

Chapitre 1

La Logique : Historique et Principes

Avertissement : cette introduction n'est pas rédigée

1.1 Introduction

Les mathématiques s'intéresse à l'étude d'objets comme les nombres, la géométrie,... en utilisant le raisonnement mathématique.

La Logique est la science de l'étude du raisonnement.

En particulier une question a été : les mathématiques usuelles et autres ont-elle un fondement logique ?

Des questions célèbres ont un lien très fort avec la logique.

- L'axiome des parallèles dans la géométrie euclidienne.
- Constructivisme (Brouwer) n'admet que les objets qu'il peut construire
- La théorie des ensemble est-elle consistante ?

1.2 Historique

- Les grecs (syllogisme, paradoxes,...)
- Moyen-age (rasoir d'Ockham,...)
- Leibniz (tout est calcul algébrique)
- calcul Booléen (Boole)
- Logique du premier ordre (Frege)
- Théorème de Godel, calculabilité (Turing, Church, Kleene,...)
- Informatique et Logique.
 - Types as proposition, Programs as proof (Curry-Howard)
 - Démonstration automatique (résolution, Prolog)
 - Logique et vérification.
 - Modélisation des connaissances

1.3 Les constituants d'une logique

- Langage : ce qui permet de définir les formules.
- Sémantique : ce qui donne un sens aux formules
- Syntaxe ou système formel = système de calcul purement syntaxique sur les formules formé de
 - Axiomes
 - Règles d'inférence

Chapitre 2

Calcul Propositionnel

2.1 Le langage

Pour éviter les paradoxes comme celui du barbier, liés à la confusion entre le langage (de la logique) et le méta-langage (celui qui sert à parler de la logique), nous devons définir précisément notre langage qui sert à écrire les formules.

Definition 1 *Langage du CP0 est formé de :*

- symboles propositionnels $\{p_1, p_2, \dots\}$ (ou atomes)
- connecteurs logiques $\{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$
- symboles auxiliaires $'(, ', ', ', \dots$

PROP denote l'ensemble des symboles propositionnels qui sera supposé fini pour simplifier.

Definition 2 *L'ensemble FORM des formules est le plus petit ensemble tel que*

- tout atome est une formule
- si φ est une formule alors $\neg\varphi$ est une formule
- si φ, ψ formules alors $\varphi \circ \psi$ est une formule.

Les symboles auxiliaires sont utilisés pour lever les ambiguïtés possibles ($(p \vee q) \wedge r$ au lieu de $p \vee q \wedge r$).

Exemple $p, p \Rightarrow (q \vee r), p \vee q$ sont des formules, $\neg(\vee q), f(x) \Rightarrow g(x)$ ne sont pas des formules.

Definition 3 *La notion de sous-formule est définie par :*

- p est une sous-formule de p ,
- une sous-formule de $\neg\varphi$ est $\neg\varphi$ ou une sous-formule de φ
- une sous-formule de $(\varphi) \circ (\psi)$ est $(\varphi) \circ (\psi)$ ou une sous-formule de ψ ou de φ (où \circ désigne un des symboles $\wedge, \vee, \Rightarrow, \Leftrightarrow$).

ψ est une sous-formule stricte de φ si ψ est une sous-formule de φ qui n'est pas φ .

$SF(\phi)$ est l'ensemble des sous-formules de φ

$SF(p \Rightarrow (q \vee r)) = \{p, q, r, q \vee r, p \Rightarrow (q \vee r)\}$

2.2 Sémantique

2.2.1 Valuation et modèle

Definition 4 Une valuation v est une application de PROP dans $\{0, 1\}$ (valeurs de vérité, 0 pour faux, 1 pour vrai) qu'on étend en une application de FORM dans $\{0, 1\}$ par morphisme :

$$v(\neg\varphi) = 1 - v(\varphi)$$

$$v(\varphi \vee \psi) = \max(v(\varphi), v(\psi))$$

$$v(\varphi \wedge \psi) = \min(v(\varphi), v(\psi))$$

$$v(\varphi \Rightarrow \psi) = 0 \text{ ssi } v(\varphi) = 1 \text{ et } v(\psi) = 0$$

$$v(\varphi \Leftrightarrow \psi) = 1 \text{ ssi } v(\varphi) = v(\psi)$$

Ce qui correspond aux tables de vérité des connecteurs logiques.

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

p	$\neg p$
0	1
1	0

p	q	$p \Rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

p	q	$p \Leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

Definition 5 Une valuation est un modèle de φ si et seulement si $v(\varphi) = 1$.

Terminologie :

- Satisfaisable : formule vraie pour au moins une valuation (c.a.d. qui a un modèle).
- Insatisfaisable : formule fausse pour toute valuation (c.a.d. qui n'a pas de modèle).
- Tautologie : formule vraie pour toute valuation. On note $\models \varphi$ pour dire que φ est une tautologie.
- Γ ensemble de formules est consistant s'il existe une valuation modèle de toutes les formules de Γ .

- Γ ensemble de formules, φ est une conséquence logique de Γ si toute valuation qui donne 1 à toutes les formules de Γ donne 1 à φ .

On étend les définitions aux ensembles de formules. Par exemple un modèle de Γ est une valuation v qui donne 1 à chaque formule de Γ .

2.2.2 Ensemble des modèles

L'ensemble des modèles de φ est noté $Mod(\varphi)$. C'est l'ensemble des valuations qui évaluent φ comme vraie. On étend la définition à un ensemble de formules.

Certaines propriétés logiques s'expriment bien en terme de relations entre modèles.

- φ est conséquence logique de Γ si et seulement si $Mod(\Gamma) \subseteq Mod(\varphi)$
- $\models \varphi \Rightarrow \psi$ si et seulement si $Mod(\varphi) \subseteq Mod(\psi)$.
- φ insatisfaisable ssi $Mod(\varphi) = \emptyset$.

2.2.3 Décidabilité de la logique

La définition suivante n'est pas complètement rigoureuse mais est suffisamment intuitive pour saisir le concept. Une logique est *décidable* s'il existe un algorithme (calcul réalisable sur un ordinateur qui termine toujours pour toute donnée) qui permet de savoir pour chaque formule si elle est une tautologie (i.e. $\models \varphi$) ou pas.

Theorème 1 *Le CP0 est décidable.*

Méthode des tables de vérité : calculer la table de vérité prenant en argument les symboles propositionnels de φ et calculer pour chaque valuation possible la valeur de v sur les sous-formules de φ .

Coût : $O(2^n)$ avec n la taille de φ (nombre de symboles).

Question : faire mieux ? en temps polynomial ?

- Il y a des algorithmes meilleurs en pratique mais qui ont toujours un coût exponentiel.
- Si la conjecture $P \neq NP$ est vraie alors il n'y a pas d'algorithme polynomial.

La méthode de résolution est un calcul qui permet de montrer si une formule est une tautologie. Il y a des formules pour lesquelles ce calcul demande un temps exponentielle, mais en général il est plus rapide que la méthode des tables de vérité.

2.2.4 Formalisation de problème et expressivité

Le règlement de l'association des logiciens du CMI

- Les membres de la direction financière sont choisis parmi ceux de la direction générale.
- Nul ne peut être à la fois membre de la direction générale et de la direction de la bibliothèque s'il n'est membre de la direction financière.

- Aucun membre de la direction de la bibliothèque ne peut être membre de la direction financière.

peut se modéliser en :

- G : être membre de la direction générale
- F : être membre de la direction financière
- B : être membre de la direction de la bibliothèque

$$F \Rightarrow G, G \wedge B \Rightarrow F, \neg(B \wedge F)$$

Ce qui donne un règlement plus simple

$$F \Rightarrow G, \neg(G \wedge B)$$

2.3 Manipulation de formules

Les formules équivalentes sont des formules indistingables par la sémantique, i.e. elles ont la même table de vérité bien que pouvant être syntaxiquement différentes.

Definition 6 Les formules φ et ψ sont équivalentes noté $\psi \equiv \varphi$ ssi $Mod(\psi) = Mod(\varphi)$.

La définition équivaut à $\models \psi \Leftrightarrow \varphi$.

Exemple : Les opérateurs \wedge, \vee sont associatifs-commutatifs et deux formules identiques à associativité-commutativité près sont équivalentes.

Remplacer une sous-formule ψ d'une formule φ par une formule équivalente ψ' donne une formule notée $\varphi[\psi \leftarrow \psi']$. Cette substitution préserve les modèles, i.e. $Mod(\varphi) = Mod(\varphi[\psi \leftarrow \psi'])$.

La mise en forme clausale (ou forme normale conjonctive) sert à transformer une formule en une formule équivalente normalisée plus adaptée aux calculs.

Forme normale négative : toutes les négations sont devant des atomes. Litéral : atome ou négation d'atome.

Clause : disjonction de littéraux.

Forme normale conjonctive (fnc) ou forme clausale : conjonction de clauses.

Forme normale disjonctive (fnd) : disjonction de conjonction d'atome. C'est la duale de la fnc.

