1, \ldots, t_n)) = \emptyset$
Atomes $\mathcal{V}ar(P(t_1, \ldots, t_n)) = \mathcal{V}ar(t_1) \cup \ldots \cup \mathcal{V}ar(t_n)$
$\mathcal{BV}ar(P(t_1, \ldots, t_n)) = \emptyset$
Formules $\mathcal{V}ar(A \wedge B) = \mathcal{V}ar(A) \cup \mathcal{V}ar(B)$
$\mathcal{BV}ar(A \wedge B) = \mathcal{BV}ar(A) \cup \mathcal{BV}ar(B)$
$\mathcal{V}ar(A \vee B) = \mathcal{V}ar(A) \cup \mathcal{V}ar(B)$

$\mathcal{BV}ar(A \vee B) = \mathcal{BV}ar(A) \cup \mathcal{BV}ar(B)$
$\mathcal{V}ar(A \Rightarrow B) = \mathcal{V}ar(A) \cup \mathcal{V}ar(B)$
$\mathcal{BV}ar(A \Rightarrow B) = \mathcal{BV}ar(A) \cup \mathcal{BV}ar(B)$
$\mathcal{V}ar(\neg A) = \mathcal{V}ar(A)$
$\mathcal{BV}ar(\neg A) = \mathcal{BV}ar(A)$
$\mathcal{V}ar(\exists x A) = \mathcal{V}ar(A) - \{x\}$
$\mathcal{BV}ar(\exists x A) = \mathcal{BV}ar(A) \cup \{x\}$
$\mathcal{V}ar(\forall x A) = \mathcal{V}ar(A) - \{x\}$
$\mathcal{BV}ar(\forall x A) = \mathcal{BV}ar(A) \cup \{x\}$

1.1.4 Positions

On représente souvent les formules comme des arbres, et on utilise des mots sur le vocabulaire des entiers naturels pour désigner les positions des sous formules. Cela permet de parler plus précisément des occurrences libres ou liées d'une variable x. Par exemple, la variable x a une occurrence libre (à la position 1.1) et une occurence liée (à la position 2.1.1) dans la formule $P(x) \vee \forall x Q(x)$, à condition de considérer $\forall x$ comme un opérateur unaire indexé par la variable x. Cela a un sens informatique très précis : la variable liée x est représentée par un pointeur sur le noeud correspondant. Deux occurrences d'une même variable liées par le même quantificateur ($\forall$ ou $\exists$) pointeront donc sur le même noeud (étiquetté par $\forall$ ou $\exists$). Deux variables distinctes ou bien liées par des quantificateurs différents pointeront sur des noeuds distincts. Les variables libres pointeront sur des noeuds ($\exists$ ou $\forall$ selon les cas) fictivement rajoutés au dessus du terme. Cette représentation des termes est due à De Bruijn.

Formellement, soit $\mathbf{N}$ l'ensemble des entiers naturels, $\mathbf{N}_+$ l'ensemble des entiers naturels non nuls, $\mathbf{N}_+^*$ l'ensemble des suites (ou séquences) finies d'entiers naturels non nuls (ou des mots sur le vocabulaire $\mathbf{N}_+$), et $\Lambda \in \mathbf{N}_+^*$ la suite (ou mot) vide. Chaque $i \in \mathbf{N}_+$ est identifié à la séquence contenant l'unique élément i. Si $p, q \in \mathbf{N}_+^*$, $p \cdot q$ dénote leur concaténation, opération interne dont la séquence vide est élément neutre à droite et à gauche. On comparera des positions par l'*ordre préfixe* : $q \leq_{pref} p$ s'il existe une position r telle que $p = q \cdot r$.

On appelle *ensemble de positions* tout ensemble fini $P \subseteq \mathbf{N}_+^*$ clos par l'ordre préfixe et sans trou : $\forall p \cdot i \in P,\ p \in P$ et $p \cdot prec(i) \in P$ si $i > 1$, où $prec(i)$ désigne le prédecesseur de i.

Étant donnée une signature F, un terme sur cette signature peut être vu comme un *arbre étiqueté*, c'est-à-dire une application t d'un ensemble de positions noté $Pos(t)$ dans $\mathcal{F}$, qui respecte l'arité des symboles de fonction :

- $t(p) \in \mathcal{F}_0$ ssi p est une feuille de $Pos(t)$
- Si $t(p) = f$ d'arité n, alors $\forall i \in \mathbf{N}$, $p \cdot i \in Pos(t) \ \Leftrightarrow 1 \leq i \leq n$.

On notera par $\phi|_p$ la sous-formule de ϕ à la position p, et par $\phi[\psi]_p$ la formule obtenue en remplçant $\phi|_p$ par ψ dans ϕ. Bien entendu, il faudra toujours respecter la catégorisation en termes et formules de manière à construire des objets bien formés.

1.1.5 Substitutions

On appelle substitution toute application de l'ensemble des variables dans les termes. On utilisera une notation postfixée pour leur application. On appelle *domaine* de la substitution σ l'ensemble $\mathcal{D}om(\sigma) = \{x \in \mathcal{X} \mid x\sigma \neq x\}$. Une substitution de domaine fini sera noté en extension, sous la forme $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$. On notera par $\sigma_{|X}$ la restriction de σ au domaine $\mathcal{D}om(\sigma) \cap X$. On appelle *image* d'une substitution σ l'ensemble $\Im(\sigma) = \bigcup_{x \in \mathcal{D}om(\sigma)} \mathcal{V}ar(x\sigma)$. L'application d'une substitution est étendue successivement aux termes, aux atomes, puis aux formules :

Variables $x\sigma = \sigma(x)$
Termes $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$
Atomes $P(t_1, \ldots, t_n)\sigma = P(t_1\sigma, \ldots, t_n\sigma)$
Formules $(A \wedge B)\sigma = A\sigma \wedge B\sigma$
$(A \vee B)\sigma = A\sigma \vee B\sigma$
$(A \Rightarrow B)\sigma = A\sigma \Rightarrow B\sigma$
$(\neg A)\sigma = \neg(A\sigma)$
$(\exists x A)\sigma = \exists y(A\sigma')$ where $\sigma' = \sigma_{|\neq x} \cup \{x \mapsto z\}$ pour $z \notin \Im(\sigma)$
$(\forall x A)\sigma = \forall y(A\sigma')$ where $\sigma' = \sigma_{|\neq x} \cup \{x \mapsto z\}$ pour $z \notin \Im(\sigma)$

1.2 Sémantique

La sémantique de Tarski a pour but d'associer une valeur de vérité parmi (T pour *true* et F pour *false*) à toute formule close, et une fonction à valeur dans les valeurs de vérité à toute formule possédant des variables libres. Certaines sémantiques utilisent trois valeurs de vérité (en ajoutant la valeur *indéfini* par exemple). Nous avons fait attention de distinguer les noms dans la syntaxe et la sémantique. Ainsi 0 est une constante logique de la syntaxe, alors que F est une valeur du domaine sémantique.

1.2.1 Sémantique des connecteurs logiques

On munit les valeurs de vérité d'une structure via une table de vérité qui décrit l'action des connecteurs Booléens sur les valeurs de vérité. Ces connecteurs Booléens expriment les propriétés des connecteurs logiques via les notions fondamentales de structure et d'interprétation.

$$\neg_{Bool}(T) = F, \quad \neg_{Bool}(F) = T$$

$$\wedge_{Bool}(T,T) = T, \quad \wedge_{Bool}(T,F) = F, \quad \wedge_{Bool}(F,T) = F, \quad \wedge_{Bool}(F,F) = F$$

$$\vee_{Bool}(T,T) = T, \quad \vee_{Bool}(T,F) = T, \quad \vee_{Bool}(F,T) = T, \quad \vee_{Bool}(F,F) = F$$

$$\Rightarrow_{Bool}(T,T) = T, \quad \Rightarrow_{Bool}(T,F) = F, \quad \Rightarrow_{Bool}(F,T) = T, \quad \Rightarrow_{Bool}(F,F) = T$$

1.2.2 Structures

Définition 1.3 *Une* $(\mathcal{F}, \mathcal{P})$-structure *(ou simplement* structure*) est une paire* (S, I) *où* S *est un ensemble appelé* domaine de discours *et* I *est une* interprétation *qui consiste d'une part en un homomorphisme de* $T(\mathcal{F})$ *dans une* $\mathcal{F}$-algèbre *de domaine* S *munie d'un ensemble*

d'opérations $I(\mathcal{F})$ tel que $f_I \in I(\mathcal{F})$ d'arité n ssi $f \in \mathcal{F}$ d'arité n, et d'autre part en une interprétation des symboles de prédicats telle que $P_I \in I(\mathcal{P})$ est un sous ensemble de D^n ssi $P \in \mathcal{P}_n$.

On abrégera souvent (S, I) en S lorsque l'interprétation est déterminée sans ambigüité par le contexte.

Lorsque le symbole q est d'arité nulle, la relation q_I est alors une valeur de vérité, *T (pour true) ou F (pour false).*

Quelques exemples de structures importantes :

Exemple 1.4

Domaine	*Opérations*	*Relations*	*Nom*	*Notation*
N	$0, S, +$	$=, \leq$	*Arithmétique de Presburger*	*Presburger*
N	$+, *, 0, S$	$=, \leq$	*Arithmétique de Peano*	*Peano*
R	$+, *, 0, S$	$=, \leq$	*Théorie des Réels*	*Tarski*
$\{T, F\}$	$\emptyset$	$\{p_0, \ldots, p_n\}$	*Structure propositionnelle*	*Prop*
$T(F)$	$\hat{F}$	$\{=\}$	*Structure algèbrique*	*F-Algèbre*
$T(F)$	$\hat{F}$	$\hat{R}$	*Modèle de Herbrand*	*Herbrand*

Dans le cas d'une structure algébrique, et plus généralement du modèle de Herbrand, les opérations et relations qui servent à l'interprétation de la syntaxe sont définies comme suit :

$$\forall f \in \mathcal{F},\ \hat{f}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

$$s = t \text{ est égal à } T \text{ ssi } s \equiv t$$

$$\forall R \in \mathcal{P} \setminus \{=\}, \hat{R}(t_1, \ldots, t_n) \in \{T, F\}$$

1.2.3 Interprétation des formules

Nous allons maintenant interpréter les formules, et il y a deux façons de procéder. La première consiste à interpréter une formule comme une fonction Booléenne de ses variables libres. Une formule close sera donc interprétée par une valeur de vérité. Techniquement, l'ensemble des variables libres d'une formule n'étant pas identique à celui de ses sous-formules, il va être nécessaire de calculer la fonction d'interprétation d'une formule vis à vis d'un ensemble quelconque de variables contenant ses variables libres.

La seconde consiste à se donner à priori des valeurs du domaine d'interprétation pour toutes les variables de $\mathcal{X}$. Techniquement, cela nécessite l'introduction d'une fonction (arbitraire) de $\mathcal{X}$ dans le domaine D, appellée *valuation*. Une valuation étant donnée, l'interprétation d'une formule relativement à cette valuation devient une valeur de vérité. Cette solution évite donc la gestion d'un ensemble de variables dans la définition des interprétations, au prix d'un concept supplémentaire, celui de valuation, que l'on peut voir comme une variable globale donnant la valeur de toute les variables afin que les interpretations soient des valeurs plutôt que des fonctions.

Dans ce cours, nous avons choisi la première solution. Il pourra être utile pour le lecteur de reformuler les énoncés et refaire les preuves avec la seconde.

On notera par $[A]_{I,\{x_1,\ldots,x_n\}}(d_1, \ldots, d_n)$ la valeur pour le n-uplet $(d_1, \ldots, d_n)$ de la fonction d'interprétation de la formule A vis à vis de l'interprétation I et de l'ensemble

de variables $\{x_1, \ldots, x_n\}$ contenant $\mathcal{V}ar(A)$. On omettra l'indice I lorsque cela sera sans ambiguité. On utilisera la notation $\overline{x}$ pour la liste de variables $\{x_1, \ldots, x_n\}$ prises dans cet ordre, la lettre d pour désigner un élément du domaine S, et $\overline{d}$ pour la liste de valeurs $\{d_1, \ldots, d_n\}$ prises dans cet ordre.

Variable $[y]_{\{\ldots,y,\ldots\}}(\ldots, d, \ldots) = d$

Terme $[f(t_1, \ldots, t_p)]_{\overline{x}}(\overline{d}) = f_I([t_1]_{\overline{x}}(\overline{d}), \ldots, [t_p]_{\overline{x}}(\overline{d}))$

Atome $[R(t_1, \ldots, t_p)]_{\overline{x}}(\overline{d}) = R_I([t_1]_{\overline{x}}(\overline{d}), \ldots, [t_p]_{\overline{x}}(\overline{d}))$

Formules $[0]_{\overline{x}}(\overline{d}) = F, \quad [1]_{\overline{x}}(\overline{d}) = T$

$[\neg A]_{\overline{x}}(\overline{d}) = \neg_{Bool}[A]_{\overline{x}}(\overline{d})$

$[A \wedge B]_{\overline{x}}(\overline{d}) = [A]_{\overline{x}}(\overline{d}) \wedge_{Bool} [B]_{\overline{x}}(\overline{d})$

$[A \vee B]_{\overline{x}}(\overline{d}) = [A]_{\overline{x}}(\overline{d}) \vee_{Bool} [B]_{\overline{x}}(\overline{d})$

$[A \Rightarrow B]_{\overline{x}}(\overline{d}) = [A]_{\overline{x}}(\overline{d}) \Rightarrow_{Bool} [B]_{\overline{x}}(\overline{d})$

$[\exists x A]_{\overline{x}}(\overline{d}) = T$ ssi il existe $d \in S$ tel que $[A]_{x,\overline{x}}(d, \overline{d}) = T$

$[\forall x A]_{\overline{x}}(\overline{d}) = T$ ssi pour tout $d \in S$, $[A]_{x,\overline{x}}(d, \overline{d}) = T$

Dans la définition ci-dessus, notons que x ne doit pas apparaître dans la liste $\overline{x}$. Si cela était le cas, il faudrait renommer la variable liée x, ce qui est toujours possible. Par ailleurs, l'utilisation du ssi implique que l'interprétation est égale à F lorsque les conditions indiquées ne sont pas satisfaites.

Lemme 1.5 *Soient deux ensembles de variables $\overline{x}$ et $\overline{y}$ tels que $\mathcal{V}ar(A) \subseteq \overline{x}$. Alors la restriction de $[A]_{\overline{x}\cup\overline{y}}$ à $\overline{x}$ coincide avec $[A]_{\overline{x}}$.*

Preuve
Par récurrence sur la structure de A. □

Définition 1.6 *On appelle interprétation de A dans la structure (S, I) la fonction $[A]_{I,\mathcal{V}ar(A)}$ encore notée $[A]_I$ ou tout simplement $[A]$ lorsque cela n'est pas ambigu.*

Le lemme suivant indique que les notions d'interprétation et de substitution sont compatibles :

Lemme 1.7 $[A\sigma]_{\overline{y}}(\overline{d}) = [A]_{\overline{x}}([x_1\sigma]_{\overline{y}}(\overline{d}), \ldots, [x_n\sigma]_{\overline{y}}(\overline{d}))$*, avec les conventions habituelles $\mathcal{V}ar(A) \subseteq \overline{x}$, et $\mathcal{V}ar(x_i\sigma) \subseteq \overline{y}$ pour tout $1 \leq i \leq n$.*

1.2.4 Validé et satisfiabilité

Définition 1.8 *Soit (S, I) une structure interprétant le langage du premier ordre $\mathcal{CP}1(\mathcal{F}, \mathcal{P}, \mathcal{X})$, et A une formule de n variables libres. On dit que :*

- A est satisfiable *(ou, mieux encore,* satisfaisable*) dans (S, I) s'il existe $\overline{d} \in S^n$ tel que $[A](\overline{d}) = T$,*

- A est satisfiable *s'il existe une structure (S, I) dans laquelle A est satisfiable, et* insatisfiable *dans le cas contraire,*

- A est valide *dans (S, I), noté $(S, I) \models A$, si pour tout $\overline{d} \in S^n$, alors $[A](\overline{d}) = T$, et on dit dans ce cas que (S, I) est un* modèle *de A,*

- A est (universellement) valide*, noté $\models A$, si A est valide dans toute structure (S, I), et on dit alors également que A est une* tautologie*.*

Une formule est donc satisfiable si sa clôture existentielle $\exists \mathcal{V}ar(A)A$ s'interprète en T dans une certaine structure (S, I), et valide si sa clôture universelle $\forall \mathcal{V}ar(A)A$ s'interprète en T dans toute structure (S, I). On peut donc toujours raisonner sur des formules closes.