Mise en forme Clausale :

$$\begin{aligned} \text{Etape1 : } \varphi \Leftrightarrow \psi &\rightarrow (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi) \\ \varphi \Rightarrow \psi &\rightarrow \neg\varphi \vee \psi \end{aligned}$$

$$\begin{aligned} \text{Etape2 } \neg\neg\varphi &\rightarrow \varphi \\ \neg(\varphi \wedge \psi) &\rightarrow \neg\varphi \vee \neg\psi \\ \neg(\varphi \vee \psi) &\rightarrow \neg\varphi \wedge \neg\psi \end{aligned}$$

$$\begin{aligned} \text{Etape3 } \varphi \vee (\psi_1 \wedge \psi_2) &\rightarrow (\varphi \vee \psi_1) \wedge (\varphi \vee \psi_2) \\ (\psi_1 \vee \psi_2) \wedge \varphi &\rightarrow (\psi_1 \vee \varphi) \wedge (\psi_2 \vee \varphi) \end{aligned}$$

Proposition 1 *Le calcul précédent termine et donne une formule en forme clausale équivalente à la formule initiale.*

Il faut noter qu'il n'y a pas unicité de la forme clausale.

2.4 Un système formel pour le CPO : la résolution

2.4.1 Définition d'un système formel

Un système formel définit des règles de calcul entre formules qui simulent le raisonnement, ce qui permet de déduire de nouvelles formules. Il est défini par

- le langage et les formules sur ce langage,
- les axiomes : formules ou schémas de formules supposées vraies,
- les règles d'inférence : règles permettant de déduire une nouvelles formule (conclusion) à partir d'un ensemble de formules (prémises ou hypothèses).

On note $\vdash \varphi$ si φ peut se déduire dans le système formel considéré. Deux questions fondamentales sont systématiquement posées pour relier le calcul et la sémantique :

1. Le système est-il correct : une formule déduite est-elle une tautologie ?
2. Le système est-il complet : toute tautologie peut-elle se déduire ?

2.4.2 La méthode de résolution

La méthode de résolution est un système formel qui utilise des formules sous formes de clauses, n'a pas d'axiomes, et comporte deux règles d'inférence. Elle est réfutationnelle : pour montre qu'on peut prouver φ on suppose $\neg\varphi$ et on montre que cela conduit à une absurdité.

- Mettre $\neg\varphi$ en forme clausale.
- Remplacer la conjonction de clauses $C_1 \wedge \dots \wedge C_n$ par l'ensemble $\mathcal{C} = \{C_1, \dots, C_n\}$.
- Saturer l'ensemble en rajoutant à \mathcal{C} les clauses qu'on peut déduire avec les règles

$$(Resolution) \frac{l \vee C \quad \neg l \vee C'}{C \vee C'}$$

avec l un littéral. La clause $C \vee C'$ est appelée la résolvente.

$$(Factorisation) \frac{l \vee C_1 \vee l \vee C_2}{C_1 \vee l \vee C_2}$$

avec l un littéral.

L'écriture des règles est modulo associativité-commutativité de \vee (ce qui veut dire qu'une clause est vue comme le multi-ensemble de ses littéraux (attention pas l'ensemble). Par exemple $p \vee p \vee \neg q$ correspond à $\{p, p, \neg q\}$. On s'arrête quand l'application des règles engendre la clause vide \square (qui signifie une inconsistance, et est fausse pour toute valuation).

On note $\vdash \varphi$ si on peut dériver \square à partir de la forme clausale de $\neg\varphi$ par résolution. On dit alors que la formule φ est un *théorème* ou qu'elle *est prouvable*.

Exemple $\varphi = ((p \Rightarrow q) \wedge (q \Rightarrow p) \wedge (p \vee q)) \Rightarrow (p \wedge q)$, alors $\neg\varphi$ est :

$$\neg[(p \Rightarrow q) \wedge (q \Rightarrow p) \wedge (p \vee q)] \Rightarrow (p \wedge q)$$

- La mise en forme clausale donne $(p \vee \neg q) \wedge (\neg p \vee q) \wedge (p \vee q) \wedge (\neg p \vee \neg q)$
- L'ensemble de clauses est :

$$C = \{p \vee \neg q, \neg p \vee q, p \vee q, \neg p \vee \neg q\}$$

Une preuve par résolution est :

$$\frac{\frac{\frac{p \vee \neg q \quad \neg q \vee \neg p}{\neg q \vee \neg q}}{\neg q} \quad \neg p \vee q}{\frac{\neg p \quad p \vee q}{q}} \quad \neg q}{\square}$$

L'exemple montre qu'on peut avoir besoin d'utiliser plusieurs fois une clause pour dériver la clause vide : ici $\neg q$ (qu'on a dérivée par résolution et factorisation) est utilisée deux fois.

On peut remarquer que la règle de factorisation est nécessaire dans l'exemple : sans cette règle toutes les résolvantes ont deux littéraux et on ne peut donc pas dériver la clause vide.

2.4.3 Terminaison

La méthode précédente n'a pas de critère de terminaison autre que *l'ensemble des clauses contient* \square . Si la formule à démontrer n'est pas un théorème, il faut soit montrer à la main qu'il n'est pas possible d'engendrer des résolvantes qui ne sont pas déjà dans l'ensemble de clauses, soit avoir un critère permettant d'arrêter de générer de nouvelles résolvantes (souvent en utilisant des *stratégies de résolution*).

A la main, sur des exemples simples on peut souvent arrêter la résolution car on obtient un ensemble de clauses (clauses initiales plus toutes les clauses déduites) pour lequel on trouve un modèle. Ce modèle est donc un contre-exemple à la formule initiale qui n'est donc pas prouvable (c'est une conséquence des deux propriétés de complétude et correction qui suivent).

Exemple $C = \{p \vee \neg q, q \vee \neg p\}$

La méthode engendre $\{p \vee \neg p, q \vee \neg q, p \vee \neg q, q \vee \neg p\}$ (et rien de plus, mais il faut un raisonnement pour le prouver !) qui a un modèle $v(p) = 1, v(q) = 1$. Donc la formule $\neg((p \vee \neg q) \wedge (q \vee \neg p))$ n'est pas une tautologie.

Exemple $C = \{p \vee \neg p \vee \neg q\}$

La méthode engendre $p \vee \neg p \vee \neg q \vee \neg q$ et de manière générale les clauses de la forme $p \vee \neg p \vee \neg q \vee \neg q \vee \dots \vee \neg q$ et aucune autre (la encore un raisonnement est à faire). Donc la résolution ne termine pas et engendre un ensemble de clauses infini $\{p \vee \neg p \vee \neg q, p \vee \neg p \vee \neg q \vee \neg q, \dots\}$ qui ne contient pas la clause vide. On peut conclure soit en trouvant directement un modèle (on peut en trouver plusieurs ici) soit en appliquant un résultat qui va être démontré dans la preuve de complétude : un ensemble de clauses saturé par résolution qui ne contient pas la clause vide a un modèle. Donc la formule $\neg(p \vee \neg p \vee \neg q)$ n'est pas une tautologie.

2.5 Correction et complétude

2.5.1 Correction

Ce qui est prouvable est vrai.

Théorème 2 $Si \vdash \varphi$ alors $\models \varphi$.

$\neg\varphi$ est équivalente à la conjonction de clauses obtenue par mise en forme clausale.

Par construction les clauses ajoutées sont des conséquences logiques de l'ensemble de clause initial. Donc dériver la clause vide signifie que celui-ci est insatisfaisable.

□

2.5.2 Complétude

Ce qui est vrai est prouvable.

Théorème 3 $Si \models \varphi$ alors $\vdash \varphi$.

La preuve est donnée pour un ensemble PROP fini. Pour le cas PROP infini dénombrable, il faut soit utiliser une branche infinie lors de la construction du modèle, soit utiliser d'abord le théorème de compacité (voir section ??) et se ramènera la preuve donnée ici. On montre $\not\vdash \varphi$ implique $\not\models \varphi$.

On montre d'abord : *un ensemble de clauses saturé \mathcal{C} est insatisfaisable si et seulement si il contient la clause vide.*

Sens \Leftarrow : immédiat.

Cas \Rightarrow . On raisonne par contraposée. On suppose que \mathcal{C} est saturé et ne contient pas la clause vide et on va montrer que \mathcal{C} a un modèle.

La preuve est par récurrence sur le nombre n de symboles propositionnels de \mathcal{C} .

– Cas de base : $n=1$. \mathcal{C} ne peut contenir p et $\neg p$ car il est saturé, donc il existe un modèle.

- On suppose la propriété vraie pour n .
Soit p un symbole propositionnel apparaissant dans une clause de \mathcal{C} . On définit :
 - \mathcal{C}_0 l'ensemble des clauses qui ne contiennent pas le literal $\neg p$ et \mathcal{C}'_0 l'ensemble des clauses de \mathcal{C}_0 obtenues en remplaçant p par 0.
 - \mathcal{C}_1 l'ensemble des clauses qui ne contiennent pas le literal p et \mathcal{C}'_1 l'ensemble des clauses de \mathcal{C}_0 obtenues en remplaçant p par 1.

Un des \mathcal{C}'_i est non vide et ne contient pas la clause vide, sinon \mathcal{C} contiendrait p et $\neg p$ et donc \square . Par hypothèse d'induction il a un modèle sur $\text{PROP} \setminus \{p\}$. Sans perte de généralité, supposons que c'est \mathcal{C}_1 . Alors l'extension du modèle de \mathcal{C}'_1 en posant $v(p) = 1$ est un modèle de \mathcal{C} .

Si l'ensemble saturé obtenu par résolution à partir de $\neg\varphi$ ne contient pas \square alors $\neg\varphi$ est satisfaisable et donc $\not\models \varphi$.

\square

2.6 Traitement de la conséquence logique

2.6.1 Théorème de compacité

Théorème 4 *Un ensemble (fini ou infini) de formules Γ a un modèle si et seulement si tout sous-ensemble fini de formules de Γ a un modèle.*

On suppose connu le théorème de Koenig : *un arbre à branchement fini dont toutes les branches sont finies est fini.*

On suppose que Γ est insatisfaisable mais tous ses sous-ensembles finis sont satisfaisable. On construit l'arbre des valuations sur PROP.

Un noeud n de l'arbre est un noeud d'échec s'il existe une formule $\varphi(n)$ de Γ sur p_1, \dots, p_n falsifié par la valuation définie par le chemin de la racine au noeud.

- Si aucun noeud d'échec existe sur une branche alors on a un modèle de Γ , contradiction.
- Sinon on considère l'arbre obtenu en coupant tous les noeuds sous un noeud d'échec. Cet arbre est à branchement fini, ses branches sont finies donc il est fini. L'ensemble des formules étiquettant les feuilles est fini et n'a aucun modèle par construction, contradiction.

2.6.2 Conséquence Logique

Une formule close φ est conséquence logique d'un ensemble de formules closes si φ est vrai dans tout modèle de Γ , écrit $\Gamma \models \varphi$.

On adapte la résolution en mettant toutes les formules de Γ et $\neg\varphi$ en forme clausale, puis en déduisant la clause vide par résolution et factorisation. On note $\Gamma \vdash \varphi$ si on peut déduire la clause vide à partir de $\Gamma \cup \{\neg\varphi\}$.

Proposition 2 *Soit Γ un ensemble de formules, alors $\Gamma \models \varphi$ ssi $\Gamma \vdash \varphi$.*

On montre en fait $\Gamma \models \varphi$ ssi il existe Γ_f fini inclus dans Γ tel que $\Gamma_f \vdash \varphi$.

Par définition $\Gamma \models \varphi$ ssi $\Gamma \cup \{\neg\varphi\}$ est insatisfaisable.

D'après le théorème de compacité $\Gamma \cup \{\neg\varphi\}$ est insatisfaisable ssi un sous-ensemble fini Δ de $\Gamma \cup \{\neg\varphi\}$ est insatisfaisable.

On peut prendre $\Delta = \Gamma_f \cup \{\neg\varphi\}$ pour un ensemble Γ_f fini contenu dans Γ (si Δ ne contenait pas $\neg\varphi$, on peut toujours lui rajouter $\neg\varphi$, il restera insatisfaisable).

$\Gamma_f \cup \{\neg\varphi\}$ est insatisfaisable ssi

$\models \bigwedge_{\gamma \in \Gamma_f} \gamma \Rightarrow \varphi$

D'après le théorème de complétude, $\models \bigwedge_{\gamma \in \Gamma_f} \gamma \Rightarrow \varphi$ ssi $\vdash \bigwedge_{\gamma \in \Gamma_f} \gamma \Rightarrow \varphi$.

La forme clause de $\neg(\bigwedge_{\gamma \in \Gamma_f} \gamma \Rightarrow \varphi)$ est l'ensemble des clauses des formes clauseuses de $\gamma \in \Gamma_f$ et de $\neg\varphi$. Par conséquent $\vdash \bigwedge_{\gamma \in \Gamma_f} \gamma \Rightarrow \varphi$ ssi $\Gamma_f \vdash \varphi$. \square

Résumé

atome : symbole propositionnel

littéral : atome ou négation d'un atome

clause : disjonction de littéraux

modèle : une valuation v qui donne 1

$Mod(\varphi)$ l'ensemble des modèles de φ .

satisfaisable : il existe un modèle ($Mod(\varphi) \neq \emptyset$)

insatisfaisable : n'admet pas de modèle ($Mod(\varphi) = \emptyset$)

tautologie : toute valuation donne 1

φ conséquence logique de Γ : toute valuation qui donne 1 aux formules de Γ donne 1 à φ

Une logique comporte une syntaxe (pour définir les formules), une sémantique (pour définir le sens d'une formule), un système formel (un calcul pour prouver une formule).

Un calcul est correct s'il ne prouve pas de formules qui ne sont pas vraies

Un calcul est complet si tout ce qui est vrai peut se prouver.

La méthode de résolution est correcte et complète.

Chapitre 3

Termes et Unification

Avertissement : ce cours n'est pas rédigé en totalité

3.1 Les Termes

3.1.1 Définitions

Les termes se définissent avec :

- un ensemble dénombrable de variables X ,
- un ensemble fini F de symboles de fonctions, chaque symbole ayant une arité fixe.

Les symboles d'arité 0 sont appelés constante et notés c, \dots au lieu de $c(), \dots$

L'ensemble $T_F(X)$ des termes est le plus petit ensemble tel que

- $x \in X$ alors $x \in T_F(X)$,
- si $f \in F$, d'arité $n \geq 0$, et $t_1, \dots, t_n \in T_F(X)$ alors $f(t_1, \dots, t_n) \in T_F(X)$

Les termes clos sont les termes qui ne contiennent pas de variables. L'ensemble des termes clos est noté T_F . $Var(t)$ est l'ensemble des variables apparaissant dans t .

Exemples Un terme correspond à un arbre étiqueté :

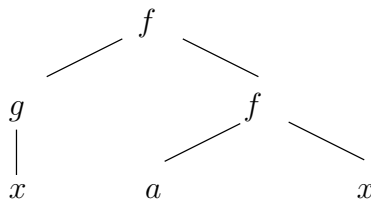


FIG. 3.1 – L'arbre de $f(g(x), f(a, x))$

3.1.2 Positions, substitutions

Position : un position est une suite d'entier qui permet de naviguer dans un terme, la suite vide est notée ϵ . Pos est l'ensemble des positions et une position p est $p = \epsilon$, ou $p = i.p'$ avec $i \in \mathcal{N}^+$, $p' \in Pos$.

Le sous terme de t à la position p est défini par

- $t_{|\epsilon} = t$,
- $t_{|i.p} = t_i|_p$ si $t = f(t_1, \dots, t_n)$, avec $1 \leq i \leq n$,
- sinon $t_{|p}$ est non-défini.

Une position p est une position de t si $t_{|p}$ est défini. L'ensemble des positions d'un terme t est noté $Pos(t)$.

Definition 7 Une substitution $\sigma = \{x_1 \leftarrow s_1, \dots, x_n \leftarrow s_n\}$ est une application de $T_F(X)$ dans $T_F(X)$ définie par :

- $\sigma(x_i) = s_i$ $i = 1, \dots, n$,
- $\sigma(x) = x$ si $x \neq x_i$ $i = 1, \dots, n$,
- $\sigma(f(t_1, \dots, t_n)) = f(\sigma(t_1), \dots, \sigma(t_n))$.

Le domaine de σ est $Dom(\sigma) = \{x_1, \dots, x_n\}$

En particulier, si c est une constante alors $\sigma(c) = c$. L'application de la substitution σ au terme t remplace chaque occurrence de variable par son image par σ . Une substitution est un morphisme c'est à dire une application telle que $\sigma(f(t_1, \dots, t_n)) = f(\sigma(t_1), \dots, \sigma(t_n))$ pour tout symbole $f \in \mathcal{F}$ d'arité n .

La substitution identité id est celle qui ne modifie aucune variable, i.e. $Dom(id) = \emptyset$.

Exemples On donne la substitution $\sigma_1 = \{x \leftarrow a, y \leftarrow g(x), z \leftarrow b\}$ et les termes $s = f(x, g(y))$, $t = f(f(x, x), f(a, z))$.

Alors $\sigma_1(s) = f(a, g(x))$ et $\sigma_1(t) = f(f(a, a), f(a, b))$ \square .

3.2 Unification

L'unification est l'opération qui permet de calculer les valeurs des variables qui permettent de rendre identiques deux termes. La mise au point d'algorithmes d'unification efficaces a permis de définir des méthodes de démonstration automatique qui sont effectives.

Une substitution σ est plus générale qu'une substitution θ s'il existe une substitution ρ telle que $\theta = \rho \circ \sigma$.