Notons qu'une formule peut être satisfiable sans posséder de modèle. Notons également qu'une formule A est valide ssi sa négation $\neg A$ est insatisfiable. Celle dernière remarque est à la base des méthodes de preuve par réfutation.

1.2.5 Conséquence et équivalence sémantiques

Définition 1.9 *B est* conséquence sémantique *de A, noté $A \models B$, si pour toute structure (S, I), $[B]_{\overline{x}}(\overline{d}) = T$ chaque fois que $[A]_{\overline{x}}(\overline{d}) = T$. La relation de conséquence sémantique s'étend naturellement à des ensembles de formules aussi bien à droite qu'à gauche du symbole $\models$.*

Pour des formules closes, $A \models B$ ssi $(S, I) \models B$ pour tout modèle (S, I) de A.

La relation de conséquence sémantique est un préordre dont l'équivalence associée est l'*équivalence sémantique* notée $\equiv$.

Exemple 1.10 $A \Rightarrow B \equiv B \vee \neg A$

Cette équivalence nous aurait permis de définir un calcul des prédicats sans l'implication, puis de rajouter l'implication $A \Rightarrow B$ comme une abréviation de la formule équivalente $B \vee \neg A$. C'est pour cette raison que nous nous sommes dispensés de quelques autres opérateurs logiques importants, comme l'équivalence, $A \leftrightarrow B$ étant l'abbréviation de $(A \Rightarrow B) \wedge (B \Rightarrow A)$, le ou exclusif, $A \otimes B$ étant l'abbréviation de $(A \vee B) \wedge \neg(A \wedge B)$, etc ...

L'équivalence sémantique permet de munir l'ensemble des formules d'une structure d'algèbre de Boole. Le choix d'un ensemble d'opérateurs adéquat (basé sur le ou exclusif noté $\oplus$) résulte même en une structure d'anneau Booléen. Nous donnons ci-dessous les axiomes de l'algèbre de Boole :

$$
\begin{array}{rcl@{\qquad}rcl}
A \vee (B \vee C) & \equiv & (A \vee B) \vee C & A \wedge (B \wedge C) & \equiv & (A \wedge B) \wedge C \\
A \vee B & \equiv & B \vee A & A \wedge B & \equiv & B \wedge A \\
A \vee 0 & \equiv & A & A \wedge 0 & \equiv & 0 \\
A \vee 1 & \equiv & 1 & A \wedge 1 & \equiv & A \\
A \vee A & \equiv & A & A \wedge A & \equiv & A \\
A \vee (B \wedge C) & \equiv & (A \vee B) \wedge (A \vee C) & A \wedge (B \vee C) & \equiv & (A \wedge B) \vee (A \wedge C) \\
\neg(A \vee B) & \equiv & (\neg A) \wedge (\neg B) & \neg(A \wedge B) & \equiv & (\neg A) \vee (\neg B) \\
\neg 1 & \equiv & 0 & \neg 0 & \equiv & 1 \\
A \vee \neg A & \equiv & 1 & A \wedge \neg A & \equiv & 0
\end{array}
$$

$$\neg\neg A \equiv A$$

Ces égalités ne font pas intervenir explicitement les quantificateurs —ni les symboles de fonction ou de prédicats—, ce sont des égalités *propositionnelles*. Si on les supprime, on se retrouve dans le calcul propositionnel. Savoir si une formule propositionnelle est satisfiable est le problème NP-complet qui a reçu la plus grande attention, tant pour son importance théorique que pour ses applications pratiques, en particulier à la preuve de circuits logiques.

Ces axiomes égalitaires peuvent aussi s'appliquer à des expressions quelconques, pouvant éventuellement contenir des quantificateurs. La prise en compte explicite des interactions entre les quantificateurs et la négation fait apparaître une nouvelle série d'axiomes :

$$\begin{array}{rcl@{\qquad}rcl}
\neg\forall x A & \equiv & \exists x \neg A & \neg\exists x A & \equiv & \forall x \neg A \\
(\forall x A) \wedge B & \equiv & \forall x (A \wedge B) & (\forall x A) \vee B & \equiv & \forall x (A \vee B) \\
 & & \text{if } x \notin \mathcal{V}ar(B) & & & \\
(A \Rightarrow \forall x B) & \equiv & \forall x (A \Rightarrow B) & (A \Rightarrow \exists x B) & \equiv & \exists x (A \Rightarrow B) \\
 & & \text{if } x \notin \mathcal{V}ar(A) & & & \\
(\forall x A \Rightarrow B) & \equiv & \exists x (A \Rightarrow B) & (\exists x A \Rightarrow B) & \equiv & \forall x (A \Rightarrow B) \\
 & & \text{if } x \notin \mathcal{V}ar(B) & & &
\end{array}$$

Ces axiomes joueront un rôle essentiel plus tard lors de la mise en forme normale des formules logiques. La question se pose bien sûr de leur application dans le cas où la condition d'application portant sur la variable quantifiée n'est pas satisfaite. Il suffit alors de la renommer :

Lemme 1.11 *$\forall x\ A \equiv \forall y\ A\{x \mapsto y\}$ pourvu que $y \notin \mathcal{V}ar(A)$.*

La preuve de ce lemme simple est laissée au lecteur.

1.2.6 Théorie d'une structure

Définition 1.12 *La* théorie *d'une structure (S, I) est l'ensemble des formules closes de $CP1$ dont (S, I) est un modèle. On la note $Th(S, I)$.*

Définition 1.13 *Un ensemble Γ de formules closes est une* théorie *s'il est clos par conséquence sémantique, cad $A \in \Gamma$ ssi $\Gamma \models A$.*

La théorie d'une structure est bien-entendu une théorie.

Définition 1.14 *Une théorie Γ est* consistante *si elle ne contient pas à la fois une formule A et sa négation $\neg A$.*

Une théorie Γ est complète *si pour toute formule A, $A \in \Gamma$ ou $(\neg A) \in \Gamma$.*

Une théorie Γ est finiment axiomatisable *s'il existe un sous-ensemble fini de Γ dont Γ soit conséquence sémantique.*

Une théorie Γ est récursivement axiomatisable *s'il existe un sous-ensemble récursif (reconnu par un algorithme qui termine toujours) de Γ dont Γ soit conséquence sémantique.*

Une théorie Γ est décidable *s'il existe une procédure qui termine toujours, telle que pour toute formule A la procédure retourne oui si $A \in \Gamma$ et non dans le cas contraire.*

Une théorie Γ est semi-décidable *s'il existe une procédure telle que pour toute formule A la procédure retourne oui si $A \in \Gamma$ et non ou ne termine pas dans le cas contraire.*

Exemple 1.15 *L'arithmétique de Presburger (resp. Peano, Tarski) est la théorie de la structure Presburger (resp. Peano, Tarski) de l'exemple 1.4.*

Notons que l'usage veut que l'on appelle théorie de Tarski (ou théorie des réels) l'arithmétique du même nom.

Lemme 1.16 *L'arithmétique de Presburger est décidable. L'arithmétique de Peano est indécidable. La théorie de Tarski est décidable.*

Les preuves de ces assertions ne sont pas immédiates, la dernière étant difficile. La décidabilité de l'arithmétique de Presburger peut s'obtenir par des techniques d'automates.

Exemple 1.17 *La théorie du graphe* $(\{1,2\},\{(1,2),(2,1)\})$ *contient les formules* $\forall xy(xEy \Rightarrow yEx)$, $\forall x.\neg xEx$, $\forall xy(xEy \vee \neg xEy)$, *..., où* $E \in \mathcal{P}_2$.

Exemple 1.18 *Le prédicat d'égalité est en général implicitement présent dans le vocabulaire d'un langage du premier ordre. Sa théorie (dite de l'égalité) est définie par les axiomes :*

$$\textit{réflexivité :}\ \forall x\ x = x$$

$$\textit{symétrie :}\ \forall xy\ x = y \Rightarrow y = x$$

$$\textit{transitivité :}\ \forall xyz\ x = y \wedge y = z \Rightarrow x = z$$

Pour chaque entier n *et chaque symbole de fonction* $f \in \mathcal{F}_n$ *:*

$$\textit{congruence :}\ \forall \overline{xy}\ \overline{x} = \overline{y} \Rightarrow f(\overline{x}) = f(\overline{y})$$

Pour chaque entier n *et chaque symbole de prédicat* $R \in \mathcal{P}_n$ *:*

$$\forall \overline{xy}\ \overline{x} = \overline{y} \Rightarrow R(\overline{x}) \leftrightarrow R(\overline{y})$$

Cette théorie possède donc un ensemble fini d'axiomes si le langage $CP1$ *possède un vocabulaire fini.*

Exemple 1.19 *La théorie des groupes est engendrée par exemple par les trois axiomes suivants :*

$$\forall xyz\ x*(y*z) = (x*y)*z,\ \ \forall x\ e*x = x,\ \ \forall x \exists y\ x*y = e$$

La théorie des groupes n'est pas complète, car il existe des groupes commutatifs et d'autres qui ne le sont pas.

Exemple 1.20 *L'arithmétique de Peano est engendrée par les axiomes suivants :*

$$\forall x\ \neg(S(x) = 0),\ \ \forall xy\ S(x) = S(y) \Rightarrow x = y$$

$$\forall x\ x + 0 = x,\ \ \forall xy\ x + S(y) = S(x + y)$$

$$\forall x\ x * 0 = 0,\ \ \forall xy\ x * S(y) = x * y + x$$

$$\forall x\ \neg(x < 0),\ \ \forall x\ 0 < S(x),\ \ \forall xy\ S(x) < S(y) \Rightarrow x < y$$

Pour toute formule A*, et pour toute variable* x *libre dans* A*,*

$$[A(0) \wedge \forall x(A(x) \Rightarrow A(S(x)))] \Rightarrow \forall x A(x)$$

Cette dernière expression est le schéma d'axiomes de récurrence. Il y a donc un axiome de récurrence par formule et variable libre de la formule. L'arithmétique de Peano n'est donc pas finiment axiomatisable, mais elle est récursive car il existe une procédure qui termine toujours qui, étant donnée une formule, réponde oui s'il s'agit d'un axiome de récurrence et non dans le cas contraire.

Lemme 1.21 *L'arithmétique de Peano est incompléte.*

La preuve d'incomplétude est due à Gödel. Sa lecture est recommandée.

Nous avons donc maintenant deux définitions de l'arithmétique de Peano. Montrer qu'elles coincident est laissé à la sagacité (ou à l'érudition) du lecteur.

Exemple 1.22 *La théorie des arbres finis étiquetés sur un alphabet fini est fondamentale à l'informatique. Elle joue un rôle clé en programmation logique en particulier, où elle sous-tend la notion d'unification et les axiomes de Clarke pour le traitement de la négation. Elle sera étudiée plus en détail dans le chapitre* **??**. *Cette théorie est engendrée par les axiomes de l'égalité que nous avons déjà vus ainsi que par les axiomes suivants qui expriment la "liberté" de l'algèbre $T(F)$:*

Décomposition : *Pour chaque entier n et chaque symbole $f \in \mathcal{F}_n$:*

$$\forall \overline{xy}[f(\overline{x}) = f(\overline{y}) \leftrightarrow \overline{x} = \overline{y}]$$

Conflit : *Pour chaque couple d'entiers m, n et chaque couple de symboles $f \in \mathcal{F}_m, g \in \mathcal{F}_n$:*

$$\forall \overline{xy}[f(\overline{x}) = g(\overline{y}) \leftrightarrow 0]$$

Occurence :

$$\forall xy[x[y]_{p \neq \Lambda} = y \leftrightarrow 0]$$

où $x[y]_{p \neq \Lambda}$ est une notation signifiant que y est un sous-arbre de x à une position p différente de la racine. La définition axiomatique de cette notation est laissée au lecteur.

Monde clos :

$$\forall x(\exists \overline{y}\, x = f(\overline{y})) \vee \ldots \vee (\exists \overline{y}\, x = g(\overline{z}))$$

la disjonction précédente énumérant tous les symboles de fonction de $\mathcal{F}$. Cet axiome est donc finitaire si $\mathcal{F}$ est fini.

Notons que cet ensemble de formules est infini, à cause des axiomes d'occurrence, mais récursif.

Theorem 1.23 *La théorie des arbres finis étiquetés est complète.*

Ce résultat est dû à Mal'cev. Une axiomatisation récursive légèrement différente a été donnée par Maher, qui se généralise aisément au cas des arbres infinis (rationels ou pas). Une preuve récente, qui se généralise volontiers à des structures d'arbres quotients par des axiomes comme la commutativité d'un symbole, est due à Comon. Ces questions font l'objet du dernier chapitre.

Définition 1.24 *Deux structures (S, I) et $(S', I,)$ sont* isomorphes *s'il existe une bijection ξ de S dans S' (étendue aux produits cartésiens S^n) telle que :*

- pour tout symbole de fonction $f \in \mathcal{F}_n$ et tout $\overline{d} \in S^n$, $f_I(\overline{d}) = c$ ssi $f_{I'}(\xi(\overline{d})) = \xi(c)$,

- pour tout symbole de relation $R \in \mathcal{P}_n$ et tout $\overline{d} \in S^n$, $\overline{d} \in R_I$ ssi $\xi(\overline{d}) \in R_{I'}$,

Par exemple, deux graphes qui ne diffèrent que par le nom des sommets sont isomorphes.

De difficulté très inégale, les preuves des lemmes qui suivent sont laissées au plaisir du lecteur.

Lemme 1.25 *Deux structures isomorphes sont modèles des même formules.*

Lemme 1.26 *Deux structures finies qui valident le même ensemble de formules sont isomorphes.*

La réciproque de ce théorème est fausse. Les deux structures $(Q, \{<_Q\})$ et $(R, \{<_R\})$ ne peuvent être isomorphes puisque leurs domaines n'ont pas la même cardinalité. Toutefois,

Lemme 1.27 *Les deux structures $(Q, \{<_Q\})$ et $(R, \{<_R\})$ sont modèles des même formules du premier ordre.*

En fait, les deux structures sont bien sûr distinguables, mais pas par des formules du premier ordre construites sur le seul symbole relationnel $<$. Il faut soit rajouter des symboles de fonctions, par exemple 0 ou S, soit considérer des formules de second ordre.

Lemme 1.28 *Si un ensemble de formules a un modèle unique à isomorphisme près, alors sa clôture par conséquence sémantique est une théorie complète.*

Le problème dual, la définissabilité, est par essence informatique : il consiste à programmer dans une structure au moyen du langage du calcul des prédicats du premier ordre. Ce problème a été très étudié dans le cadre des langages de bases de données, pour lesquelles les structures peuvent être considérées comme finies.

Définition 1.29 *Une relation R d'arité n sur une $(F, \mathcal{P})$-structure (S, I) est* définissable *dans la logique du premier ordre s'il existe une formule $A \in \mathcal{CP}1(\mathcal{F}, \mathcal{P}, \mathcal{X})$ dépendant des n variables libres $\overline{x}$ telle que $[A]_{I,\overline{x}}(\overline{d}) = T$ ssi $\overline{d} \in R$. La définition s'étend aux classes de structures.*

Exemple 1.30 *La relation de la structure Peano $R(x)$ ssi x est premier est définissable par la formule :*

$$\forall yz\ x = y * z \Rightarrow (y = x \wedge z = S(0)) \vee (z = x \wedge y = S(0))$$

L'interprétation de cette formule dans la structure Peano vaut en effet T pour tout nombre premier et F pour tout nombre non premier.

De très nombreuses propriétés élémentaires des structures usuelles comme les graphes ne sont pas exprimables on logique du premier ordre (voir les exercices). La logique du premier ordre est donc assez pauvre. Pour remédier à ce problème, la solution consiste à enrichir le langage logique, en autorisant la quantification sur des variables de fonctions ou de prédicat. On obtient ainsi la logique du second ordre, dont une restriction, la logique monadique du second ordre, n'autorise que la quantification sur les symboles de prédicats unaires, cad sur des variables interprétées comme des sous-ensembles du domaine d'interprétation. Ce fragment de la logique a une grande importance en informatique théorique, puisqu'il permet de caractériser les langages (de mots ou d'arbres) reconnaissables par un automate fini (de mot ou d'arbre). Il a une grande importance pratique également, puisque les automates sont utilisés de manière routinière pour modéliser les circuits logiques.

1.3 Exercices

1.3.1

Soit la structure propositionnelle $(S, \{\}, \{p_1, \ldots, p_n\})$.