Definition 8 Unificateur : une substitution σ est un unificateur de s et t ssi $\sigma(s) = \sigma(t)$.

Cette notion est très importante pour définir la résolution au premier ordre.

Definition 9 Une équation est une expression $s = t$ avec $s, t \in T_F(X)$. Un problème d'unification est un ensemble d'équations ou bien \perp (echec, le système sans solution).

Une *solution* de $P = \{s_1 = t_1, \dots, s_n = t_n\}$ est une substitution σ telle que $\sigma(s_i) = \sigma(t_i)$ pour $i = 1, \dots, n$.

Une variable x est dite *résolue* dans P si P contient une équation $x = t$ et si x n'apparaît qu'une seule fois dans P . Un problème d'unification est en forme résolue s'il peut s'écrire $\{x_1 = t_1, \dots, x_n = t_n\}$ avec x_i pour $i = 1, \dots, n$ des variables n'apparaissant qu'une seule fois dans le système. Une forme résolue définit une substitution $\sigma = \{x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n\}$ qui est le plus grand unificateur.

Algorithme d'unification de Robinson. Il est donné par les règles suivantes :

Clash $P, f(t_1, \dots, t_n) = g(s_1, \dots, s_m) \rightarrow \perp$

Simplification $P, f(t_1, \dots, t_n) = f(s_1, \dots, s_n) \rightarrow P, t_1 = s_1, \dots, t_n = s_n$

Occur-check $P, x = t \rightarrow \perp$ si $x \in \text{Var}(t)$ et t n'est pas une variable

Simplification des équations triviales $P, t = t \rightarrow P$

Remplacement $P, x = t \rightarrow P_{x \leftarrow t}, x = t$ si aucune autre règle ne s'applique et x n'est pas résolue (on remarque que x devient résolue).

L'algorithme original correspond à une certaine stratégie d'utilisation de ces règles, mais celui donné ici est plus général.

Proposition 3 (Correction) Si $P \rightarrow P'$ par une des règles précédentes, alors si σ est une solution de P' alors σ est une solution de P .

Par définition des règles et de l'application d'une substitution. \square

Proposition 4 (Complétude) Si $P \rightarrow P'$ par une des règles précédentes, alors si σ est une solution de P alors σ est une solution de P' .

Par définition des règles et de l'application d'une substitution. \square

Proposition 5 Toute application des règles termine soit avec \perp (et le problème n'a pas de solution) soit avec une forme résolue qui définit le plus grand unificateur solution du système.

La mesure de complexité $\mu(P)$ d'un système P est une paire d'entiers naturels ($nbvarnr, nbsym$) avec

- $nbvarnr$ le nombre de variable non résolues
 - $nbsym$ le nombre de symboles (de fonction ou variables) de P
- L'ordre sur les paires d'entiers est défini par $(n, m) > (n', m')$ ssi

1. $n > n'$,
2. ou $n = n'$ et $m > m'$,

L'ordre sur les paires d'entiers est un ordre noethérien (il n'y a pas de suite infinie décroissante pour cet ordre).

Il suffit de vérifier que toute règle fait décroître la mesure de complexité du problème : si $P \rightarrow P'$ alors $\mu(P) > \mu(P')$.

\square

Le résultat de l'algorithme de Robinson sur le problème $s = t$ est soit \perp (si s, t ne sont pas unifiable, soit un système définissant un plus grand unificateur de s et t . Correction et complétude permettent de savoir que le système en forme résolu calculé donne un unificateur σ . Tout autre unificateur est par définition une instance de σ . Par contre l'unificateur le plus général n'est unique qu'à renommage près (un renommage ρ est une substitution qui effectue une permutation sur les variables de $Dom(\rho)$)

Proposition 6 *L'algorithme de Robinson peut être exponentiel dans le pire des cas.*

Prendre le problème : $f(x_1, \dots, x_n) = f(g(x_0, x_0), g(x_1, x_1), \dots, g(x_{n-1}, x_{n-1}))$
□

D'une certaine façon, l'exemple montre qu'on est nécessairement exponentiel avec la représentation classique d'arbre puisque l'écriture de la solution demande un temps exponentiel. En utilisant de bonnes structures de données avec du partage, on peut définir des algorithmes d'unification polynomiaux (même linéaires).

3.3 Partage de structure

3.3.1 Un algorithme d'unification plus efficace

La règle de remplacement est celle qui cause l'explosion de complexité dans l'algorithme de Robinson.

Elle est remplacée par la règle de fusion (Merge)

$$x = t, x = s, P \rightarrow x = t, t = s, P \text{ si } |t| < |s|, t \notin X$$

et de remplacement de variable

$$x_i = x_j, P \rightarrow x_i = x_j, P[x_j \leftarrow x_i] \text{ si } i < j \text{ et } x_j \in Var(P)$$

(on suppose que les variables sont x_1, \dots, x_n ce qui correspond à se donner un ordre strict sur ces variables))

et le test d'occurrence est remplacé par le test de cycle

$$x_1 = t_1, x_2 = t_2, \dots, x_n = t_n, P \rightarrow \perp \text{ si } \begin{array}{l} x_2 \in Var(t_1), \quad t_1, \dots, t_n \notin \mathcal{X} \\ x_3 \in Var(t_2), \\ \dots, \\ x_n \in Var(t_{n-1}), \\ x_1 \in Var(t_n) \end{array}$$

On appelle R' l'ensemble de règles correspondant.

Proposition 7 *L'application des règles R' termine.*

On remplace la mesure de complexité en remplaçant $(nbvarnr, nbsym)$ par $(\Sigma \#x_i * i, |P|)$ avec $\#x_i$ le nombre d'occurrence de x_i dans P et $|P|$ le nombre de symboles de P . Cette mesure décroît à chaque application de règle. Noter que si la condition de la règle merge n'est pas respectée on peut ne pas terminer.

La définition de forme résolue change et devient

Definition 10 *Un système est en forme résolue s'il ne contient pas de cycle et il est de la forme*

$$x_{i_1} = t_1, \dots, x_{i_m} = t_m \wedge \bigwedge_{j=1, \dots, m} \left(\bigwedge_{k \in I_j} x_{j_i} = x_k \right)$$

avec I_i l'ensemble des indices des variables x_k égales à x_{j_i} (avec $k > j_i$ et $j \neq 1, \dots, m$).

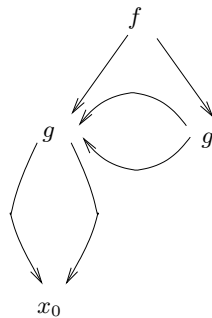
Proposition 8 *L'algorithme termine et calcule un système en forme résolue permettant d'obtenir un pgu.*

- La terminaison se fait comme avec l'algorithme de Robinson, car la règle du merge assure que la taille du système diminue.
- La préservation des solutions est évidente.
- Le pgu s'obtient en effectuant les substitutions que définit le système en forme résolue (ce qui peut amener à un calcul exponentiel).

□

3.3.2 DAG : directed acyclic graph

Un DAG est un graphe permettant de représenter un terme ou un ensemble de terme en partageant les structures communes.



Représentation en DAG de $f(g(x_0, x_0), g(g(x_0, x_0), g(x_0, x_0)))$

FIG. 3.2 – Le DAG pour $f(g(x_0, x_0), g(g(x_0, x_0), g(x_0, x_0)))$

Chaque noeud est étiqueté par un symbole de fonction ou une variable et les arcs issus d'un noeud relient le noeud aux noeuds représentant les fils. Le

graphe est un DAG s'il est acyclique. les noeuds n'ayant aucun prédecesseur ont les racines et chaque racine correspond à un terme représenté par le DAG. Chaque racine correspond à un terme représenté par le DAG. Chaque noeud du DAG correspond à un sous-terme des termes t_1, \dots, t_n représentés.

Un DAG permet de représenter un terme de taille exponentielle de manière compacte s'il y a beaucoup de redondance. Ainsi le terme défini par $t = f(t_1, \dots, t_n)$ avec $t_{i+1} = g(t_i, t_i)$ et $t_1 = g(a, a)$ qui est de taille exponentielle en n se représente par un DAG de taille linéaire en n . Réciproquement, afficher sous forme de terme un DAG peut demander un temps exponentiel.

Etant donné un DAG représentant des termes t_1, \dots, t_n tel que $\{x_1, \dots, x_n\} = Var(t_1) \cup \dots \cup Var(t_n)$, une substitution σ telle que $Dom(\sigma) \subseteq \{x_1, \dots, x_n\}$ et $sigma(x)$ est un sous-terme de t_1, \dots, t_n . L'application de σ aux termes du DAG donne un DAG qui se calcule en un temps linéaire en le nombre de variables.

Sur une représentation en DAG, le test de cycle se fait en temps linéaire et la forme résolue sous forme de DAG donne la substitution solution correspondant à un pgu. L'algorithme précédent est donc polynomial.