1. Soit $n = 3$. On considère la formule $A = (p_1 \wedge p_2 \wedge \neg p_3) \vee (p_1 \wedge \neg p_2 \wedge p_3) \vee (\neg p_1 \wedge p_2 \wedge p_3)$.
   A et $\neg A$ sont-elles satisfiables ? valides ?
2. Soit B la formule où p_1 est remplaçé par $\neg p_1$.
   Est-ce que $A \models B$ ou $B \models A$?
3. Soit maintenant $n = 4$. Trouver une formule Ψ qui s'interprète en T dans la structure propositionnelle ssi le nombre de symboles propositionnels égaux à T dans la structure est égal au nombre de symboles propositionnels égaux à F.
4. Peut-on généraliser à $n = 2k$?

1.3.2

Exercice 1.31 *Le but de cet exercice est l'axiomatisation de la théorie de Presburger. On considère donc un langage qui contient une constante r, un symbol de fonction unaire S et comme seul symbol de prédicat $=$.*

On désire étudier une classe de graphes considérés comme des structures $\{E, \{r_E, S_E\}, \{=\}\}$. r est interprété comme la racine (supposée unique) du graphe, et S comme une fonction injective donnant le successeur de chaque nœud (élément de E) dans la relation du graphe, qui est donc fonctionnelle. Ces propriétés sont axiomatisées par les trois axiomes :

$$A : \ \forall x \ \neg(S(x) = r)$$

$$B : \ \forall x \ (\forall y \neg(S(y) = x)) \Rightarrow x = r$$

$$C : \ \forall xy \ (S(x) = S(y) \Rightarrow x = y)$$

On considère les structures :

$$N = \{N, \{0, succ\}, \{=\}\}, \quad Z = \{Z, \{succ\}, \{=\}\}, \quad Z/p = \{[0..(p-1)], \{succ\}, \{=\}\}$$

et appellerons $NZ * Z/p*$ l'ensemble des structures formées d'une copie unique de N, d'un nombre fini ou infini (de cardinal arbitraire) de copies de Z, et d'un nombre fini ou infini (de cardinal arbitraire) de copies de Z/p.

1. *Montrer que les axiomes A, B, C n'ont pas de modèle fini.*
2. *Montrer que toute structure de $NZ * Z/p*$ est un modèle de A, B, C.*

On se propose maintenant de montrer que tout modèle de A, B, C est isomorphe à une structure de $NZ * Z/p*$. Pour cela, on considère les trois sortes de composantes connexes d'un modèle, infini avec racine, infini sans racine, et fini.

1. *Considérons une composante connexe infinie avec racine r. Soit ξ l'application de N dans les sommets de la composante connexe définie par $\xi(0) = r$ et $\xi(succ(x)) = S(\xi(x))$ pour tout entier $x \geq 0$. Montrer successivement que ξ est une application de N dans les sommets de la composante connexe de r, puis qu'elle est injective, enfin qu'elle est surjective. En déduire que la composante connexe est isomorphe à N.*

2. *Considérons une composante connexe infinie sans racine. Soit ξ l'application de Z dans les sommets de la composante connexe définie par $\xi(0) = a$, où a désigne un sommet arbitraire de la composante connexe, $\xi(succ(x)) = S(\xi(x))$ pour tout entier $x \geq 0$, et $\xi(pred(x)) = y$ tel que $S(y) = x$ pour tout entier $x \leq 0$, et En déduire que la composante connexe est isomorphe à Z.*

3. *Considérons une composante connexe finie ayant p sommets. Montrer que cette composante connexe est en fait un cycle Hamiltonien élémentaire (passant une fois et une seule par chaque sommet). En déduire qu'elle est isomorphe à Z/p.*

4. *En déduire que tout modèle de A, B, C est isomorphe à une structure de $NZ * Z/p*$.*

On se propose maintenant d'éliminer certains modèles superflus pour se rapprocher du modèle "standard" des entiers qui est formé d'une unique composante connexe. Pour cela, on va rajouter le schéma d'axiomes d'induction :

$$D : \Phi(0)[\forall x(\Phi(x) \Rightarrow \Phi(S(x)))] \Rightarrow \forall x \Phi(x)$$

pour toute formule Φ et variable libre x de Φ. Soit Ψ la formule $\forall x \neg S^p(x) = x$, où S^p désigne l'application p fois de S.

1. *Montrer que Ψ n'est pas valide dans un modèle qui contient une copie de Z/p.*

2. *Montrer que $\neg S^p(0) = 0$ est valide dans les modèles de A, B, C, D.*

3. *Montrer que $\forall x\ (\neg S^p(x) = x \Rightarrow \neg S^p(S(x)) = S(x))$ est valide dans les modèles de A, B, C.*

4. *En déduire que Ψ est valide dans les modèles de A, B, C, D.*

5. *En déduire que les modèles de A, B, C, D sont formés d'une unique copie de N, et d'un nombre arbitraire de copies de Z.*

6. *Peut-on éliminer les copies de Z en rajoutant de nouveaux axiomes ou schémas d'axiomes ?*

On dit qu'une théorie T est catégorielle *pour un cardinal α ssi tous les modèles de T de cardinal α sont isomorphes.*

- *Pour quels cardinals α la clôture de A, B, C est-elle catégorielle ?*
- *Même question pour la clôture de A, B, C, D.*

Le théorème de Łos-Vaught énonce que si une théorie T sur un langage fini n'a que des modèles infinis, et si T est catégorielle pour un cardinal infini, alors T est complète.

- *La clôture par conséquence sémantique de A, B, C est-elle complète ?*
- *La clôture par conséquence sémantique de A, B, C, D est-elle complète ?*

1.3.3

Trouver une formule A batie sur le langage défini par la constante 0, le symbole de fonction unaire S, le prédicat d'égalité $=$ et le prédicat ternaire $Plus$ telle que la structure N soit un modèle de A ssi le symbole de prédicat $Plus$ est interprété par la relation $P+$ définie par $(x, y, z) \in P+$ ssi $z = x + y$.

Afin de montrer que $P+$ est la seule relation R qui convient, on pourra considérer un plus petit triplet (x, y, z) pour lequel R et $P+$ diffèrent.

1.3.4

On considère la structure $(R, \{0, 1, +, -, *, \div\}, \{<\})$.

1. Evaluer la formule $A = \forall xy \exists z\ (x < y) \Rightarrow (x < z \wedge z < y)$.
2. La structure est-elle un modèle de A ?
3. Trouver les relations Qa telles que $(R, \{0, 1, +, -, *, \div\}, \{<, Qa\}) \models \forall xy(Q(x, y) \Rightarrow Q(y, x)) \Rightarrow (x \neq y)$, où Q désigne un symbole de prédicat binaire.
4. Construire une formule B exprimant la transitivité d'une relation binaire, et une formule C exprimant son antisymmétrie. Est-ce que $B \wedge C \models A$?
5. Construire une formule exprimant que l'ordre partiel défini par B et C est en fait un ordre total.

1.3.5

Peut-on caractériser la connexité d'un graphe quelconque par une formule de $CP1$?

1.3.6

Soient $\mathcal{F} = \{a\}, \mathcal{P} = \{P\}$ et ϕ la formule $\neg P(a) \vee \exists x P(x)$. ϕ est elle satisfaisable ? Possède-t-elle un modèle de Herbrand ? Que faire ?

1.3.7

Décidabilité de l'arithmétique de Presburger par des techniques d'automates.

Le principe sera d'associer à toute formule un automate qui reconnait le langage des entiers (en représentations binaire) qui satisfont la formule. Comme une formule peut avoir plusieurs variables libres, ces automates vont donc devoir lire plusieurs mots à la fois, que l'on représentera sur autant de lignes qu'il y a de variables libres. Ces lignes de mots peuvent être vus comme des mots sur l'alphabet $\{0, 1\}^n$ où n est le nombre de variables libres. Comme les mots associés à deux entiers distincts n'ont pas la même longueur, on les complète par des zéros non significatifs de façon à les égaliser tous.

La première étape est celle de la représentation d'un nombre comme un mot sur l'alphabet $\{0, 1\}$. Comme on désire lire les mots sur l'automate en partant des bits de faible poids, ils seront représentés à l'envers. Ainsi le nombre 11 aura-t-il $\{1101, 11010, 110100, \ldots\}$ parmi ses représentations possibles.

1. Quelle est la taille minimale de la représentation du nombre de valeur n ?

2. Donner les automates déterministes minimaux reconnaissant les formules 0 et 1.
3. Donner les automates déterministes minimaux reconnaissant les formules de la forme $S^n(0) = S^m(0)$ et $S^n(0) \leq S^m(0)$.
4. Énumérer les diverses formes d'atomes clos possibles.
5. Énumérer les diverses formes d'atomes possibles ayant une unique variable libre.
6. Énumérer les diverses formes d'atomes possibles ayant deux variables libres.
7. Pour chaque forme d'atome possible, donner l'automate déterministe minimal reconnaissant le langage des mots qui satisfont la formule.
8. Soit $\mathcal{A}$ un automate reconnaissant le langage des mots qui satisfont la formule ϕ. Donner un automate reconnaissant le langage des mots qui satisfont la formule $\neg\phi$.
9. À partir de maintenant, on suppose que les variables sont ordonnées, on les notera $x_1, \ldots, x_n, \ldots$.
   Soit $\mathcal{A}$ un automate reconnaissant le langage des mots qui satisfont la formule $\phi(\overline{x}, \overline{z})$ une formule que l'on considère maintenant comme dépendant en plus de la variable z. Donner un automate reconnaissant le le langage des mots qui satisfont la formule $\phi(\overline{x}, y, \overline{z})$.
10. Soit $\mathcal{A}_1$ et A_2 des automates reconnaissant le langage des mots qui satisfont les formules ϕ_1 et ϕ_2. Donner un automate reconnaissant le langage des mots qui satisfont la formule $\phi_1 \wedge \phi_2$.
11. Soit $\mathcal{A}$ un automate reconnaissant le langage des mots qui satisfont la formule ϕ. Donner un automate reconnaissant le langage des mots qui satisfont la formule $\exists x \phi$.
12. Soit $\mathcal{A}$ un automate reconnaissant le langage des mots qui satisfont la formule ϕ. Donner un automate reconnaissant le langage des mots qui satisfont la formule $\forall x \phi$.
13. En déduire que la théorie de Presburger est décidable.
14. Quelle est la complexité de l'algorithme obtenu ?

Chapitre 2

Séquents, Clauses et Résolution

Ce chapitre a pour but d'introduire des méthodes calculatoires qui trouvent leur origine dans des travaux de Herbrand et de Robinson. Ces méthodes reposent sur la transformation de formules en clauses. Leur justification dans ce cours fera appel aux travaux de Skolem et Gentzen.

2.1 Feuille de route

Un théorème se présente sous la forme d'un ensemble d'hypothèses $H_1, \ldots, H_n$ et d'une conclusion C, et il s'agit donc de montrer que la formule $H_\wedge \ldots \wedge H_n \Rightarrow C$ est valide, et, en utilisant la complétude du calcul des séquants de Gentzen, d'en construire une preuve dans ce calcul. En fait, nous allons utiliser une démarche plus complexe dans le but d'utiliser les propriétés du calcul des séquents pour certains types de formules appellées clauses.

1. Dans une première étape, nous remarquons que
   $H_\wedge \ldots \wedge H_n \Rightarrow C$ est valide ssi
   $H_\wedge \ldots \wedge H_n \wedge \neg C$ est insatisfiable.
   Par exemple, si l'on désire se convaincre du bien-fondé du raisonnement
   tous les hommes sont grecs ; Socrate est un homme ; donc Socrate est grec.
   on doit tout d'abord le formaliser, ce qui donne :

   $$(\forall x H(x) \Rightarrow G(x)) \wedge H(S) \Rightarrow G(S)$$

   et on est donc amené à vérifier que la formule

   $$(\forall x H(x) \Rightarrow G(x)) \wedge H(S) \wedge (\neg G(S))$$

   est insatisfiable.
2. Dans une seconde étape, nous allons transformer la formule $H_\wedge \ldots \wedge H_n \wedge \neg C$ en un ensemble de clauses $\{C_1, \ldots C_p\}$, c'est-à-dire de formules de la forme

   $$\forall \overline{x} A_1 \vee \ldots \vee A_M \vee \neg B_1 \vee \ldots \vee \neg B_n$$

où les A_i et les B_i sont des formules atomiques, tout en préservant l'insastifiabilité.
On verra que cela est équivalent à transformer chaque élément de la conjonction en clauses. Dans l'exemple précédent, cela donne l'ensemble de clauses :

$$\{H(x) \Rightarrow G(x), H(S), \neg G(S)\}$$

3. Dans une troisième étape, nous reviendrons à la prouvabilité dans le calcul des séquents, grâce toujours au théorème de complétude : l'insatisfiabilité de l'ensemble de clauses $\{C_1, \ldots C_p\}$ équivant à la validité de la formule $C_1 \wedge \ldots \wedge C_p \Rightarrow \bot$ et donc à la prouvabilité du séquent $C_1, \ldots C_p \longrightarrow$.
4. La quatrième étape consistera en l'application du théorème de Herbrand, qui permettra de ramener la prouvabilité du séquent $C_1, \ldots C_p \longrightarrow$ à celle d'un nouveau séquent $C'_1, \ldots C'_{p'} \longrightarrow$ dont les clauses C'_j sont des instances sans variables ni quantificateurs des clauses C_i.
5. La cinquième étape consistera à analyser la preuve devenue propositionnelle du séquent $C'_1, \ldots C'_{p'} \longrightarrow$ en groupant certaines étapes structurelles avec la règle d'introduction $\vee G$, donnant naissance au calcul des séquents clausal.
6. La sixième étape aura pour but d'en déduire la complétude de la règle dite de résolution close :
$$\frac{A \vee C \quad \neg A \vee D}{C \vee D}$$
Dans cette *règle d'inférence*, A dénote un atome, et C et D dénotent des clauses (éventuellement vides). Les clauses $A \vee C$, $\neg A \vee D$, $C \vee D$ sont supposées en forme normale vis à vis des règles de l'algèbre de Boole. En conséquence, ni A ni $\neg A$ ne peuvent apparaître dans $C \vee D$. Elles sont également supposées ne pas avoir de variables communes, ce qui peut nécessiter des renommages appropriés.
La complétude de la règle de résolution s'énoncera comme l'appartennace de la clause vide à l'ensemble $Res^*(\{C'_1, \ldots, C'_{p'}\})$ des clauses inférables par résolution à partir de l'ensemble de clauses $\{C'_1, \ldots, C'_{p'}\}$.
La complétude de la résolution close sera une conséquence directe de la complétude du calcul des séquents clausal.
7. La septième étape sera celle du lemme de relèvement qui permettra en fait de travailler avec des clauses avec variables, en généralisant la règle de résolution close en une règle de résolution générale grâce à l'opération d'unification :
$$\text{Résolution : } \frac{P_1 \vee \ldots \vee P_p \vee C \quad \neg N_1 \vee \ldots \vee \neg N_n \vee D}{C\sigma \vee D\sigma}$$
Dans cette *règle d'inférence* de résolution, σ désigne l'unificateur principal du problème d'unification

$$P_1 = P_2 \wedge \ldots \wedge P_1 = P_p \wedge P_1 = N_1 \wedge N_1 = N_2 \wedge \ldots \wedge N_1 = N_n$$

On peut alors terminer notre exemple, avec les preuves possibles suivantes :

$$\dfrac{\dfrac{\dfrac{G(x) \vee \neg H(x) \quad H(S)}{G(S)} \quad \neg G(S)}{\square} \qquad \dfrac{\dfrac{\dfrac{G(x) \vee \neg H(x) \quad \neg G(S)}{\neg H(S)} \quad H(S)}{\square}}$$

qui engendre la clause vide à partir des clauses de départ.

L'opération d'unification permet en fait d'assurer que si deux clauses ont des instances closes qui peuvent se résoudre par résolution close, alors la résolvante (close) est une instance close de celle obtenue par résolution (générale) des deux closes de départ. Il ne sera donc nul besoin de *connaitre* les instances closes fournies par le théorème de Harbrand, seule leur existence est importante.

8. La dernière étape sera celle de l'unification, dont nous détaillerons les propriétés algorithmiques.

Nous allons maintenant considérer en détail ces différentes étapes.

2.2 Mise sous forme clausale

On commence par la mise des formules sous forme prénexe.

Définition 2.1 *Une formule est en forme* prénexe *si elle est non quantifiée, ou de l'une des deux formes $\forall x\phi$ ou $\exists x\phi$ où ϕ est une formule prénexe.*

Lemme 2.2 *Toute formule logique est équivalente à une formule prénexe.*

Il suffit pour cela d'utiliser les axiomes vus précédemment qui permettent de repousser les quantificateurs à l'extérieur des formules lorsqu'ils sont appliqués (arbitrairement) de la gauche vers la droite (un nombre nécessairement fini de fois). Cela peut bien sûr nécessiter de renommer les variables liées d'une formule.

Définition 2.3 *Une* formule clausale *est une formule prénexe close sans quantificateur existentiel.*

Une clause *est une formule clausale de la forme $\forall\overline{x}(A_1 \wedge \ldots \wedge A_n \Rightarrow B_1 \vee \ldots \vee B_p)$, ou, de manière équivalente $\forall\overline{x}(\neg A_1) \vee \ldots \vee (\neg A_n) \vee B_1 \vee \ldots \vee B_p$. Les quantificateurs d'une clause étant tous universels, ils sont souvent omis. Il sera parfois utile de distinguer la version quantifiée d'une clause de sa version non quantifiée en écrivant $\forall\overline{x}C$ pour la première et C pour la seconde. Une clause sera dite close si elle nést pas quantifiée et si elle ne contient pas de variable.*

On dira qu'une clause close C' est une instance d'une clause $\forall\overline{x}C$ si elle est obtenue par mise sous forme clausale df'une formule de la forme $C\gamma$ où γ est une substitution close de domaine $\mathcal{V}ar(C)$.

Lemme 2.4 *Toute formule clausale est équivalente à une conjonction de clauses.*

Preuve
La preuve et constructive. Il suffit d'appliquer les règles de l'algèbre de Boole (sauf les axiomes de commutativité et associativité), et de remarquer que ces règles terminent lorsqu'elles sont appliquées de gauche à droite sur les classes d'équivalence de formules modulo la commutativité et l'associativité de la conjonction et de la disjonction. La forme normale est une conjonction de clauses. On peut aussi faire une preuve par récurrence. □

L'importance de la notion de clause vient du théorème de Skolem :

Theorem 2.5 *Pour toute formule close A, il existe un ensemble S de clauses tel que S est insatisfiable ssi A l'est.*

Preuve
Par récurrence sur le nombre de quantificateurs existentiels de A supposée en forme prénexe sans perte de généralité, on démontre l'existence d'une formule clausale A' telle que $A' \models A$. Et donc A' est insatisfiable puisque A l'est. Mais il ne serait pas correct de dire que A et A' sont équivalentes, car on va voir que les deux formules ne sont pas construites sur le même langage logique.

Si A ne possède pas de quantificateur existentiel, c'est terminé. Sinon, soit $A = \forall \overline{x} \exists y\ B$, où B est donc une formule prénexe. Soit f un nouveau symbole de fonction (dit *symbole de Skolem*) dont l'arité est égale au nombre de variables dans $\overline{x}$, et C la formule prénexe $\forall \overline{x} B\{y \mapsto f(\overline{x})\}$. On vérifie que $C \models A$ grâce au lemme 1.7. C contient un quantificateur existentiel de moins que A, donc il existe une formule clausale A' telle que $A' \models C$. On conclue par transitivité de la relation de conséquence sémantique, et application du lemme précédent.

Notons que si C était satisfiable dans une interprétation (S, I), alors on en déduirait aisément une interprétation (S, I') (prenant pour I' la restriction de I au langage de A) telle que A soit satisfiable. □

Exemple 2.6 *Mise en forme clausale de la formule $\neg(\forall x \exists y.P(x,y)) \vee \exists z.Q(z)$:*

$$\begin{array}{lcl} \neg(\forall x \exists y.P(x,y)) \vee \exists z.Q(z) & \rightarrow & \exists x \forall y \exists z.\neg P(x,y) \vee Q(z) \\ & \rightarrow & \forall y.\neg P(a,y) \vee Q(f(y)) \end{array}$$

Mais la forme clausale n'est pas unique ; on peut aussi effectuer la transformation :

$$\begin{array}{lcl} \neg(\forall x \exists y.P(x,y)) \vee \exists z.Q(z) & \rightarrow & \exists x.(\forall y.\neg P(x,y) \vee Q(x)) \\ & \rightarrow & \forall y.\neg P(a,y) \vee Q(a) \end{array}$$

On peut bien sûr également s'intéresser à minimiser le nombre de clauses, ou tout autre mesure de la taille du problème final à résoudre.

2.3 Théorème de Herbrand

Le théorème d'élimination des coupures de Gentzen peut être renforcé, de manière à obtenir le théorème dit "du séquent intermédiaire" qui énonce que tout séquent valide Σ : $\Gamma \longrightarrow \Delta$ possède une preuve organisée en deux parties :

- une partie supérieure propositionnelle, c'est-à-dire sans utilisation des règles portant sur les quantificateurs, aboutissant à la preuve d'un unique séquent Σ' : $\Gamma' \longrightarrow \Delta'$ dont toutes les formules sont des sous-formules non quantifiées de Γ, Δ au sens suivant : à la notion de sous-formule structurelle habituelle, on ajoute que $A(t)$ est uns sous-formule de $\forall x_\tau A(x)$ à condition que t soit un terme de type τ construit sur la signature disponible.
- une partie inférieure dont toutes les règles sont des introductions de quantificateurs ou des contractions.

Nous allons décrire ce théorème et sa preuve dans le cas beaucoup plus simple des clauses. Ce cas élimine le besoin d'appliquer des règles droites ainsi que la règle $\exists L$, et donc le vocabulaire de la preuve ne change pas pendant la preuve, ce qui permet de l'omettre.

Théorème 2.7 *Tout séquent valide $\Gamma \longrightarrow$ où Γ est une conjonction de clauses possède une preuve organisée en deux parties :*

- *une partie supérieure propositionnelle, dont les seules règles autorisées sont des $\vee G$ et des règles structurelles gauches, aboutissant à la preuve d'un séquent $\Gamma' \longrightarrow$ formé de clauses qui sont des instances closes des clauses de Γ ;*
- *une partie inférieure dont toutes les règles sont des introductions $\forall G$ ou des contractions gauches.*

Preuve
Par récurrence appropriée et transformation (récursive) de la preuve par des règles de permutation adéquates à partir d'une preuve initiale supposée sans coupure. Notons quíl faut modifier la règle Initial, de manière à manipuler des clauses, Elle devient :

$$\frac{}{A, \neg A \longrightarrow}$$

où A désigne un atome.

□

Donnons un exemple de permutation de façon à voir son impact sur la preuve.
Preuve initiale :

$$\dfrac{\dfrac{\dfrac{\dfrac{A[t/x], \Gamma' \longrightarrow}{\forall x A, \Gamma' \longrightarrow}\forall L}{\text{Règles Structurelles RS}} \qquad \dfrac{\ldots}{C, \forall x A, \Gamma \longrightarrow}}{B, \forall x A, \Gamma \longrightarrow}}{B \vee C, \forall x A, \Gamma \longrightarrow}\vee L$$

Preuve transformée :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A[t/x], \Gamma' \longrightarrow}{A[t/x], \forall x A, \Gamma' \longrightarrow}WL}{\text{Règles Structurelles RS}} \qquad \dfrac{\dfrac{\ldots}{C, \forall x A, \Gamma \longrightarrow}}{C, A[t/x], \forall x A, \Gamma \longrightarrow}WL}{B, A[t/x], \forall x A, \Gamma \longrightarrow}}{B \vee C, A[t/x], \forall x A, \Gamma \longrightarrow}\vee L}{B \vee C, \forall x A, \forall x A, \Gamma \longrightarrow}\forall L}{B \vee C, \forall x A, \Gamma \longrightarrow}CL$$

On peut remarquer que la distance de la règle $\forall L$ à la racine de la preuve a décru. Il y a bien sûr d'autres cas à considérer, et il faut trouver une mesure de la complexité de preuve qui convienne à tous.

Notons que l'on pourrait également intégrer à cette étape la phase de skolémisation. Il suffirait pour cela de partir d'un séquent $\Sigma : \Gamma \longrightarrow$ où Γ devrait alors être formé de formules prénexes. Mais la théorème du séquent intermédiaire s'avère alors beaucoup plus difficile à prouver.

2.4 Calcul des séquents clausal

Les règles en sont :

$$\frac{}{\Gamma \longrightarrow} \quad A \in \Gamma, \neg A \in \Gamma \text{ for some atom } A$$

$$\frac{C_1, \ldots, C_m, \Gamma' \longrightarrow \qquad B, \Gamma' \longrightarrow}{D_1 \vee B, \ldots, D_n \vee B, \Gamma \longrightarrow}$$

où

(i) Γ', Γ'' sont des sous-ensembles de $\{D_1 \vee B, \ldots, D_n \vee B, \Gamma\}$
(ii) $\{C_1, \ldots, C_m\}$ est un sous-ensemble non vide de $\{D_1, \ldots, D_n\}$
(iii) B est un littéral

Théorème 2.8 *Le calcul des séquents clausal est correct et complet.*

Preuve
Il s'agit cette fois de regrouper des séquences d'application de règles d'inférences du calcul des séquents dans la partie propositionnelle d'une preuve de la forme précédente.

On regroupe tout d'abord la règle d'initialisation avec les règles structurelles qui la suivent et on obtient ainsi la première règle du calcul.

On regroupe ensuite plusieurs étapes $\vee L$ successives pourvu qu'elles mettent toutes en jeu des séquent de la forme $B, \Gamma \longrightarrow$, ainsi que les règles structurelles qui les suivent, ce qui donne la seconde règle du calcul.

2.5 Complétude de la résolution close

Preuve
Par récurrence sur la structure de l'arbre de preuve du calcul des séquents clausal.

Cas de base : la preuve se réduit à une feuille de la forme $D_1, \ldots, D_n \longrightarrow$, auquel cas il existe D_i et D_j avec $i \neq j$ auxquels appliquer la résolution close pour en déduire la clause vide.

Cas général : La preuve dans le calcul des séquents clausal se termine donc par l'application de la règle , et l'arbre de preuve a donc la forme :

$$\frac{\dfrac{T_1}{C'_1, \ldots, C'_P, \Gamma' \longrightarrow} \qquad \dfrac{T_2}{B, \Gamma'' \longrightarrow}}{C_1 \vee B, \ldots, C_n \vee B, \Gamma \longrightarrow}$$

Par hypothèse de récurrence, $\Box \in Res^*(C'_1, \ldots, C'_p, \Gamma')$ (réfutation R_1) et $\Box \in Res^*(B, \Gamma'')$ (réfutation R_2).

Cas 1 : L'une des réfutations est à partir de $D_1, \ldots, D_n$ et c'est terminé.

Cas 2 : Aucune des réfutations ne travaille à partir de $D_1, \ldots, D_n$ et donc $B \notin \{D_1, \ldots, D_n\}$. On remplace alors tout d'abord $C'_1, \ldots, C'_p$ par $C'_1 \vee B, \ldots, C'_p \vee B$ dans R_1, ce qui fournit un arbre de résolution R'_1 aboutissant en B. Il suffit alors de brancher cet arbre dans R_2 aux occurrences de B pour avoir une réfutation de la clause vide à partir des clauses de départ.

2.6 Relèvement et Complétude de la résolution

Lemme 2.9 *Soient $A \vee C\gamma$ et $\neg A \vee D\eta$ deux instances closes (en forme normale disjonctive) des clauses $P \vee C$ et $N \vee D$, telles que (i) $\mathcal{V}ar(P \vee C) \cap \mathcal{V}ar(N \vee D) = \emptyset$, et (ii) les atomes de $P\gamma$ soient tous égaux à A et ceux de $N\eta$ à $\neg A$. Alors $P = N$ possède un plus grand unificateur σ tel que $C\sigma \vee D\sigma \in Res(P \vee C, N \vee D)$, et $C\gamma \vee D\eta = (C\sigma \vee D\sigma)\tau$ pour une certaine substitution τ.*

Preuve
Comme on peut toujours supposer que les clauses $P \vee C$ et $N \vee D$ ont des variables toutes distinctes, on en déduit que la substitution close $\gamma \cup \eta$ égale à γ pour les variables libres de $P \vee C$ et à η pour celles de $N \vee D$ unifie le problème $P = N$. Soit σ l'unificateur principal du problème, et τ la substitution telle que $\gamma \cup \eta = \sigma\tau$. On vérifie sans peine la propriété annoncée. □

On en déduit :

Theorem 2.10 *(Robinson) Un ensemble S de clauses est insatisfiable ssi la clause vide appartient à $Res^*(S)$.*

Preuve
Elle utilise le théorème de Herbrand, la complétude dans le cas clos et le lemme précédent.

Par le théorème de Herbrand, S est insatisfiable ssi cela est le cas de l'ensemble fini E des instances closes de S. Par le lemme de relèvement, $Res^*(E)$ est inclus dans les instances closes de $Res^*(S)$. La clause vide ne pouvant être l'instance que d'elle même, on en déduit le résultat.

Notons que l'utilisation du lemme de relèvement impose l'élimination par résolution de tous les atomes qui deviennent égaux dans l'unification. □

2.7 Lectures

[12, 13, 8]

2.8 Exercices

1 Décrire un algorithme qui engendre comme précédemment un ensemble de clauses insatisfiables pour lequel le nombre de symboles de Skolem ajoutés soit minimal.

2 Montrer la correction du raisonnement suivant : Les Crétois sont tous des menteurs. Je suis Crétois. Donc je suis menteur.

3 Soit $Freres(x, y)$ une relation symétrique, transitive et antiréflexive. Montrer la correction du raisonnement suivant qui se passe en Crète, royaume des menteurs :

   1. tout Crétois a un frère menteur ;
   2. tout Crétois a les cheveux blonds ou bruns ;
   3. deux frères ont la même couleur de cheveux ssi ils sont menteurs tous deux ou pas menteurs tous deux ;
   4. Donc, en Crète, si deux frères n'ont pas la même couleur de cheveux, alors ils ont un troisième frère.

4 Montrer que tout ensemble insatisfiable de clauses contient au moins une clause dont tous les littéraux sont négatifs.

5 Montrer que tout ensemble insatisfiable de clauses contient au moins une clause dont tous les littéraux sont positifs.

6 On suppose les vocabulaires $\mathcal{F}$, $\mathcal{X}$ et $\mathcal{A}$ dénombrables. Soit $\mathcal{T}(\mathcal{F})$ l'ensemble des termes clos, appellé univers de Herbrand et noté $\mathcal{G}$. Soit $\mathcal{A}(\mathcal{P}, \mathcal{F})$ l'ensemble des atomes clos, appellé base de Herbrand et noté $\mathcal{B}$. Une interprétation H de Herbrand
   - a pour domaine l'univers de Herbrand $\mathcal{G}$,
   - interprète le symbole $f \in \mathcal{F}_n$ par l'opération f_H d'arité n telle que $f_H(\overline{t}) = f(\overline{t})$,
   - interprète le symbole de prédicat $R \in \mathcal{P}_n$ par un sous-ensemble arbitraire R_H de $\mathcal{G}^n$.

   On demande de montrer les propriétés suivantes :

   1. Un ensemble S de clauses est satisfiable ssi il l'est dans une interprétation de Herbrand.
   2. La base de Herbrand est dénombrable, et donc $\mathcal{B} = \{A_i\}_{i \in N}$.
   3. Étant donnée une énumération de la base de Herbrand, donner la définition d'un arbre binaire, appellé *arbre des interprétations*, dont les branches soient en correspondance biunivoque avec les interprétations de Herbrand.
   4. Donner l'enumération de la base de Herbrand dans le cas de l'arbre des interprétations de la figure 2.1, pour lequel le vocabulaire est formé des symboles 0 (symbole de constante), s (symbole de fonction unaire) et P (symbole de prédicat binaire).

   Un chemin de la racine à un noeud $p \in (N^+)^*$ de l'arbre sera noté H_p et appellé une *interprétation partielle* des atomes A_0 à $A_{|p|-1}$. On dira que l'interprétation I_q *prolonge* l'interprétation I_p si le chemin p est un préfixe du chemin q. On définit un calcul des interprétations partielles dans une logique trivaluée (avec une nouvelle valeur $\perp$ qui se veut représenter la valeur indéfinie) de la manière suivante :
   Pour toute clause C,
   – $H_p(C) = T$ si $H(C) = T$ dans toute interprétation totale qui prolonge H_p.

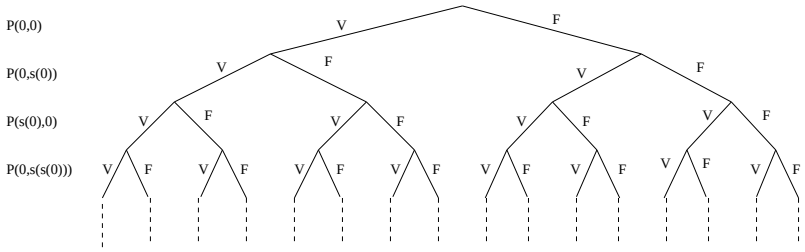

FIG. 2.1 – Exemple d'arbre des interprétations

– $H_p(C) = F$ si $H(C) = F$ dans toute interprétation totale qui prolonge H_p.
– $H_p(C) = \bot$ dans le cas contraire.