On peut définir aussi directement l'algorithme d'unification sur la structure de DAG, les opérations étant alors des manipulations de pointeurs.

Chapitre 4

Logique du premier ordre

4.1 Langage et Formules

Le langage est formé avec :

- un ensemble de connecteurs logiques,
- des quantificateurs existentiels \exists et universels \forall ,
- un ensemble dénombrable de variables X ,
- un ensemble fini de symboles de fonctions F chaque symbole ayant une arité fixe.

Les symboles de fonctions et les variables permettent de définir les termes comme vu précédemment. Pour simplifier on supposera que l'ensemble des symboles de fonctions contient une constante au moins.

L'ensemble \mathcal{Form} des formules est le plus petit ensemble tel que

- $r \in R$ d'arité n , $t_1, \dots, t_n \in T_F(X)$ alors $r(t_1, \dots, t_n) \in \mathcal{Form}$ (atomes),
- $\varphi, \psi \in \mathcal{Form}$ alors $\neg\varphi, \varphi \wedge \psi, \varphi \vee \psi, \varphi \Rightarrow \psi, \varphi \Leftrightarrow \psi \in \mathcal{Form}$
- $\varphi \in \mathcal{Form}$ alors $\forall x\varphi, \exists x\varphi \in \mathcal{Form}$.

Exemple $F = \{o, s(-), +(-, -)\}$, $R = \{\geq\}$ avec $+, \geq$ notées de manières infixées.

$$\forall x s(x) \geq o \wedge \forall x, y s(x) \geq s(y) \Rightarrow x \geq y$$

est une formule

Mais $s(s(x))$ n'en n'est pas une (c'est un terme), ni $\forall x s(x)$.

□

4.1.1 Variables libres, liées

On définit la notion d'occurrence libre, liée et de $FV(\varphi)$ l'ensemble des variables libres d'une formule ainsi :

- si φ est un atome alors toute occurrence d'une variable x dans φ est libre.

- si φ est $\exists x \psi$ ou $\forall x \psi$ alors $FV(\varphi) = FV(\psi) - \{x\}$ et toute occurrence libre de x dans ψ devient liée dans φ par le quantificateur introduit.
 - $FV(\varphi \circ \varphi') = FV(\varphi) \cup FV(\varphi')$ et l'ensemble des occurrence libre d'une variable est l'union des ensembles des occurrences libre dans φ et dans φ' .
- Une formule φ est close ou fermée ssi $FV(\varphi) = \emptyset$.

Exemples

$$[\forall x(\exists y p(x, y))] \Rightarrow [\exists z \forall y y q(x, z, y)]$$

a une variable libre x , des variables liées x, y, z . La première occurrence de x est liée par le quantificateur $\forall x$. La première occurrence de y est liée par $\exists y$ et la seconde par $\forall y$ \square .

4.2 Sémantique

4.2.1 Evaluation d'une formule

Il faut donner un sens vrai ou faux aux formules, les interpréter. Pour cela il faut donner un sens aux fonctions, puis aux relations. Pour cela il faut dire dans quel domaine les individus se trouvent. Cela se fait en se donnant une interprétation.

Etape 1 : choisir un domaine D qui est un ensemble non vide.

exemple :

D est l'ensemble des étudiants de licence D_E

D est l'ensemble des entiers positifs ou nuls \mathcal{N}

D est l'ensemble des triangles isocèles Tri

Etape 2 : donner un sens aux fonctions. A chaque symbole $f \in F$ d'arité n on associe une fonction $I(f)$ notée aussi f^I , de $D^n \rightarrow D$ qui est son interprétation. En particulier une constante sera interprété par un élément du domaine.

exemple :

– La constante c s'interprète par Pierre Dupont (pour un choix de domaine $D = D_E$)

– La fonction f d'arité 2 s'interprète par l'addition (pour un choix de domaine $D = \mathcal{N}$)

– La fonction s s'interprète comme le symétrique par rapport à l'origine (pour un choix de domaine $D = Tri$)

Noter qu'on interprète par des fonctions de $D^n \rightarrow D$ et pas par des fonctions d'un autre domaine : par exemple si $D = Tri$ on ne peut interpréter la fonction s comme l'aire du triangle (fonction de Tri dans \mathcal{R}).

Etape 3 : donner un sens aux relations. A chaque symbole $r \in R$ d'arité n on associe une relation $I(r)$ (noté aussi r^I) de D^n qui est son interprétation.

exemple :

- La relation binaire r peut s'interpréter par *suivre le même cursus*, les cursus étant math-info, physique-info ou informatique (pour un choix de domaine $D = D_E$).
- La relation binaire bg s'interprète \geq (pour un choix de domaine $D = \mathcal{N}$).
- La fonction *isoc* s'interprète *être isocèle* (pour un choix de domaine $D = \text{Tri}$)

Une structure $\langle D, I \rangle$ est la donnée d'un domaine D et d'une interprétation I qui interprète chaque symbole de fonction et de relation.

Etape 4 : donner une valeur 1 pour vrai ou 0 pour faux à une formule dans une structure $\langle D, I \rangle$.

On sait le faire intuitivement, mais la définition formelle demande à être plus précis.

La première difficulté est l'existence de variables libres auxquelles il faut donner une valeur : la formule $\forall x x \geq y$ avec $D = \mathcal{N}$, \geq interprété comme plus grand, peut être vraie ou fausse selon la valeur de y .

1. Assignment. Une assignation est une application A de X dans D qu'on étend en une application de $T_F(X)$ dans D par :
 - $A(c) = I(c)$
 - $A(f(t_1, \dots, t_n)) = f^I(A(t_1), \dots, A(t_n))$
 Une x -variante B d'une assignation A est une assignation telle que $B(y) = A(y)$ pour toute $y \neq x \in \mathcal{X}$ (égale à A sur \mathcal{X} sauf peut-être en x).
2. Evaluer une formule φ étant donné une structure $\langle D, I \rangle$ et une assignation A c'est calculer la valeur de vérité $\varphi^{I,A}$ par :
 - $p(t_1, \dots, t_n)^{I,A} = 1$ ssi $p^I(t_1^{I,A}, \dots, t_n^{I,A})$
 - $(\varphi \vee \psi)^{I,A} = \max(\varphi^{I,A}, \psi^{I,A})$ (et similaire pour $\wedge, \Rightarrow, \Leftrightarrow$)
 - $\exists x \varphi^{I,A} = 1$ ssi il existe une x -variante B de A telle que $\varphi^{I,B} = 1$.
 - $\forall x \varphi^{I,A} = 1$ ssi $\varphi^{I,B} = 1$ pour toute x -variante B de A .

Exemples :

- $\forall x, y, z (r(x, y) \wedge r(y, z) \Rightarrow r(x, z))$ dans le domaine D des étudiants du CMI avec $r^I = \text{suivre le même cursus}$ est vraie.
- Avec le même domaine et la même interprétation, $\exists y \forall x (r(x, y))$ est fausse : il n'y a pas un étudiant qui suit le même cursus que tous les autres (il n'est pas possible de suivre deux cursus différents et il y en a plusieurs au CMI).
- $\forall x x > y$ est vraie dans la structure $\langle \mathcal{N}, I \rangle$ avec $I(o)=0$, $I(s)$ la fonction successeur, $I(>)$ la relation $>$ avec l'assignation $A(x) = 0$, fausse pour toute

4.2.2 Terminologie

Par définition, si φ est une formule close alors la valeur de vérité de $\varphi^{I,A}$ est indépendante de A et on la note φ^I . On dit que la formule est satisfaisable si cette valeur est 1 (i.e. vrai). Dans toute la suite on ne s'intéressera qu'à des

formules closes. Quand la formule n'est pas close, on considère usuellement sa clôture universelle c'est à dire $\forall x_1, \dots, x_n \varphi$ si $FV(\varphi) = \{x_1, \dots, x_n\}$.

Si φ est une formule close et $\langle D, I \rangle$ une structure telle que φ est satisfaisable dans $\langle D, I \rangle$, alors on dit que $\langle D, I \rangle$ est un *modèle* de φ . Un modèle d'un ensemble de formules closes Γ est une interprétation qui est modèle de chaque formule de Γ .

Une formule close qui n'a pas de modèle est dite insatisfaisable.

Exemple : $\forall x (p(x) \wedge \neg p(x))$

Une formule close φ qui est satisfaisable dans n'importe quelle structure est une tautologie, écrit $\models \varphi$.

Exemple : $(\forall x (p(x) \wedge q(x))) \Leftrightarrow (\forall x p(x) \wedge \forall x q(x))$

Une formule close φ est conséquence logique d'un ensemble de formules closes si φ est vrai dans tout modèle de Γ , écrit $\Gamma \models \varphi$.

Théorie : une théorie est l'ensemble des conséquences logiques d'un ensemble de formules closes. Par exemple la théorie des groupes est l'ensemble des formules logiques qui sont vraies dans tous les groupes.