On demande de répondre aux questions suivantes :

4. Soit H_p une interprétation partielle qui énumère tous les atomes de la clause close C. Montrer que $[C]_{H_p} \neq \bot$.
5. Soit C une clause telle que $[\forall \overline{x} C]_H = F$. Montrer qu'il existe une interprétation partielle H_p telle que $[\forall \overline{x} C]_{H_p} = F$.

On dit que p est un *noeud d'échec* pour l'ensemble $S = \{C_1, \ldots, C_n\}$ de clauses s'il existe une clause C_i et une substitution close γ telle que $[C_i\gamma]_{H_p} = F$ et la propriété n'est vrai pour aucun prédécesseur de p dans l'arbre des interprétations.

On appelle *arbre sémantique* d'un ensemble S de clauses l'arbre obtenu à partir de l'arbre des interprétations en élaguant les sous-arbres issus d'un noeud d'échec et en étiquettant ce dernier par les instances des clauses réfutées.

Un arbre sémantique sera dit *clos* si toute branche se termine en un noeud d'échec.

7. Supposons que $\mathcal{F}$ est réduit à un symbole de constante 0 et à un symbole unaire s, $\mathcal{P}$ étant réduit au symbole de prédicat unaire A. Soit S l'ensemble de clauses $\{\neg A(x) \vee A(s^2(x)), A(0), A(s(0)), \neg A(s^3(0)) \vee \neg A(s(0))\}$. Dessiner l'arbre sémantique associé pour une énumération naturelle des atomes clos de la base de Herbrand.
8. Cet ensemble de clauses est-il satisfiable ?
9. Quel est l'arbre sémantique clos pour l'ordre sur les atomes qui énumère tous les atomes de la forme $A(s^{2n+1}(0))$ avant d'énumérer les atomes de la forme $A(s^{2n}(0))$?
10. Montrer la propriété suivante [Théorème de Herbrand] : un ensemble S de clauses est insatisfiable ssi il possède un arbre sémantique clos.
11. En déduire qu'un ensemble S de clauses est insatisfiable si et seulement si cela est vrai d'un ensemble fini d'instances closes de clauses de S.
12. En déduire la complétude réfutationnelle de la résolution close.
13. En déduire la complétude réfutationnelle de la résolution close négative, c'est-à-dire faisant nécessairement intervenir une clause dont tous les littéraux sont négatifs.

Chapitre 3

Unification

Les termes avec variables servent à représenter l'ensemble de toutes leurs *instances closes*. Par exemple, si les entiers naturels sont représentés en unaire à l'aide des symboles 0 et s (successeur), le terme $s(x)$ représente l'ensemble infini $\{s(0), s(s(0)), \ldots\}$. Certaines opérations sur les termes sont alors fondamentales. Par exemple, l'intersection des ensembles d'instances de deux termes est obtenue grâce à l'*unification*. L'opération d'unification correspond également à la borne supérieure de deux termes (modulo similarité) pour le préordre d'inclusion des instances.

3.0.1 Problèmes d'unification

Définition 3.1 *Un* problème d'unification P *est ou bien la formule* $\bot$ *ou bien une formule*

$$t_1 = u_1 \wedge \ldots \wedge t_n = u_n$$

où $t_1, \ldots, t_n, u_1, \ldots, u_n$ *sont des termes. Lorsque* $n = 0$*, le problème obtenu est par convention la formule* $\top$*.*

On note par $\mathcal{V}ar(P)$ *l'ensemble des variables de* P*.*

Le signe "=" est symétrique, si bien que nous ne faisons aucune différence entre $s = t$ et $t = s$. Étant donné un problème d'unification, on distingue ses solutions des unificateurs qui les représentent. On fera en général l'hypothèse implicite qu'il existe une constante dans $\mathcal{F}$.

Définition 3.2 *Étant donné un problème d'unification* $t_1 = u_1 \wedge \ldots \wedge t_n = u_n$*,*

- *une* solution *est une substitution close* σ *telle que* $\sigma(t_i) = \sigma(u_i)$ *pour tout* i *;*
- *un* unificateur *est une substitution* σ *telle que toute substitution close soit solution du problème d'unification* $t_1\sigma = u_1\sigma \wedge \ldots \wedge t_i\sigma = u_i\sigma$*.*

Toute substitution close est solution de $\top$ *et aucune substitution close n'est solution de* $\bot$*.*

Deux problèmes d'unification sont équivalents *s'ils ont mêmes ensembles de solutions.*

Exemple 3.3 *Le problème d'unification* $f(x, g(x)) = f(h(y, z), y)$ *n'a pas de solution dans* $T(F, X)$*.*

Un unificateur sert donc à décrire (en utilisant des variables) des ensembles potentiellement infinis de solutions. L'opération d'instantiation nous permet de passer d'un unificateur à un autre, et donc de les comparer :

Définition 3.4 *étant donné un problème d'unification P, on dit que l'unificateur σ est plus général que l'unificateur τ, noté $\sigma \preceq \tau$ s'il existe une substitution λ telle que $\sigma = \tau\lambda$.*

Définition 3.5 *Un unificateur de P est dit* principal *(ou plus général) s'il est minimal pour l'ordre $\preceq$.*

Propriéte 3.6 *Si σ et τ sont deux substitutions principales de Γ, alors elles sont similaires, c'est-à-dire s'échangent par renommage de leur variables.*

Preuve
Laissée au lecteur. □

L'unificateur principal de deux termes, lorsqu'il existe, est donc unique modulo $\stackrel{\bullet}{=}$.

3.0.2 Formes résolues

Les *formes résolues* sont des problèmes d'unification plus simples pour lesquels le calcul des solutions comme des unificateurs (qui forment toujours deux ensembles non-vides) est immédiat ; plusieurs formes résolues sont interessantes : nous verrons les arbre-formes résolues et les DAG-formes résolues et comment les calculer.

Définition 3.7 *Un problème d'unification est une* arbre-forme résolue *(ou forme résolue tout court) si c'est $\bot$ ou $\top$ ou s'il peut s'écrire :*

$$x_1 = t_1 \wedge \ldots \wedge x_n = t_n$$

où $x_1, \ldots, x_n$ sont des variables distinctes qui n'ont pas d'autre occurrence dans le problème ($\bot$ est une abbréviation pour le cas $n = 0$).

Lemme 3.8 *Les solutions d'une arbre-forme résolue $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$ sont les instances closes d'une substitution $\sigma = \{x_1 \rightarrow t_1 \ldots, x_n \rightarrow t_n\}$ qui en est l'*unificateur principal.

Preuve
On vérifie aisément que σ est un unificateur, car $x_i\sigma = t_i$ et $t_i\sigma = t_i$ d'après la condition sur les variables. De plus, si θ est un unificateur arbitraire, $x_i\theta = t_i\theta = (x_i\sigma)\theta$, d'où $\theta = \sigma\theta$, ce qui montre que θ est une instance de σ qui est donc une substitution principale pour l'ensemble des unificateurs. Comme toute solution est un unificateur particulier, et toute instance close d'un unificateur est une solution particulière, on en déduit le résultat. □

Définition 3.9 *Un problème d'unification est une DAG-forme résolue si c'est $\top, \bot$ ou s'il peut s'écrire :*

$$x_1 = t_1 \wedge \ldots \wedge x_n = t_n$$

où $x_1, \ldots, x_n$ sont des variables distinctes telles que, pour tout $i \leq j$, $x_i \notin Var(t_j)$.

Ces formes résolues ne font que représenter un plus général unificateur de façon compacte :

Lemme 3.10 *Si $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$ est une DAG-forme résolue équivalente à $s = t$, alors $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$, où $\sigma_i = \{x_i \to t_i$ pour tout $i \in [1..n]$, est un plus général unificateur de s et t.* □

Preuve
On montre d'abord que σ est un unificateur. On a $x_i\sigma = x_i\sigma_i \ldots \sigma_n = t_i\sigma_{i+1} \ldots \sigma_n = t_i\sigma$ grâce à la condition sur les variables de la forme résolue.

On montre ensuite que σ est principale parmi l'ensemble des unificateurs par récurrence sur n. Le cas de base $n = 0$ est trivial. Dans le cas général, $\tau = \sigma_1\sigma_2 \ldots \sigma_{n-1}$ est unificateur principal de la forme résolue $P' = x_1 = t_1 \wedge \ldots \wedge x_{n-1} = t_{n-1}$. Mais tout unificateur γ de $s = t$ est en particulier unificateur de P', et donc $\gamma = \tau\theta$. Comme γ unifie aussi l'équation $x_n = t_n$, on a $x_n\tau\theta = t_n\tau\theta$, d'où $x_n\theta = t_n\theta$ par condition sur les variables de la forme arbre-résolue, et donc θ est une instance de la substitution $\{x_n \mapsto t_n\}$ qui est unificateur prioncipal de cette équation par le lemme 3.8. Et donc γ est une instance de σ, ce qui termine la preuve. □

Le lecteur aura intérêt à se demander où a été utilisée notre hyothèse d'existence d'une constante.

3.0.3 Règles de transformation

Nous donnons maintenant des règles de transformation qui permettent de réécrire un problème d'unification en un problème d'unification équivalent jusqu'à obtenir une forme résolue.

Les règles de la figue 3.1 sont ici volontairement redondantes (nous verrons pourquoi dans le paragraphe 3.0.5). Ce sont des (schémas de) règles de réécriture dans l'algèbre des problèmes d'unification. En particulier elles peuvent s'appliquer à n'importe quel sous-problème, remplaçant le membre gauche par le membre droit dès que la condition d'application est vérifiée. Le symbole $\wedge$ est ici supposé associatif et commutatif.

Proposition 3.11 *Chacune des règles de la figure 3.1 transforme un problème d'unification en un problème équivalent.*

Preuve
Les règles d'élimination ou de remplacement de variables, d'élimination des équations triviales et de fusion sont des conséquences des axiomes de l'égalité, ou, si l'on préfère, ce sont des conséquence du fait que l'égalité des termes est une congruence.

Les règles de décomposition et d'incompatibilité sont une conséquence des axiomes des termes.

Enfin, le test d'occurrence exprime qu'un terme fini ne peut être égal à un de ses sous-termes stricts. En effet, si c'était le cas, par propriété de congruence de l'égalité, on obtiendrait par récurrence un arbre infini. □

3.0.4 Terminaison

Les règles de la figure 3.1 ne précisent pas de stratégies d'application : à partir de ces règles on peut fabriquer de nombreux algorithmes d'unification en spécifiant un peu plus

équations triviales	$s = s$ $\longrightarrow$ $\top$
Decompose	$f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$ $\longrightarrow$ $s_1 = t_1 \wedge \ldots \wedge s_n = t_n$
Conflit	$f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)$ $\longrightarrow$ $\bot$ Si $f \neq g$
Coalescence	$x = y \wedge P$ $\longrightarrow$ $x = y \wedge P\{x \rightarrow y\}$ Si $x, y \in Var(P),\ x \neq y$
Elimination de variable	$x = s \wedge P$ $\longrightarrow$ $x = s \wedge P\{x \rightarrow s\}$ Si $x \in Var(P), x \notin Var(s)$ et $s \notin \mathcal{X}$
Fusion	$x = s \wedge x = t$ $\longrightarrow$ $x = s \wedge s = t$ Si $x \in X$ et $0 < \|s\| \leq \|t\|$
Test d'occurrence	$x_1 = t_1[x_2]_{p_1} \wedge \ldots \wedge x_n = t_n[x_1]_{p_n}$ $\longrightarrow$ $\bot$ Si $p_1 \cdot \ldots \cdot p_n \neq \Lambda$
Absorbsion	$\bot \wedge P$ $\longrightarrow$ $\bot$
Neutralisation	$\top \wedge P$ $\longrightarrow$ P

FIG. 3.1 – Règles d'unification

dans quel ordre et à quelle sous-problème appliquer les règles. Mais nous allons montrer que, quelle que soit la stratégie d'utilisation des règles, l'algorithme obtenu termine toujours. Ceci permet de factoriser les preuves de terminaison de différents algorithmes d'unification.

Theorem 3.12 *Le système de règles de la figure 3.1 définit un ordre strict bien fondé sur les problèmes d'unification.*

Preuve
On considère les fonctions d'interprétation suivantes des problèmes d'unification :

- Une variable x est dite *résolue* dans un problème P si $P \equiv x = s \wedge Q$ et $x \notin Var(s)$ et $x \notin Var(Q)$.
  $\phi_1(P)$ est le nombre de variables non résolues de P.
- Si $P \equiv s_1 = t_1 \wedge \ldots \wedge s_n = t_n$, alors $\phi_2(P)$ est le multi-ensemble $\{m_1, \ldots, m_n\}$ où $m_i = m(s_i = t_i)$ et $m(s = t) = \max(|s|, |t|)$. Par convention, on pose $m(\bot) = m(\top) = 0$
- $\phi_3(P)$ est le nombre d'équations de P dont l'un des membres au moins est une variable.

$\phi(P)$ est le triplet $(\phi_1(P), \phi_2(P), \phi_3(P))$. L'ensemble de ces triplets est muni de la composée lexicographique de trois ordres :

1. L'ordre habituel sur les entiers naturels
2. L'extension multi-ensemble de l'ordre sur les entiers naturels
3. L'ordre habituel sur les entiers naturels

Il est ainsi muni d'un ordre bien fondé puisque construit à partir d'extensions multi-ensemble et lexicographiques d'ordres bien fondés.

Il suffit maintenant de montrer que si $P \longrightarrow Q$ alors $\phi(P) > \phi(Q)$. Le sens de variations des trois fonctions d'interprétation est résumé dans le tableau ci-dessous :

	ϕ_1	ϕ_2	ϕ_3
équations triviales	$\searrow$	$\searrow$	
Decompose	$\searrow$	$\searrow$	
Incompatibilité	$\searrow$	$\searrow$	
Élimination de variable	$\searrow$		
Fusion	$\searrow$	$=$	$\searrow$
Test d'occurrence	$\searrow$	$\searrow$	
Absorbsion	$\searrow$	$\searrow$	
Neutralisation	$\searrow$	$\searrow$	

Il n'est pas difficile de vérifier qu'aucune règle ne crée de nouvelle variable non-résolue. Les conditions d'application des deux règles d'élimination et de remplacement de variables garantissent par ailleurs qu'une variable non résolue (x dans la formulation des règles doit apparaitre dans P) devient résolue après leur application.

Notons également que la règle de décomposition fait bien décroître ϕ_2 puisqu'elle remplace $m = \max(|f(s_1, \ldots, s_n)|, |f(t_1, \ldots, t_n)|)$ par n entiers strictement inférieurs.

La règle de fusion conserve bien ϕ_2 à cause de la condition $|s| \leq |t|$: $\max(|x|, |t|) = \max(|s|, |t|)$. Enfin, ϕ_3 est bien décroissante par fusion à cause de la condition $s \notin x$ qui,

avec $|s| \leq |t|$, garantit que $s = t$ est une équation dont aucun des membres n'est une variable.

Bien entendu certaines fonctions d'interprétation (ϕ_2 ou ϕ_3) peuvent croître par application de certaines règles, mais la composée lexicographique des trois interprétations est toujours décroissante comme le montre le tableau ci-dessus. □

3.0.5 Complétude

Nous extrayons successivement 3 ensembles de règles de la figure 3.1 et nous montrons que dans chaque cas, l'ensemble de règles permet d'obtenir les formes résolues souhaitées, ce qui fournit trois algorithmes d'unification.

Proposition 3.13 *Si P est un problème d'unification auquel aucune règle de la figure 3.1 ne s'applique, alors P est une arbre-forme résolue. (NB : les règles de fusion et de remplacement de variables sont inutiles pour obtenir ce résultat).*

Preuve
Comme les règles d'incompatibilité et de décomposition ne s'appliquent pas, toutes les équations ont une variable x dans un de leurs membres. Comme le test d'occurrence et la règle des équations triviales ne s'applique pas, si $x = t$ est une des équations de P, x n'est pas une variable de t. Comme la règle d'élimination de variable ne s'applique pas, x ou t doit de plus être une variable résolue, ce qui conduit au résultat souhaité. □

Définition 3.14 *Deux termes s et t sont dits* unifiables *si $s = t$ possède au moins une solution.*

Corollaire 3.15 *Deux termes s et t sont unifiables si et seulement si ils ont un plus général unificateur.*

Property 3.16 *Il existe des termes s et t dont la forme résolue est exponentiellement plus grande que s et t.*

La preuve est à faire en exercice.

Proposition 3.17 *Si P est un problème d'unification auquel la seule règle (peut-être) applicable parmi les règles de la figure 3.1 est l'élimination de variables, alors P est une DAG-forme résolue.*

Preuve
De même que ci-dessus, si la seule règle applicable à P est l'élimination de variable, alors toutes les équations de P sont de la forme $x = t$ où $x \notin Var(t)$. Considérons alors la relation *d'occurrence* $\geq_{occ}$ sur les variables de P définie comme la plus petite relation de préordre telle que, si $x = t[y]_p$ est dans P, alors $x \geq_{occ} y$. Comme le test d'occurrence ne s'applique pas, la relation d'équivalence associée à ce préordre est la plus petite relation d'équivalence $=_S$ qui contient $x =_S y$ si $x = y$ est dans P. Mais, si $x =_S y$, x ou y est une variable résolue puisque la règle de remplacement de variables ne s'applique pas. On

peut donc construire un ordre $\geq$ sur les variables de P tel que : les variables résolues sont minimales et $x >_{occ} y \Rightarrow x > y$. En complétant $\geq$ en un ordre total, on obtient le résultat souhaité. □

Property 3.18 *La forme DAG-résolue de deux termes s et t quelconque est de taille linéaire en la taille de s et t.*

La preuve est à faire en exercice.

Property 3.19 *La forme DAG-résolue de deux termes s et t quelconque peut être calculée en temps $\mathcal{O}(|s| + |t|)$.*

La preuve (non évidente) est laissée au lecteur. Il s'agit de trouver un ordre d'application des règles qui assure la linéarité du calcul.

Exemple 3.20

$$\begin{array}{lll} x = f(f(x)) \wedge x = f(x) & \rightarrow_{\textit{Fusion}} & x = f(x) \wedge f(f(x)) = f(x) \\ & \rightarrow_{\textit{Décomposition}} & x = f(x) \wedge x = f(x) \\ & \rightarrow_{\textit{Fusion}} & x = f(x) \wedge f(x) = f(x) \\ & \rightarrow_{\textit{équations triviales}} & x = f(x) \\ & \rightarrow_{Testd'occurrence} & \bot \end{array}$$

3.1 Lectures

[6, 7, 11, 3]

3.2 Exercices

1. Donner une DAG-forme résolue et une arbre-forme résolue pour les problèmes d'unification suivants :
   (a) $f(f(x_1, x_2), f(x_3, x_4)) = f(f(f(x_2, x_2), f(x_3, x_3)), f(f(x_4, x_4), f(a, a)))$
   (b) $f(x) = y \wedge f(f(y)) = f(z) \wedge x = f(z)$
   (c) $f(x, f(x, x)) = f(f(y, y), f(f(f(z, z), f(a, x')), f(f(a, a), f(a, a))))$
2. On pourrait définir les unificateurs de manière analogue aux solutions, en supprimant la restriction que la substitution est close. Les deux définitions sont-elles équivalentes ?
3. On dit que le terme s est plus général que le terme t s'il existe une substitution σ telle que $t = s\sigma$.
   (a) Montrer que la relation de généralité (ou filtrage) est un préordre.
   (b) Montrer que l'equivalence engendrée par le préordre de généralité est le renommage $\stackrel{\bullet}{=}$.
   (c) La partie stricte du préordre de généralité est bien fondée.

4. Montrer par des exemples que toutes les conditions d'application des règles de la figure 3.1 sont nécessaires pour assurer la terminaison.
5. Soient trois termes unifiables u, v, w avec θ pour unificateur principal, σ l'unificateur principal des deux premiers et τ l'unificateur de $u\sigma$ et w. Montrer que $u\theta = w\tau$.
6. Étant donnés n termes $t_1, \ldots, t_n$ à unifier, quelle est la complexité d'un algorithme d'unification procédant de manière incrémentale, en unifiant (en temps linéaire) successivement, $t_1 = t_2$ (résultat σ_1), puis $\sigma_1 \wedge t_1 = t_3$, etc, jusqu'à $\sigma_1 \wedge \ldots \wedge \sigma_{n-1} \wedge t_{n-1} = t_n$?
7. Si $x_1 = t_1 \wedge \ldots \wedge x_n = t_n$ est une arbre-forme résolue, on appelle *paramètres* les variables autres que $x_1, \ldots, x_n$. Montrer que deux formes résolues équivalentes ont même nombre de variables résolues et le même nombre de paramètres. On parlera donc dans la suite du *nombre de paramètres d'un système d'équations*
8. Soient P et P' deux problèmes d'unification. Montrer que, si toute solution de P est une solution de P' et que, de plus, P et P' ont même nombre de paramètres, alors P et P' ont même ensemble de solutions.
9. Montrer que, si E possède N paramètres et $E_1, \ldots, E_k$ sont des problèmes d'unification ayant chacun strictement moins de N paramètres, alors E possède au moins une solution qui n'est solution d'aucun des problèmes E_i.
10. Montrer que deux arbre-forme résolues équivalentes ont même taille (i.e. même nombre de symboles)
11. Donner un exemple de DAG-forme résolue dont la taille est $O(n)$ alors que l'arbre-forme résolue correspondante est de taille $O(2^n)$.
12. Montrer que si un problème d'unification E est équivalent à une disjonction $E_1 \vee \ldots \vee E_n$ de problèmes d'unification, alors E est équivalent à l'un des E_i.

Chapitre 4

Clauses de Horn et PROLOG

4.1 Stratégies de résolution

Dans ce paragraphe, nous ne faisons pas d'hypothèse restrictive sur la forme des clauses manipulées.

4.1.1 Résolution binaire

La règle de résolution que nous avons donné a le désavantage d'unifier des termes en nombre arbitraire plutôt que des couples de termes. Nous allons donc donner une nouvelle version de la résolution à l'aide de règles d'inférences dites binaires, appellées RFB, pouvant coder la résolution générale.

$$\text{Résolution Binaire : } \frac{A \vee C \quad \neg B \vee D}{C\sigma \vee D\sigma}$$

où σ est l'unificateur principal des atomes A et B.

$$\text{Factorisation Binaire : } \frac{A \vee B \vee C}{B\sigma \vee C\sigma}$$

où σ est l'unificateur principal des littéraux A et B.

Theorem 4.1 *Résolution et factorisation binaires sont correctes et complètes, cad un ensemble de clauses S est insatisfiable ssi la clause vide appartient à $RFB^*(S)$.*

Preuve
On montre que toute résolution se code comme une séquence de résolutions et factorisations binaires. Ce codage utilise de façon essentielle le fait que les clauses résolues ne partagent pas de variables. On part d'une inférence par résolution :

$$\frac{P_1 \vee \ldots \vee P_p \vee C \quad \neg N_1 \vee \ldots \vee \neg N_n \vee D}{C\sigma \vee D\sigma}$$

où σ désigne l'unificateur principal du problème d'unification

$$P_1 = P_2 \wedge \ldots \wedge P_1 = P_p \wedge P_1 = N_1 \wedge N_1 = N_2 \wedge \ldots \wedge N_1 = N_n$$

et on résoud le problème d'unification avec une stratégie adéquate d'application des règles d'unification de manière à faire apparaître les factorisations binaires successives terminées par une résolution binaire. Il est important de noter ici que les clauses à résoudre sont renommées de manière à ce qu'elles ne partagent pas de variable.

Pour cela, on commence par unifier $P_1 = P_2$, ce qui donne un unificateur principal σ_2, et l'on remplace les variables de $\mathcal{D}om(\sigma_2)$ dans le reste du problème, c'est-à-dire dans $P_3, \ldots, P_p$. Cela fait apparaître une factorisation, et l'on recommence jusqu'à résolution complète du problème $P_1 = P_2 \wedge \ldots \wedge P_1 = P_p$, qui fait donc apparaître les unificateurs partiels $\sigma_2, \ldots, \sigma_p$. On recommence ensuite avec le problème $N_1 = N_2 \wedge \ldots \wedge N_1 = N_n$, ce qui introduit les unificateurs partiels $\tau_2, \ldots, \tau_n$. Il nous reste à terminer avec le problème $P_1\sigma_2 \ldots \sigma_n = N_1\tau_2 \ldots \tau_n$, qui engendre une résolution binaire.

□

On peut bien sûr se demander quel est l'impact de cette stratégie sur la complexité des opérations d'unification qui doivent être effectuées, d'un seul coup pour l'unification générale, et sous forme d'unifications binaires succesives dans le cas binaire. Cette question intéressante est laissée à la sagacité du lecteur.

4.1.2 Résolution négative

Une preuve par résolution consiste à explorer l'ensemble $Res^*(S)$ organisé généralement sous forme d'un arbre. Cet ensemble étant en général infini, et la clause vide pouvant se trouver loin de la racine, il est essentiel d'en réduire la taille d'une part, et d'adopter des stratégies d'exploration efficaces d'autre part. Pour cela, une méthode consiste à contraindre l'application de la règles de résolution. Parmi les nombreuses restrictions de la résolution, l'une d'elle joue un rôle essentiel, la résolution négative, pour laquelle au moins l'une des deux clauses de départ ne contient que des littéraux négatifs.

$$\text{Résolution négative : } \frac{P \vee C \quad N \vee D}{C\sigma \vee D\sigma}$$

où σ est l'unificateur principal du problème $P = N$, et la clause $N \vee D$ ne contient que des littéraux négatifs. On notera par $Res - neg^*(S)$ l'ensemble des clauses qui peuvent être inférées à partir d'un ensemble S de clauses par la règle de résolution négative.

Theorem 4.2 *La résolution négative est correcte et complète : un ensemble S de clauses est insatisfiable ssi la clause vide appartient à $Res - neg^*(S)$.*

Preuve
Le lemme de relèvement s'appliquant sans changement, il suffit de prouver la complétude dans le cas clos. On peut le faire par une méthode sémantique, ce qui fait l'objet de l'exercice 2.8.6.13. On peut aussi le faire par une méthode syntaxique, comme nous l'avons fait pour la résolution, en faisant des permutations appropriées dans la preuve obtenue par le théorème du séquent intermédiaire. Le cas général des clauses arbitraires s'avère toutefois assez complexe à mettre en œuvre. □

On peut bien sûr se restreindre à une règle de résolution négative binaire, pourvu que l'on conserve la règle de factorisation binaire.

4.1.3 Stratégie Input

La stratégie dite *input* résout toujours une clause engendrée par résolution (en principe la dernière) avec une clause de l'ensemble S de départ. Cette stratégie a la propriété de ne pas faire exploser l'espace de recherche, elle a donc une importance fondamentale. Elle est malheureusement incomplète, comme on pourra s'en convaincre avec l'exemple $S = \{A \vee B,\ A \vee \neg B,\ \neg A \vee B,\ \neg A \vee \neg B\}$.

4.2 Stratégies de résolution complètes pour les Clauses de Horn

Définition 4.3 *Une clause de Horn est une clause comportant au plus un littéral positif.*

Une remarque essentielle est que le résultat d'une inférence négative mettant en jeu des clauses de Horn est une clause de Horn.

L'étape suivante consiste à préciser ce que sont un programme et une requête en logique de Horn.

Définition 4.4 *Un* programme *en logique de Horn est un ensemble fini de clauses comportant chacune exactement un littéral positif. On notera souvent par $A : -B_1, \ldots, B_p$ une telle clause, n pouvant être nul.*

Une requête *est un ensemble fini (disjonction) de littéraux négatifs. On notera souvent par $: -R_1, \ldots, R_n$ une telle clause.*

Notons que tout programme possède des modèles, puisqu'il suffit d'interpréter tous les atomes de la base de Herbrand par vrai.

Des définitions, on déduit facilement :

Theorem 4.5 *La résolution négative input est complète pour les programmes en logique de Horn.*

Preuve

Il suffit de remarquer que la règle de factorisation est inutile dans le cas des clauses de Horn (la preuve utilise à nouveau une stratégie d'unification particulière qui va permettre de résoudre autant de fois que nécessaire avec la même clause du programme), et que seules les clauses du programme comportent un littéral positif les clauses inférées étant toutes des requêtes. Il est à nouveau essentiel de renommer les clauses avant toute résolution. Les détails de cette preuve sont laissés au lecteur. □.

En pratique, étant donnée une requête $R_1, \ldots, R_p$ (donc chaque R_i est un littéral négatif), la stratégie négative consiste à unifier l'un des R_i avec le littéral positif H d'une clause $H \vee A_1 \vee \ldots \vee A_n$ du programme. Si σ est l'unificateur principal de H et R_i, on infère $A_1\sigma \ldots, A_n\sigma$ qui peuvent être considérés comme de nouvelles requêtes. Comme, par la stratégie, la clause but qui vient d'être engendrée est toujours une des prémisses de la nouvelle inférence, le but R_i peut être effacé. Ceci revient ainsi à réécrire le but (ou plutôt un des littéraux du but) en utilisant une des clauses du programme, vu comme une réécriture de la tête (littéral positif) dans le corps (littéraux négatifs).

On peut facilement montrer que le choix d'un littéral particulier du but n'a pas d'influence sur l'obtention de la clause vide. Il suffit pour cela de faire commuter les résolutions

successives. On parle à ce propos de non-déterminisme “don’t care”, et on va donc adopter un ordre particulier pour le choix du littéral à éliminer à chaque étape.

On appelle *fonction de sélection* un ordre total sur les littéraux négatifs qui spécifie le but à réécrire en priorité : on résoud toujours sur le but minimal. Cette stratégie est complète (i.e. à chaque étape il suffit de considérer un seul but). Par contre, lors de la recherche d’une preuve, toutes les clauses permettant de réécrire ce but devront être considérées : on parle cete fois de non-déterminisme “don’t know”.

La règle de résolution avec une fonction de sélection et pour les programmes logiques est appelée *SLD-résolution*. Des considérations ci-dessus il résulte que :

Theorem 4.6 *La SLD-résolution est complète.*

Enfin, en programmation logique, on s’intéresse aussi au *résultat* d’une requête, c’est-à-dire aux valeurs des variables de la requête claculées au cours de l’obtention de la clause vide par résolution. Pour des raisons d’efficacité, on ne souhaite pas appliquer les substitutions. Pour ces raisons, on considère des *clauses contraintes*. Une clause contrainte est une paire formée d’une clause C et d’un problème d’unification ϕ et notée $C \mid \phi$. Une telle clause contrainte représente en fait l’ensemble des clauses $\{C\sigma \mid \sigma \models_{T(F)} \phi\}$. ($\sigma \models_{T(F)} \phi$ si σ est une substitution close des variables libres de ϕ et $\phi\sigma$ est valide dans la théorie des termes finis). En général, on remplace aussi $\models_{T(F)}$ par $\models_{RT(F)}$ afin d’éviter d’avoir à faire le test d’occurrence dans l’unification. Dans ce contexte, la règle de résolution peut s’écrire :

$$\frac{A(t) \vee C \qquad \neg B(u) \vee D \mid \phi}{C \vee D \mid \phi \wedge A(t) = B(u)}$$

et il faut alors lui adjoindre de nouveles règles qui ont trait au traitement des contraintes (on note par $R \mid \phi$ la requête courante, exprimée comme une clause contrainte) :

$$\begin{array}{lcll} R \mid \phi & \rightarrow & R \mid \phi' & \text{si } \phi \longrightarrow_{UNIF} \phi' \\ R \mid \bot & \rightarrow & \text{échec} & \\ \Box \mid \phi & \rightarrow & \text{succès}(\phi) & \phi \text{ est en forme résolue} \end{array}$$

Les variables de t sont supposées ne pas apparaître libres dans u. De plus les variables n’apparaissant pas dans le but initial et n’apparaissant plus non plus dans le but sont implicitement quantifiées existentiellement (ce qui reflète le fait qu’on ne s’intéresse qu’aux valeurs des variables du but initial).

La *réponse* à une requête $\neg B$ est alors la contrainte ψ telle que l’on ait inféré la clause $\Box \mid \psi$. La contrainte ψ est donnée en forme résolue. Ses solutions sont des substitutions σ telles que $P \models B\sigma$.

Le fait que les contraintes soient ici des équations ne joue de rôle que pour la mise en forme résolue et on peut donc employer d’autres systèmes de contraintes plus expressifs (c’est souvent le cas en pratique). De même les clauses du programme P peuvent être contraintes.