4.2.3 Modèle de Herbrand

Une interprétation de Herbrand est une interprétation dont le domaine D est l'ensemble des termes et un symbole de fonction est interprété par lui même, cad f^I est la fonction $f^I : t_1, \dots, t_n \rightarrow f(t_1, \dots, t_n)$. La seule latitude qui est donnée est dans l'interprétation des prédicats (qu'on identifie à un sous-ensemble de T_F^n pour un prédicat d'arité n)

Intérêt des modèles de Herbrand : il n'y a pas à chercher le domaine ni les interprétations de fonctions.

Limite : certaines formules n'ont pas de modèles de Herbrand (dans la signature donnée).

Résultat fondamental (th. de Herbrand) : une formule universelle (i.e. de la forme $\forall x_1 \dots x_n \varphi$ où φ ne contient pas de quantification) a un modèle si et seulement si elle a un modèle de Herbrand. Ce résultat sera montré plus loin.

De plus le paragraphe suivant montre qu'on peut transformer une formule en une formule universelle en préservant la satisfaisabilité (mais pas l'équivalence logique).

4.3 Modélisation

La logique du premier ordre est très expressive et permet de modéliser de nombreux problèmes. Pour correspondre totalement à ce qui est utile en pratique, il manque de pouvoir parler d'ensemble et d'un prédicat fondamental l'égalité qui a des propriétés particulières. L'axiomatisation de la théorie des ensembles n'est pas le sujet ici et celle de l'égalité ne sera qu'évoquée.

Une théorie est l'ensemble des conséquences logiques d'un ensemble de formules closes (les axiomes de la théorie).

Théorie des groupes abélien le prédicat sera = (noté de manière infixée) et les symboles fonctionnels sont + d'arité 2, *op* d'arité 1, et 0 d'arité 0. Les axiomes de la théorie des groupes abéliens sont

$$\begin{aligned} \forall x, y \quad x + y &= y + x \\ \forall x, y, z \quad x + (y + z) &= (x + y) + z \\ \forall x \quad x + 0 &= x \\ \forall x \quad x + op(x) &= 0 \end{aligned}$$

avec en plus les axiomes définissant = comme l'égalité (voir la partie sur l'égalité). Une formule qui n'est pas un axiome vraie dans la théorie des groupes abélien est $\forall x \exists y \quad x + y = 0$.

4.4 Manipulation de formules

La relation définie par $\models \varphi \Leftrightarrow \psi$ avec φ, ψ des formules closes est une relation d'équivalence. On dit que φ et ψ sont logiquement équivalentes.

On va donner des règles de calcul qui mettent les formules sous une forme plus simple en préservant l'équivalence logique.

4.4.1 Mise en forme préfixe

Le renommage permet de se débarrasser des variables à la fois liées et libres.

Proposition 9 *Pour toute formule il existe une formule équivalence telle que une variable ne peut avoir des occurrences libres et liées et toutes les occurrences liées d'une variable le sont par le même quantificateur.*

On suppose maintenant que les formules ont été renommées. La mise en forme préfixe revient à mettre toutes les quantification en tête.

Une formule est en forme préfixe si elle s'écrit $(\forall|\exists)^* x_1 \dots x_n \varphi$ avec φ une formule sans quantificateur.

$\frac{\exists x \varphi \vee \psi}{\exists x(\varphi \vee \psi)}$	$\frac{\forall x \varphi \vee \psi}{\forall x(\varphi \vee \psi)}$	$\frac{\exists x \varphi \wedge \psi}{\exists x(\varphi \wedge \psi)}$	$\frac{\forall x \varphi \wedge \psi}{\forall x(\varphi \wedge \psi)}$
$\frac{\varphi \vee \exists x \psi}{\exists x(\varphi \vee \psi)}$	$\frac{\varphi \vee \forall x \psi}{\forall x(\varphi \vee \psi)}$	$\frac{\varphi \wedge \exists x \psi}{\exists x(\varphi \wedge \psi)}$	$\frac{\varphi \wedge \forall x \psi}{\forall x(\varphi \wedge \psi)}$
$\frac{(\forall x \varphi) \Rightarrow \psi}{\exists x(\varphi \Rightarrow \psi)}$	$\frac{(\exists x \varphi) \Rightarrow \psi}{\forall x(\varphi \Rightarrow \psi)}$	$\frac{\varphi \Rightarrow (\forall x \psi)}{\forall x(\varphi \Rightarrow \psi)}$	$\frac{\varphi \Rightarrow (\exists x \psi)}{\exists x(\varphi \Rightarrow \psi)}$
$\frac{\neg(\exists x \varphi)}{\forall x \neg \varphi}$	$\frac{\neg(\forall x \varphi)}{\exists x \neg \varphi}$	$\frac{\varphi \Leftrightarrow \psi}{(\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi)}$	

Les règles préservent l'équivalence logique et terminent.

Proposition 10 *Pour toute formule φ il existe une formule équivalente en forme préfixe.*

4.4.2 Skolémisation

On veut obtenir une formule sous forme universelle à partir d'une forme prenex.

Idée : remplacer $\forall x \exists y \varphi$ par $\forall x \varphi_{y \leftarrow f(x)}$ car le y qui fait que la formule est vraie dépend, est une fonction, de x . Cette opération change l'ensemble des symboles de fonctions, ce qui a une conséquence importante sur la sémantique (il y aura un symbole de plus à interpréter).

Règle de Skolémisation.

$$\frac{\forall x_1, \dots, x_n \exists y \varphi}{\forall x_1, \dots, x_n \varphi_{y \leftarrow f(x_1, \dots, x_n, X_1, \dots, X_m)}}$$

avec

- f un nouveau symbole (dit de Skolem)
- X_1, \dots, X_m les variables libres de $\forall x_1, \dots, x_n \exists y \varphi$

Une formule universelle obtenue par application itérée de la règle précédente est appelée forme de skolem ou skolémisée de φ .

Exemple :

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

qui donne après mise en forme prenex

$$\forall x \exists y \forall x' \exists y' (p(y, x) \Rightarrow p(y', x'))$$

qui donne

$$\forall x \forall x' (p(x, g(x)) \Rightarrow p(h(x, x'), x'))$$

On peut être plus astucieux en prenant en compte les portées des quantificateurs et obtenir

$$\forall y (p(a, y) \Rightarrow \forall x p(f(x), x))$$

Proposition 11 *Si φ_s est obtenue par skolémisation à partir de φ alors φ^s est satisfaisable si et seulement si φ est satisfaisable.*

Il suffit de le montrer pour la règle de transformation. La différence entre les deux formules est que l'une est sur une signature comprenant une symbole de fonction de plus.

- φ^s satisfaisable entraîne φ satisfaisable.
Si $S = \langle D, I \rangle$ et A une assignation qui rend vraie φ^s alors l'assignation qui évalue y en $f^I(B(x_1), \dots, B(x_n), B(X_1), \dots, B(X_m))$ rend vraie $\forall x \exists y \varphi$.
- φ satisfaisable entraîne φ^s satisfaisable.
Soit $S = \langle D, I \rangle$ et A une assignation qui rend vraie φ . Soit B une $x_1, \dots, x_n, X_1, \dots, X_m$ -variante de A .
Pour chaque valeur de $B(x_1), \dots, B(x_n), B(X_1), \dots, B(X_m)$ il existe une valeur $B(y)$ telle que $\varphi^I(B(x_1), \dots, x_n, B(y), B(X_1), \dots, B(X_m))$.
Soit I' obtenue en étendant I en prenant pour $I(f)$ une fonction qui vaut $B(y)$ en $B(x_1), \dots, x_n, B(y), B(X_1), \dots, B(X_m)$.
Alors D, I' est un modèle de φ_s .

□

L'équivalence n'est pas préservée : $\forall x \exists y (x \geq y)$ et $\forall x x \geq f(x)$ ne sont pas équivalentes. On prend $D = \mathcal{N}$ et \geq^I interprété par \geq . Si on interprète f par $f^I : x \rightarrow x + 1$ la première formule est vraie et la deuxième est fausse. Le problème de non-équivalence est l'introduction d'un symbole qu'on peut interpréter comme on veut : soit on choisit la "bonne" fonction qui donne les valeurs qu'il faut pour valider la formule, soit on prend une fonction arbitraire qui peut falsifier la nouvelle formule.

Remarques essentielles.

- Une formule skolemisée φ^s et la formule initiale φ **ne sont pas équivalentes** en général.
- Une formule peut avoir plusieurs formes prenexes et plusieurs formes de skolem.
- La méthode présentée est inefficace et doit être raffinée pour éviter d'introduire trop de symboles de Skolem et pour limiter leurs arités.

4.4.3 Mise en forme clauseale

Une formule peut être transformée en une conjonction de clauses (en préservant la satisfiabilité mais pas l'équivalence) :

- Mettre en forme préfixe.
- Skolemiser.
- Utiliser la mise en forme clauseale propositionnelle.

Exemple $\forall x, y, z [(\forall x r(x, x)) \wedge ((\forall x (r(x, y)) \Rightarrow (\exists y (r(y, x) \wedge \forall y r(y, z)))))]$
se met en forme clauseale :

- renommage.

$$\forall x, y, z, [(\forall x' r(x', x')) \wedge ((\forall x'' (r(x'', y)) \Rightarrow (\exists y' (r(y', x) \wedge \forall y'' r(y'', z)))))]$$

- Mise en forme préfixe :

$$\forall x, y, z, x' [r(x', x') \wedge ((\forall x'' (r(x'', y)) \Rightarrow (\exists y' (r(y', x) \wedge \forall y'' r(y'', z)))))]$$

$$\forall x, y, z, x' \exists y' \forall y'' [r(x', x') \wedge ((\forall x'' (r(x'', y)) \Rightarrow ((r(y', x) \wedge r(y'', z)))))]$$

$$\forall x, y, z, x' \exists y' \forall y'' \exists x'' [r(x', x') \wedge ((r(x'', y)) \Rightarrow ((r(y', x) \wedge r(y'', z))))]$$

- Skolemisation :

$$\forall x, y, z, x', y'' \exists x'' [r(x', x') \wedge ((r(x'', y)) \Rightarrow ((r(g(x, y, z, x'), x) \wedge r(y'', z))))]$$

$$\forall x, y, z, x', y'' [r(x', x') \wedge ((r(f(x, y, z, x', y'')) \Rightarrow ((r(g(x, y, z, x'), x) \wedge r(y'', z))))]$$

– Forme clausale :

$$\forall x, y, z, x', y'' [r(x', x') \wedge (\neg r(f(x, y, z, x', y'')) \vee (r(g(x, y, z, x'), x)) \wedge (\neg r(f(x, y, z, x', y'')) \vee r(y'', z)))]$$

d’où l’ensemble de clauses

$$\{r(x', x'), \neg r(f(x, y, z, x', y'')) \vee (r(g(x, y, z, x'), x)), \neg r(f(x, y, z, x', y'')) \vee r(y'', z)\}$$

4.5 La méthode de résolution

4.5.1 Principe

C’est un calcul qui permet de montrer qu’une formule φ se déduit dans un système formel qui

- est réfutationnel : on part de la formule $\neg\varphi$
- met la formule en forme clausale qui donne un ensemble de clauses
- sature l’ensemble de clauses obtenues avec les règles de

1. Résolution

$$\frac{l \vee C \quad \neg l' \vee C'}{\sigma(C \vee C')}$$

avec $l \vee C$ et $\neg l' \vee C'$ deux clause ne partageant pas de variables, l, l' deux littéraux unifiables, σ le pgu de l et l' . La clause $\sigma(C \vee C')$ est appelée la résolvente.

($\sigma(C)$ désigne la clause obtenue à partir de C en appliquant σ à chaque élément de C)

2. Factorisation

$$\frac{l \vee C_1 \vee l' \vee C_2}{\sigma(C_1) \vee \sigma(l) \vee \sigma(C_2)}$$

avec l, l' deux littéraux unifiables, σ le pgu de l et l' .

Un ensemble de clause clos par résolution et factorisation est dit *saturé*.

4.5.2 Exemple

Les variables des clauses sont universellement quantifiées et donc une occurrence de x dans une clause est indépendante d’une occurrence de x dans une autre clause. Avant de faire une unification, il faudrait faire un renommage pour éviter des conflits de noms qui feraient conclure à une non-unifiabilité de littéraux -en particulier avec le test d’occurrence- alors que les littéraux renommés sont unifiables (dans les exemples pour alléger l’écriture on ne renomme pas systématiquement les clauses lorsque ce n’est pas utile).

Exemple 1.

$$(\forall x(p(x) \vee q(x))) \Rightarrow (\exists x p(x) \vee \forall x q(x))$$

Mise en forme clausale de la négation :

$$\mathcal{C} = \{p(x) \vee q(x), \neg p(x'), \neg q(a)\}$$

Résolution :

$$\frac{p(x) \vee q(x) \quad \neg q(a)}{p(a)}$$

car le pgu du problème d'unification $q(x) = q(a)$ est $\{x \leftarrow a\}$

$$\frac{p(a) \quad \neg p(x)}{\square}$$

car le pgu du problème d'unification $p(x) = p(a)$ est $\{x \leftarrow a\}$

Exemple 2.

$$(\psi_1 \wedge \psi_2 \wedge \psi_3) \Rightarrow \psi_4$$

avec

$$\begin{aligned} \psi_1 &: \forall x \neg r(x, x) \\ \psi_2 &: \forall x', y', z' r(x', y') \wedge r(y', z') \Rightarrow r(x', z') \\ \psi_3 &: \forall x'', y'' r(x'', g(y'')) \Rightarrow r(g(y''), x'') \\ \psi_4 &: \forall u, v \neg r(u, g(v)) \end{aligned}$$

La forme clausale de la négation est :

$$\mathcal{C} = \{r(x, x), \neg r(x', y') \vee \neg r(y', z') \vee r(x', z'), \neg r(x'', g(y'')) \vee r(g(y''), x''), r(a, g(b))\}$$

Résolution

$$\frac{r(a, g(b)) \quad \neg r(x'', g(y'')) \vee r(g(y''), x'')}{r(g(b), a)}$$

car le pgu de $r(a, g(b)) = r(x'', g(y''))$ est $\{x'' \leftarrow a, y'' \leftarrow b\}$

$$\frac{r(a, g(b)) \quad \neg r(x', y') \vee \neg r(y', z') \vee r(x', z')}{\neg r(g(b), z') \vee r(a, z')}$$

car le pgu de $r(a, g(b)) = r(y', z')$ est $\{y' \leftarrow a, z' \leftarrow b\}$

$$\frac{r(g(b), a) \quad \neg r(g(b), z') \vee r(a, z')}{r(a, a)}$$

car le pgu de $r(g(b), a) = r(g(b), z')$ est $\{z' \leftarrow a\}$

$$\frac{r(a, a) \quad \neg r(x, x)}{\square}$$

car le pgu de $r(a, a) = r(x, x)$ est $\{x \leftarrow a\}$

4.5.3 Traitement de la Conséquence logique

En général on utilise la résolution dans le contexte suivant :

- Γ est un ensemble de formules closes (les hypothèses)
- φ est une formule close qui est une conséquence logique de Γ .

Montrer par résolution que $\Gamma \vdash \varphi$ par résolution consiste à montrer que $\Gamma \cup \{\neg\varphi\}$ permet de dériver la clause vide (après mise en forme clausale de $\neg\varphi$ et des formules de Γ).

4.6 Correction et complétude

Theorème 5 *Si $\vdash \varphi$ alors $\models \varphi$.*

Il suffit de réaliser qu'une résolvente (ou une clause obtenue par factorisation) est conséquence logique des deux clauses dont elle provient. \square

Une clause est obtenue par résolution à partir de \mathcal{C} soit elle est dans \mathcal{C} , soit il existe une suite de résolution qui termine avec la clause C . Dans ce qui suit, une instance close d'une clause C est une clause $C\theta$ avec θ une substitution qui instancie chaque variable de la clause par un terme clos, i.e. $Var(C) = \{x_1, \dots, x_n\}$, $\theta = \{x_1 \leftarrow t_1, \dots, x_n \leftarrow t_n\}$ avec $Var(t_i) = \emptyset$ pour $i = 1, \dots, n$. Par exemple, avec $F = \{o, s(-)\}$, $R = \{p(-)\}$ si $C = \neg p(x) \vee p(s(s(x)))$ une instance close est $D = \neg p(s(o)) \vee p(s(s(s(o))))$ (avec $\theta = \{x \leftarrow s(o)\}$).

La complétude repose sur le lemme de relèvement, qui permet de passer d'une résolution sur les instances closes à une résolution sur les clauses avec variables.

Proposition 12 (Lemme de relèvement) *Soit \mathcal{C} un ensemble de clauses et soit \mathcal{D} l'ensemble des clauses closes obtenu à partir de \mathcal{C} . Si D est obtenue par résolution à partir de \mathcal{D} , il existe C obtenue par résolution à partir de \mathcal{C} et une substitution θ telle que $D = C\theta$.*

On suppose que pour toute clause D obtenue par résolution de longueur $< n$ dans \mathcal{D} , il existe θ telle que $D = C\theta$ avec C obtenue par une résolution de longueur $< n$ dans \mathcal{C} .

- Cas de base $n = 0$: cela signifie que $D \in \mathcal{D}$ et donc $D = C\theta$ avec $C \in \mathcal{C}$.
- Etape d'induction.

Cas 1 : la règle appliquée est la résolution.

Soit $D_1 = L \vee D'_1$, $D_2 = \neg L \vee D'_2$ et la résolvente $D'_1 \vee D'_2$.