Exemple 4.7 *Soit le programme logique*

$$\begin{array}{lcl} I(x, [], x.[]). & & \\ I(x, x.l, x.l). & & \\ I(x, y.l, y.l') & \leftarrow & I(x, l, l') \mid x \neq y \end{array}$$

Qui a pour but d'insérer un élément dans une liste sans doublons.

Soit la requête $I(0, s(0).(x.[]), y)$. On obtient deux dérivations possibles conduisant à $\square \mid \psi$ avec une contrainte ψ satisfiable : la première est (par souci de simplicité, nous n'écrivons la contrainte complète que dans cette séquence, puis nous écrivons une contrainte simplifiée).

$$\begin{array}{rcl} I(0, s(0).(x.[]), y) & \rightarrow & I(x_1, l, l') \mid 0 = x_1 \wedge y_1.l = s(0).(x.[]) \wedge y = y_1.l' \wedge x_1 \neq y_1 \\ & \rightarrow & \square \mid \exists x_2, l', l'', \wedge y = s(0).l' \wedge x_2 = 0 \wedge x_2.l'' = x.[] \wedge x_2.l'' = l' \\ & \rightsquigarrow & \square \mid x = 0 \wedge y = s(0).(0.[]) \end{array}$$

La deuxième est :

$$\begin{array}{rcll} I(0, s(0).(x.[]), y) & \rightarrow & I(x_1, l, l') \mid & x_1 = 0 \wedge l = x.[] \wedge y = s(0).l' \\ & \rightarrow & I(x_2, l_1, l'_1) \mid & \exists y_2, l, l', \ l = x.[] \wedge y = s(0).l' \wedge x_2 = 0 \\ & & & \wedge l = y_2.l_1 \wedge l' = y_2.l'_1 \wedge x_2 \neq y_2 \\ & \rightarrow & \square \mid & \exists x_2, l_1, l'_1, y_2, l, l', \ l_1 = [] \wedge l'_1 = x_2.[] \\ & & & \wedge l = x.[] \wedge y = s(0).l' \wedge x_2 = 0 \wedge l = y_2.l_1 \\ & & & \wedge l' = y_2.l'_1 \wedge x_2 \neq y_2 \\ & \rightsquigarrow & \square \mid & \wedge y = s(0).(0.(x.[]))) \wedge x \neq 0 \end{array}$$

Nous reformulons ci-dessous les résultats de correction et de complétude de la SLD-résolution en termes de requêtes et de programmes :

Proposition 4.8 *Si la réponse ψ à une requête R pour un programme logique P a pour mgu σ, alors, pour toute substitution close θ, $R\sigma\theta, P$ est insatisfiable.*

Proposition 4.9 *Si R est une requête pour un programme logique P et si $R\theta, P$ est insatisfiable, alors il existe une réponse ψ à R telle que ψ a pour mgu σ et il existe une substitution τ telle que $\sigma\tau = \theta$.*

4.3 Plus petit modèle de Herbrand et sémantique des programmes logiques

Le but de ce paragraphe est de préciser lequel, parmi tous les modèles de Herbrand d'un programme logique, est celui qui sous-tend les calculs par résolution. On va en donner une caractérisation abstraite, puis une caractérisation calculatoire, et lón montrera qu'elles coincident.

Si P est un programme logique. On note $\mathcal{B}$ la base de Herbrand et $2^{\mathcal{B}}$ l'ensemble des parties de $\mathcal{B}$, que l'on confond aussi avec l'ensemble des interprétations de Herbrand (les éléments de $I \in 2^{\mathcal{B}}$ correspondent aux éléments de la base qui s'interprètent en vrai, les autres s'inteprétant à faux). On note M_P l'intersection de tous les modèles de Herbrand de P, dont on va montrer qu'il s'agit bien d'un modèle de P, qui sera donc le "plus petit modèle de Herbrand" du programme :

$$M_P = \bigcap_{I \in 2^{\mathcal{B}}, I \models P} I.$$

Proposition 4.10 *Pour tout programme logique P, $M_P \models P$.*

Preuve
Il suffit de montrer que l'intersection d'un nombre quelconque de modèles de Herbrand de P est aussi un modèle de Herbrand de P. On fait le raisonnement sur des clauses closes pour simplifier.

Soit $A \vee \neg B_1 \vee \ldots \vee \neg B_n$ une clause C. Il y a deux cas.

$A \in M_P$. Alors $M_P \models C$.

$A \notin M_P$. Alors, par définition de M_P, il existe un modèle de Herbrand I de P tel que $A \notin I$. Mais comme $I \models C$, il existe j tel que $B_j \notin I$, et donc $B_j \notin M_P$ par définition de M_P, d'où $M_P \models C$. □

Exemple 4.11 *Soit P le programme :*

$$Q(s(x)) \vee \neg Q(x)$$
$$P(0) \vee \neg Q(s(x))$$

Alors $M_P = \emptyset$ car l'interprétation vide (tout est faux) satisfait bien P.

Ainsi M_P constitue une interprétation "standard" du programme logique P. De façon classique, on peut aussi construire M_P à l'aide de l'opérateur de conséquence immédiate :

Si P est un programme logique, on note T_P l'*opérateur de conséquence immédiate* défini sur $2^{\mathcal{B}}$ par

$$T_P(I) = \{A \in \mathcal{B} \mid A \vee \neg A_1 \vee \ldots \vee \neg A_n \text{ est une instance close d'une clause de } P, \text{ et} A_1, \ldots, A_n \in I\}$$

Exemple 4.12 *Reprenons l'exemple 5.10. Alors $T_P(\emptyset) = \emptyset$, $T_P(\mathcal{B}) = \{P(0)\} \cup \{Q(s(t)) \;\; t \in T(\mathcal{F})\}$, $T_P(T_P(\mathcal{B})) = \{P(0)\} \cup \{Q(s(s(t))) \;\; t \in T(\mathcal{F})\}$, etc.*

$2^{\mathcal{B}}$ est muni d'une structure de treillis complet pour l'ordre $\subseteq$ (i.e. l'ordre d'inclusion des interprétations). On va donc pouvoir caractériser le plus petit point fixe de T_P grâce au théorème de Tarski, comme la limite de ses approximations successives, pourvu que l'opérateur T_P soit croissant et continu.

Proposition 4.13 *T_P est croissant sur les interprétations de Herbrand.*

Preuve
La preuve en est laissée au lecteur. □

Proposition 4.14 *T_P est continu sur les interprétations de Herbrand.*

Preuve
Soit une famille croissante de parties de $\mathcal{B}$, $\{I_k\}_{k \in \mathbb{N}}$, dont le sup, le treillis des parties étant complet pour l'ordre, est noté $S = \bigcup_k I_k$. T_P étant croissante, $\{T_P(I_k)\}_{k \in \mathbb{N}}$, est une autre famille croissante dont le sup est noté $\bigcup_k T_P(I_k)$.

L'équation $T_P(\bigcup_k I_k) = \bigcup_k T_P(I_k)$ exprime la continuité de l'opérateur T_P :

$$\begin{aligned} T_P(\bigcup_{I\in S} I) = & \{A \in \mathcal{B} \mid A \vee \neg A_1 \vee \ldots \vee \neg A_n \text{ est une instance close} \\ & \text{d'une clause de } P \text{ et } \forall i A_i \in \bigcup_{I\in S} I\} \\ = & \{A \in \mathcal{B} \mid \forall i \exists I_j \in S, A \vee \neg A_1 \vee \ldots \vee \neg A_n \text{ est une instance close} \\ & \text{d'une clause de } P \text{ et } A_i \in I_j\} \\ & \qquad\qquad \text{(Notons ici que } I_j \text{ dépend de } i.) \\ = & \{A \in \mathcal{B} \mid \exists I \in S, A \vee \neg A_1 \vee \ldots \vee \neg A_n \text{ est une instance close} \\ & \text{d'une clause de } P \text{ et } \forall i A_i \in I\} \end{aligned}$$

Notons ici que I est la clause I_j pour j maximal, donc ne dépend plus de i.
Cela utilise bien sûr la propriété de croissance de la famille d'interprétations.

Par le théorème de Tarski, nous en déduisons que le plus petit point fixe de T_P existe (grâce à la propriété de croissance) et est le sous-ensemble

$$F \stackrel{d}{=} T_P^\omega(\emptyset) = \bigcup_k T_P^k(\emptyset)$$

(grâce à la propriété de continuité).

De façon générale, lorsque T_P est monotone, T_P^α pour un ordinal α est défini par récurrence transfinie par : $T_P^0 = \emptyset$, si $T_P^{\alpha+1} = T_P(T_P^\alpha)$ et, si α est un ordinal limite, T_P^α est la borne supérieure des T_P^β pour $\beta < \alpha$. C'est la continuité qui implique que $\alpha = \omega$.

Nous allons maintenant montrer que le modèle minimal est un point fixe de T_P, et que tout point fixe de T_P est un modèle de P. Cela assurera que le modèle minimal coincide avec le plus petit point fixe. Ces propriétés découlent du lemme suivant :

Proposition 4.15 *Soit I une interprétation de Herbrand et P un programme logique. I est un modèle de P si et seulement si $T_P(I) \subseteq I$.*

Preuve
$I \models P$ ssi pour toute instance close $A \vee \neg A_1 \vee \ldots \vee \neg A_n$ d'une clause de P, $I \models A_1, \ldots, A_n$ implique $I \models A$. □

Theorem 4.16 *Si P est un programme logique, M_P est le plus petit point fixe de T_P.*

Preuve
Tout point fixe de T_P est un modèle de P par la proposition 5.15, et il nous sufit donc de montrer que M_P est un point fixe de T_P de mani‘ere à conclure par minimalité de M_P.

M_P étant un modèle de P, $T_P(M_P) \subseteq M_P$ par la proposition 5.15. D'autre part, d'après la proposition 4.13 on a aussi $T_P^2(M_P) \subseteq T_P(M_P)$ et donc $T_P(M_P)$ est un modèle de P par la proposition 5.15. Par minimalité de M_P on a donc $M_P \subseteq T_P(M_P)$ et donc $T_P(M_P) = M_P$. □

Le théorème du point fixe nous assure que $T_P^\omega(\emptyset)$ est le plus petit point fixe de l'opérateur continu T_P. Le modèle minimal est donc calculable, du moins ses approximations finies le sont-elles.

Il est possible de généraliser la définition de T_P aux clauses arbitraires :

$$T_P(I) = \{A \in \mathcal{B} \mid A \vee C \text{ est une instance close d'une clause de } P \text{ et } I \models \neg C\}$$

Mais dans ce cas T_P n'est plus nécessairement continu ni même croissant :

Exemple 4.17 *Supposons P réduit à la seule clause $A \vee B$. $T_P(\emptyset) = \{A, B\}$ et $T_P(\{A, B\}) = \emptyset$, $T_P(\{A\}) = \{A\}$ et $T_P(\{B\}) = \{B\}$. Par exemple pour la partie dirigée $S = \{\emptyset, \{A\}\}$, $T_P(\bigcup_{I \in S} I) = T_P(\{A\}) = \{A\} \neq \{A, B\} = T_P(\emptyset) = \bigcup_{I \in S} T_P(I)$.*

4.4 Lectures

[9, 10, 1, 2, 4]

4.5 Exercices

4.5.1

1. Montrer que résolution et factorisation binaires sont correctes.
2. Montrer que résolution et factorisation binaires sont complètes.
3. La stratégie *unitaire* consiste à ne considérer que des résolutions dont une des prémisses est une clause réduite à un littéral.

   Montrer que la stratégie unitaire est en général incomplète. (Autrement dit, donner un ensemble de clauses insatifiable tel que factorisation + résolution unitaire ne permet pas de dériver la clause vide).

   Plus généralement, montrer qu'il existe une réfutation d'un ensemble S de clauses par la résolution unitaire si et seulement si il existe une réfutation de S par la stratégie "input".
4. Montrer que la stratégie unitaire est complète pour les clauses de Horn. Est-ce une stratégie intéressante ?
5. Étant donné un ensemble P de clause de Horn, montrer que l'opérateur T_P est croissant.
6. Donner un ensemble de clauses tel que T_P^ω n'est pas un point fixe alors que $T_P^{2\times\omega}$ est un point fixe.

Chapitre 5

Stratégies de Résolution et Clauses de Horn

5.1 Résolution binaire

La règle de résolution que nous avons donné a le désavantage d'unifier des termes en nombre arbitraire plutôt que des couples de termes. Nous allons donc donner une nouvelle version de la résolution à l'aide de règles d'inférences dites binaires, appellées RFB, pouvant coder la résolution générale.

$$\text{Résolution Binaire : } \frac{A \vee C \quad \neg B \vee D}{C\sigma \vee D\sigma}$$

où σ est l'unificateur principal des atomes A et B.

$$\text{Factorisation Binaire : } \frac{A \vee B \vee C}{B\sigma \vee C\sigma}$$

où σ est l'unificateur principal des littéraux A et B.

Theorem 5.1 *Résolution et factorisation binaires sont correctes et complètes, cad un ensemble de clauses S est insatisfiable ssi la clause vide appartient à $RFB^*(S)$.*

Preuve
On montre que toute résolution se code comme une séquence de résolutions et factorisations binaires. □

On peut bien sûr se demander quel est l'impact de ce codage sur la complexité des opérations d'unification qui doivent être effectuées, d'un seul coup pour l'unnification générale, et sous forme d'unifications binaires succesives dans le codage. Cette question intéressante est laissée à la sagacité du lecteur.

5.2 Résolution négative

Une preuve par résolution consiste à explorer l'ensemble $Res^*(S)$ organisé généralement sous forme d'un arbre. Cet ensemble étant en général infini, et la clause vide pouvant

se trouver loin de la racine, il est essentiel d'en réduire la taille d'une part, et d'adopter des stratégies d'exploration efficaces d'autre part. Pour cela, une méthode consiste à contraindre l'application de la règles de résolution. Parmi les nombreuses restrictions de la résolution, l'une d'elle joue un rôle essentiel, la résolution négative, pour laquelle au moins l'une des deux clauses de départ ne contient que des littéraux négatifs.

$$\text{Résolution négative : } \frac{P \vee C \quad N \vee D}{C\sigma \vee D\sigma}$$

où σ est l'unificateur principal du problème $P = N$, et la clause $N \vee D$ ne contient que des littéraux négatifs. On notera par $Res - neg^*(S)$ l'ensemble des clauses qui peuvent être inférées à partir d'un ensemble S de clauses par la règle de résolution négative.

Theorem 5.2 *La résolution négative est correcte et complète : un ensemble S de clauses est insatisfiable ssi la clause vide appartient à $Res - neg^*(S)$.*

Preuve
Le lemme de relèvement s'appliquant sans changement, il suffit de prouver la complétude dans le cas clos. Pour cela, on reprend la preuve précédente en précisant le noeud J. Pour cela, on considère un chemin particulier dans l'arbre sémantique clos de $Res^*(S)$, appellé chemin gauche, qui consiste à descendre toujours à gauche dans l'arbre, sauf si le successeur gauche est un noeud d'échec, auquel cas on descend à droite. On appelle $J_1, \ldots, J_n$ les noeuds du chemin où la descente s'est poursuivie à droite, A_{i_k} l'atome interprété dans les interprétations partielles successeurs de J_k, et C_k la clause réfutée par l'interprétation fils gauche de J_k. Comme précédemment, le noeud d'échec R fils droit de J_n est étiquetté par une clause de la forme $A_{i_n} \vee C$, et le noeud d'échec L fils gauche de J_n est étiquetté par une clause de la forme $\neg A_{i_n} \vee D$. On va maintenant montrer que C peut être choisie purement négative, et on pourra alors faire une résolution négative à partir de ces deux clauses, entrainant une contradiction.

Pour cela, on remarque que les seuls atomes positifs pouvant apparaître dans C sont ceux qui sont interprétés en F dans l'interprétation L, cad $A_{i_1}, \ldots, A_{i_{n-1}}$. On va donc faire une récurrence sur le nombre de noeuds d'échecs réfutés par des clauses non purement négatives, de manière à remplacer les clauses en question, du haut vers le bas, par des clauses purement négatives obtenues par inférence. On utilise pour cela au plus $n^2/2$ inférences négatives. On peut ensuite éliminer les atomes positifs de $\neg A_{i-n} \vee C$ par au plus n résolutions négatives avec les clauses obtenues. □

On peut bien sûr se restreindre à une règle de résolution négative binaire, pourvu que l'on conserve la règle de factorisation binaire.

La stratégie dite *input* résout toujours une clause engendrée par résolution (en principe la dernière) avec une clause de l'ensemble S de départ. Cette stratégie a la propriété de ne pas faire exploser l'espace de recherche, elle a donc une importance fondamentale. Elle est malheureusement incomplète, comme on pourra s'en convaincre avec l'exemple $S = \{A \vee B,\ A \vee \neg B,\ \neg A \vee B,\ \neg A \vee \neg B\}$.

5.3 Clauses de Horn et stratégie input

Définition 5.3 *Une clause de Horn est une clause comportant au plus un littéral positif.*

Une remarque essentielle est que le résultat d'une inférence négative mettant en jeu des clauses de Horn est une clause de Horn.

L'étape suivante consiste à préciser ce que sont un programme et une requête en logique de Horn.

Définition 5.4 *Un* programme *en logique de Horn est un ensemble fini de clauses comportant chacune exactement un littéral positif.*

Une requête *est un ensemble fini (disjonction) de littéraux négatifs.*

On en déduit facilement :

Theorem 5.5 *La résolution négative input est complète pour les programmes en logique de Horn.*

Preuve
Il suffit de remarquer que la règle de factorisation est inutile dans le cas des clauses de Horn, et que seules les clauses du programme comportent un littéral positif. □

En pratique, étant donnée une requête $R_1, \ldots, R_p$ (donc chaque R_i est un littéral négatif), la stratégie négative consiste à unifier l'un des R_i avec le littéral positif H d'une clause $H \vee A_1 \vee \ldots \vee A_n$ du programme. Si σ est l'unificateur principal de H et R_i, on infère $A_1\sigma \ldots, A_n\sigma$ qui peuvent être considérés comme de nouvelles requêtes. Comme, par la stratégie, la clause but qui vient d'être engendrée est toujours une des prémisses de la nouvelle inférence, le but R_i peut être effacé. Ceci revient ainsi à réécrire le but (ou plutôt un des littéraux du but) en utilisant une des clauses du programme, vu comme une réécriture de la tête (littéral positif) dans le corps (littéraux négatifs).

On appelle *fonction de sélection* un ordre total sur les littéraux négatifs qui spécifie le but à réécrire en priorité : on résoud toujours sur le but minimal. Cette stratégie est complète (i.e. à chaque étape il suffit de considérer un seul but). Par contre, lors de la recherche d'une preuve, toutes les clauses permettant de réécrire ce but devront être considérées.

La règle de résolution avec une fonction de sélection et pour les programmes logiques est appelée *SLD-résolution*. Des considérations ci-dessus il résulte que :

Theorem 5.6 *La SLD-résolution est complète.*

Enfin, en programmation logique, on s'intéresse aussi au *résultat* d'une requête et, pour des raisons d'efficacité, on ne souhaite pas appliquer les substitutions. Pour ces raisons, on considère des *clauses contraintes*. Une clause contrainte est une paire formée d'une clause C et d'un problème d'unification ϕ et notée $C \mid \phi$. Une telle clause contrainte représente en fait l'ensemble des clauses $\{C\sigma \mid \sigma \models_{T(F)} \phi\}$. ($\sigma \models_{T(F)} \phi$ si σ est une substitution close des variables libres de ϕ et $\phi\sigma$ est valide dans la théorie des termes finis). En général, on remplace aussi $\models_{T(F)}$ par $\models_{RT(F)}$ afin d'éviter d'avoir à faire le test d'occurrence dans l'unification. Dans ce contexte, la règle de résolution peut s'écrire :

$$\frac{P(t) \vee C \qquad \neg B(u) \vee D \mid \phi}{C \vee D \mid \phi \wedge B(u) = P(t)}$$

et il faut alors lui adjoindre de nouveles règles qui ont trait au traitement des contraintes :

$$\begin{array}{lcll} R \mid \phi & \rightarrow & R \mid \phi' & \\ R \mid \bot & \rightarrow & nil & \\ R \mid \phi & \rightarrow & succs(\phi) & \phi \text{ est en forme résolue} \end{array}$$

Les variables de t sont supposées ne pas apparaître libres dans u. De plus les variables n'apparaissant pas dans le but initial et n'apparaissant plus non plus dans le but sont implicitement quantifiées existentiellement (ce qui reflète le fait qu'on ne s'intéresse qu'aux valeurs des variables du but initial).

La *réponse* à une requête $\neg B$ est alors la contrainte ψ telle que l'on ait inféré la clause $\Box \mid \psi$. La contrainte ψ est donnée en forme résolue. Ses solutions sont des substitutions σ telles que $P \models B\sigma$.

Le fait que les contraintes soient ici des équations ne joue de rôle que pour la mise en forme résolue et on peut donc employer d'autres systèmes de contraintes plus expressifs (c'est souvent le cas en pratique). De même les clauses du programme P peuvent être contraintes.

Exemple 5.7 *Soit le programme logique*

$$\begin{array}{l} I(x, [], x.[]). \\ I(x, x.l, x.l). \\ I(x, y.l, y.l') \quad \leftarrow \quad I(x, l, l') \mid x \neq y \end{array}$$

Qui a pour but d'insérer un élément dans une liste sans doublons.

Soit la requête $I(0, s(0).(x.[]), y)$. On obtient deux dérivations possibles conduisant à $\Box \mid \psi$ avec une contrainte ψ satisfiable : la première est (par souci de simplicité, nous n'écrivons la contrainte complète que dans cette séquence, puis nous écrivons une contrainte simplifiée).

$$\begin{array}{rcl} I(0, s(0).(x.[]), y) & \rightarrow & I(x_1, l, l') \mid 0 = x_1 \wedge y_1.l = s(0).(x.[]) \wedge y = y_1.l' \wedge x_1 \neq y_1 \\ & \rightarrow & \Box \mid \exists x_2, l', l'', \wedge y = s(0).l' \wedge x_2 = 0 \wedge x_2.l'' = x.[] \wedge x_2.l'' = l' \\ & \rightsquigarrow & \Box \mid x = 0 \wedge y = s(0).(0.[]) \end{array}$$

La deuxième est :

$$\begin{array}{rcll} I(0, s(0).(x.[]), y) & \rightarrow & I(x_1, l, l') \mid & x_1 = 0 \wedge l = x.[] \wedge y = s(0).l' \\ & \rightarrow & I(x_2, l_1, l'_1) \mid & \exists y_2, l, l', \; l = x.[] \wedge y = s(0).l' \wedge x_2 = 0 \\ & & & \wedge l = y_2.l_1 \wedge l' = y_2.l'_1 \wedge x_2 \neq y_2 \\ & \rightarrow & \Box \mid & \exists x_2, l_1, l'_1, y_2, l, l', \; l_1 = [] \wedge l'_1 = x_2.[] \\ & & & \wedge l = x.[] \wedge y = s(0).l' \wedge x_2 = 0 \wedge l = y_2.l_1 \\ & & & \wedge l' = y_2.l'_1 \wedge x_2 \neq y_2 \\ & \rightsquigarrow & \Box \mid & \wedge y = s(0).(0.(x.[]))) \wedge x \neq 0 \end{array}$$

Nous reformulons ci-dessous les résultats de correction et de complétude de la SLD-résolution en termes de requêtes et de programmes :

Proposition 5.8 *Si la réponse ψ à une requête R pour un programme logique P a pour mgu σ, alors, pour toute substitution close θ, $R\sigma\theta, P$ est insatisfiable.*

Proposition 5.9 *Si R est une requête pour un programme logique P et si $R\theta, P$ est insatisfiable, alors il existe une réponse ψ à R telle que ψ a pour mgu σ et il existe une substitution τ telle que $\sigma\tau = \theta$.*

5.4 Plus petit modèle de Herbrand et sémantique des programmes logiques

Si P est un programme logique. On note $\mathcal{B}$ la base de Herbrand et $2^{\mathcal{B}}$ l'ensemble des parties de $\mathcal{B}$, que l'on confond aussi avec l'ensemble des interprétations de Herbrand (les éléments de $I \in 2^{\mathcal{B}}$ correspondent aux éléments de la base qui s'interprètent en vrai, les autres s'inteprétant à faux). On note M_P le "plus petit modèle de Herbrand de P", c'est-à-dire l'intersection de tous les modèles de Herbarnd de P :

$$M_P = \bigcap_{I \in 2^{\mathcal{B}}, I \models P} I.$$

Exemple 5.10 *Soit P le programme :*

$$Q(s(x)) \vee \neg Q(x)$$
$$P(0) \vee \neg Q(s(x))$$

Alors $M_P = \emptyset$ car l'interprétation vide (tout est faux) satisfait bien P.

Proposition 5.11 *Pour tout programme logique P, $M_P \models P$.*

Preuve

Il suffit de remarquer que l'intersection de modèles de Herbrand de P est aussi un modèle de Herbrand de P. On fait le raisonnement sur des clauses closes pour simplifier. Soit $A \vee \neg B_1 \vee \ldots \vee \neg B_n$ une clause C. Il y a deux cas.

$A \in M_P$. Alors $M_P \models C$.

$A \notin M_P$. Alors, par définition de M_P, il existe un modèle de Herbrand I de P tel que $A \notin I$. Mais comme $I \models C$, il existe j tel que $B_j \notin I$, et donc $B_j \notin M_P$ par définition de M_P, d'où $M_P \models C$. □

Ainsi M_P consititue une interprétation "standard" du programme logique P. De façon classique, on peut aussi construire M_P à l'aide de l'opérateur de conséquence immédiate :

Si P est un programme logique, on note T_P l'*opérateur de conséquence immédiate* défini sur $2^{\mathcal{B}}$ par

$$T_P(I) = \{A \in \mathcal{B} \mid A \vee \neg A_1 \vee \ldots \vee \neg A_n \text{ est une instance close d'une clause de } P, \text{ et} A_1, \ldots, A_n \in I\}$$

Exemple 5.12 *Reprenons l'exemple 5.10. Alors $T_P(\emptyset) = \emptyset$, $T_P(\mathcal{B}) = \{P(0)\} \cup \{Q(s(t)) \;\; t \in T(\mathcal{F})\}$, $T_P(T_P(\mathcal{B})) = \{P(0)\} \cup \{Q(s(s(t))) \;\; t \in T(\mathcal{F})\}$, etc.*

$2^{\mathcal{B}}$ est muni d'une structure de treillis complet pour l'ordre $\subseteq$ (i.e. l'ordre de prolongement des interprétations).

Proposition 5.13 *T_P est continu sur les interprétations de Herbrand.*

Preuve
Tout d'abord, T_P est une fonction croissante, la preuve en est laissée au lecteur.

Soit donc une famille croissante de parties de $\mathcal{B}$, $\{I_k\}_{k\in\mathbb{N}}$, dont le sup, le treillis des parties étant complet pour l'ordre, est noté $S = \bigcup_k I_k$. T_P étant croissante, $\{T_P(I_k)\}_{k\in\mathbb{N}}$, est une autre famille croissante dont le sup est noté $\bigcup_k T_P(I_k)$.

L'équation $T_P(\bigcup_k I_k) = \bigcup_k T_P(I_k)$ exprime la continuité de l'opérateur T_P.

Pour cela,

$$\begin{aligned} T_P(\bigcup_{I\in S} I) = & \{A \in \mathcal{B} \mid A \vee \neg A_1 \vee \ldots \vee \neg A_n \text{ est une instance close} \\ & \text{d'une clause de } P \text{ et } \forall i A_i \in \bigcup_{I\in S} I\} \\ = & \{A \in \mathcal{B} \mid \forall i \exists I \in S, A \vee \neg A_1 \vee \ldots \vee \neg A_n \text{ est une instance close} \\ & \text{d'une clause de } P \text{ et } A_i \in I\} \\ = & \{A \in \mathcal{B} \mid \exists I \in S, A \vee \neg A_1 \vee \ldots \vee \neg A_n \text{ est une instance close} \\ & \text{d'une clause de } P \text{ et } \forall i A_i \in I\} \end{aligned}$$

Notons l'utilisation du fait que $\{I_k\}_k$ est une chaine croissante pour faire commuter les quantificateurs. □

Il est possible de généraliser la définition de T_P aux clauses arbitraires :

$$T_P(I) = \{A \in \mathcal{B} \mid A \vee C \text{ est une instance close d'une clause de } P \text{ et } I \models \neg C\}$$

Mais dans ce cas T_P n'est plus nécessairement continu.

Exemple 5.14 *Supposons P réduit à la seule clause $A \vee B$. $T_P(\emptyset) = \{A, B\}$ et $T_P(\{A, B\}) = \emptyset$, $T_P(\{A\}) = \{A\}$ et $T_P(\{B\}) = \{B\}$. Par exemple pour la partie dirigée $S = \{\emptyset, \{A\}\}$, $T_P(\bigcup_{I\in S} I) = T_P(\{A\}) = \{A\} \neq \{A, B\} = T_P(\emptyset) = \bigcup_{I\in S} T_P(I)$.*

Proposition 5.15 *Soit I une interprétation de Herbrand et P un programme logique. I est un modèle de P si et seulement si $T_P(I) \subseteq I$.*

Preuve

$I \models P$ ssi pour toute instance close $A \vee \neg A_1 \vee \ldots \vee \neg A_n$ d'une clause de P, $I \models A_1, \ldots, A_n$ implique $I \models A$. □

Theorem 5.16 *Si P est un programme logique, $M_P = T_P^\omega(\emptyset)$ est le plus petit point fixe de T_P.*

Preuve

Le théorème du point fixe nous assure que $T_P^\omega(\emptyset)$ est le plus petit point fixe de l'opérateur continu T_P. Il nous reste à montrer que M_P est égal au plus petit point fixe de T_P.

Tout point fixe de T_P est un modèle de P par la proposition 5.15, et il nous sufit donc de montrer que M_P est un point fixe de T_P de mani'ere à conclure par minimalité de M_P.

M_P étant un modèle de P, $T_P(M_P) \subseteq M_P$ par la proposition 5.15. D'autre part, d'après la proposition 5.13 on a aussi $T_P^2(M_P) \subseteq T_P(M_P)$ et donc $T_P(M_P)$ est un modèle de P par la proposition 5.15. Par minimalité de M_P on a donc $M_P \subseteq T_P(M_P)$ et donc $T_P(M_P) = M_P$. □

De façon générale, lorsque T_P est monotone, T_P^α pour un ordinal α est défini par récurrence transfinie par : $T_P^0 = \emptyset$, si $T_P^{\alpha+1} = T_P(T_P^\alpha)$ et, si α est un ordinal limite, T_P^α est la borne supérieure des T_P^β pour $\beta < \alpha$. C'est la continuité qui implique que $\alpha = \omega$.

5.5 Exercices

5.5.1

1. La stratégie *unitaire* consiste à ne considérer que des résolutions dont une des prémisses est une clause réduite à un littéral.

   Montrer que la stratégie unitaire est en général incomplète. (Autrement dit, donner un ensemble de clauses insatifiable tel que factorisation + résolution unitaire ne permet pas de dériver la clause vide).

   Plus généralement, montrer qu'il existe une réfutation d'un ensemble S de clauses par la résolution unitaire si et seulement si il existe une réfutation de S par la stratégie "input".

   La stratégie unitaire est donc complète pour les clauses de Horn. Est-ce une stratégie intéressante ?

2. Donner un ensemble de clauses tel que T_P^ω n'est pas un point fixe alors que $T_P^{2\times\omega}$ est un point fixe.

Bibliographie

[1] K. R. Apt and M. H. van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3), July 1982.

[2] Alain Colmerauer. Prolog II. Manuel de référence et modèle théorique. Research report, GIA Luminy, Marseille, March 1982.

[3] Alain Colmerauer. Equations and inequations on finite and infinite trees. In *FGCS'84 Proceedings*, pages 85–99, November 1984.

[4] Alain Colmerauer. Opening the Prolog III universe. *Byte Magazine*, August 1987.

[5] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.

[6] Gérard Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots \omega$. Thèse d'Etat, Univ. Paris 7, 1976.

[7] Jean-Pierre Jouannaud and Claude Kirchner. Solving equations in abstract algebras : A rule-based survey of unification. In Jean-Louis Lassez and Gordon Plotkin, editors, *Computational Logic : Essays in Honor of Alan Robinson*, pages 257–321. MIT-Press, 1991.

[8] Robert Kowalski. Semantic trees in automatic theorem-proving. *Machine Intelligence*, 4 :86–101, 1969.

[9] Robert Kowalski. Search strategies for theorem-proving. *Machine Intelligence*, 5 :181–201, 1970.

[10] Robert A. Kowalski and M. H. van Emden. The semantics of predicate logic as a programming language. *J. of the Association for Computing Machinery*, 23 :733–742, October 1976.

[11] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2) :258–282, April 1982.

[12] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1) :23–41, 1965.

[13] J.A. Robinson. The generalized resolution principle. *Machine Intelligence*, 3, 1968.