Par hypothèse d'induction $D_1 = \theta(l) \vee C_1\theta$ et $D_2 = \neg\theta(l') \vee C_2\theta$ avec $C_1 = l \vee C'_1$, $C_2 = l' \vee C'_2$, $L = \theta(l)$, $\neg L = \neg\theta(l')$. Donc l et l' sont unifiables avec un pgu σ , et par définition du pgu on a $\theta = \rho\sigma$. Par conséquent il existe une résolution

$$\frac{l \vee C_1 \quad \neg l' \vee C_2}{\sigma(C_1) \vee \sigma(C_2)}$$

avec $\sigma(l) = \sigma(l')$ et $\rho(\sigma(C_1) \vee \sigma(C_2)) = D'_1 \vee D'_2$ car $\rho\sigma = \theta$.

Cas 2 : la règle appliquée est la factorisation. Raisonement similaire.

□

La preuve montre qu'on peut relever toute la suite de résolution.

Proposition 13 $\mathcal{C} \vdash \square$ si et seulement si $\mathcal{D} \vdash \square$ avec $\mathcal{D} = \{C\theta \mid C \in \mathcal{C}, \theta \text{ close}\}$

\Rightarrow : évident en instantiant une dérivation de \square à partir de \mathcal{C} .

\Leftarrow : par le lemme de relèvement. □

Theorème 6 Si $\models \varphi$ alors $\vdash \varphi$.

Soit \mathcal{D}' l'ensemble obtenu par saturation de \mathcal{D} l'ensemble des instances closes de \mathcal{C} .

Proposition 14 \mathcal{D}' a un modèle ssi \mathcal{D}' ne contient pas \square .

L'ensemble des atomes clos peut s'énumérer comme p_1, p_2, \dots . On construit un arbre de valuation où les noeuds sont étiquetés par un ensemble de clauses défini ainsi :

- La racine est étiqueté par \mathcal{D}' ,
- Un noeud N étiqueté par \mathcal{D}_N a deux fils N_0 correspondant à $p_i = 0$ et le fils N_1 correspondant à $p_i = 1$,
- \mathcal{D}_{N_0} est l'ensemble des clauses de \mathcal{D}_N ne contenant pas $\neg p_i$ dans lesquelles p_i est remplacé par 0,
- \mathcal{D}_{N_1} est l'ensemble des clauses de \mathcal{D}_N ne contenant pas p_i dans lesquelles p_i est remplacé par 1
- un noeud étiqueté par un ensemble de clauses contenant \square n'est pas développé.

Par construction si \mathcal{D}_N est saturé et ne contient pas \square alors \mathcal{D}_{N_0} ou \mathcal{D}_{N_1} est saturé et ne contient pas \square .

Soit l'arbre contient une branche infinie, et il est aisé de voir que la valuation correspondante est un modèle de chaque clause de \mathcal{D} .

Soit toutes les branches sont finies (et terminent donc par un noeud d'échec) et l'arbre est fini (lemme de Koenig) et pour chaque feuille il existe une clause de \mathcal{D} évaluée à 0. L'ensemble de ces formules est un ensemble fini de formules qui est insatisfaisable. Chaque atome correspond à un symbole propositionnel et la complétude de la résolution en calcul propositionnel assure qu'il existe une preuve propositionnelle par résolution dérivant \square . Cette preuve donne une preuve par résolution de \square (en remplaçant un symbole propositionnel par l'atome correspondant) ce qui contredit que \mathcal{D}' ne contient pas \square et est saturé. □

On termine la preuve de complétude par contraposée : $\not\vdash \varphi$ implique $\not\models \varphi$.

$\not\vdash \varphi$ signifie que \mathcal{C} l'ensemble des clauses de la forme clausale de $\neg\varphi$ ne permet pas de déduire \square . Donc l'ensemble \mathcal{D} des instances closes de \mathcal{C} ne permet pas de déduire \square donc a un modèle. Mais ce modèle est un modèle de Herbrand de \mathcal{C} et donc de $\neg\varphi$.

4.7 Compléments

4.7.1 Egalité

L'égalité est un concept fondamental et on veut donc introduire un symbole $=$ dans la logique qui s'interprétera toujours par l'égalité du domaine d'interprétation. L'égalité est une relation d'équivalence telle que le remplacement d'égaux par des égaux ne change pas la valeur des fonctions et des relations. On montre qu'il suffit de rajouter les axiomes suivants pour que $=$ s'interprète par l'égalité pour tout domaine D

- $\forall x (x = x)$
- $\forall x_1, \dots, x_n, y_1, \dots, y_n \bigwedge_{i=1}^{i=n} x_i = y_i \Rightarrow f(x_1, \dots, x_n) = f(y_1, \dots, y_n)$
- $\forall x_1, \dots, x_n, y_1, \dots, y_n (\bigwedge_{i=1}^{i=n} x_i = y_i \Rightarrow (p(x_1, \dots, x_n) \Rightarrow p(y_1, \dots, y_n)))$

L'utilisation de ce prédicat permet d'enrichir l'expressivité de la logique :

- Le domaine d'interprétation D a au moins deux éléments : $\exists x, y \ x \neq y$
- Le domaine d'interprétation D n'a qu'un seul élément : $\forall x, y \ x = y$

La logique du premier ordre à laquelle on a rajouté cette axiomatisation est appelée la logique du premier ordre avec égalité. Elle a les mêmes propriétés fondamentale que la logique sans égalité mais cela demande un peu de travail pour l'établir.

4.7.2 Théorème de Herbrand

L'examen de la preuve de complétude permet de donner le théorème de Herbrand. Un ensemble de formules universelles est un ensemble de formules en forme préfixe tel que le seul quantificateur est le quantificateur universel.

Theorème 7 *Un ensemble de formules universelles a un modèle si et seulement si il a un modèle de Herbrand.*

La mise en forme clausale des formules n'utilise pas la skolemisation donc le l'ensemble de clause obtenu a les mêmes modèles que l'ensemble de départ. L'ensemble de clause saturé obtenu par la méthode de résolution a un modèle si et seulement l'ensemble de départ a un modèle (la résolution n'ajoute que des conséquences logiques). Si l'ensemble ne contient pas la clause vide alors il donne un modèle de Herbrand de l'ensemble initial, s'il contient la clause vide alors c'est qu'il n'a pas de modèle.

4.7.3 Théorème de compacité

Le théorème de compacité reste vrai au premier ordre :

Theorème 8 *Un ensemble de formules a un modèle si et seulement tout sous-ensemble fini a un modèle.*

Application. Il n'existe pas d'axiomatisation finie au premier ordre de la notion *être fini*.

Une telle axiomatisation serait un ensemble fini de formules closes du premier ordre dont les modèles sont exactement les ensemble finis.

On suppose qu'une telle axiomatisation A existe et on considère les formules

$$\varphi_n \equiv \bigwedge_{i \neq j, 1 \leq i, j \leq n} x_i \neq x_j$$

et Eq une axiomatisation de l'égalité.

On considère $F = A \cup Eq \cup \{\varphi_n \mid n \neq 2\}$

Tout sous-ensemble fini de F a un modèle (un ensemble fini d'au moins $p+1$ élément avec p le plus grand indice tel que φ_p est dans l'ensemble).

Mais F ne peut avoir un modèle fini.

Théorème 9 (Lowenheim-Skolem) *Si un ensemble dénombrable de formules closes Γ a un modèle alors il a un modèle dénombrable.*

Une conséquence est que l'ensemble des réels n'est pas axiomatisable au premier ordre.

Résumé

- Logique du premier ordre : variables, fonctions relation et quantification en plus du calcul propositionnel.
- Modèle : un choix de domaine, de fonctions et de relation sur ce domaine qui permet de rendre vraie une formule (close)
- Modèle de Herbrand : un modèle syntaxique *qui n'a pas de sens*.
- Preuve par résolution :
 - Mise en forme prenexe de la négation, skolémisation, puis forme clausale.
 - Dériver la clause vide à partir de l'ensemble de clause :
 - on trouve un preuve aors la formule est prouvable et c'est une tautologie (th. de correction)
 - on n'y arrive pas alors on ne sait rien dire en général! sauf si un raisonnement permet de dire que l'ensemble est saturé (difficile!)
 - si la formule est une tautologie alors on sait qu'il existe une preuve par résolution (th de complétude).

Chapitre 5

Résolution et Programmation logique

Avertissement : Ce cours n'est pas rédigé complètement

5.1 La programmation logique

5.1.1 Clause de Horn

Definition 11 Une clause de Horn est une disjonction de littéraux dont au plus un seul est positif.

On les classera en

- Fait : p
- Buts : $\neg p_1 \vee \dots \vee \neg p_n$
- Règles : $p_1 \wedge \dots \wedge p_n \Rightarrow p$ (notée $p : \neg p_1, \dots, p_n$)

Definition 12 Un programme Prolog \mathcal{P} est formé de faits et de règles.

Une requête est une clause qui est un but g .

Dans l'écriture des programmes PROLOG, les buts sont écrits en omettant la négation : on écrit p . au lieu de $\neg p$ et p_1, \dots, p_n au lieu de $\neg p_1 \vee \dots \vee \neg p_n$

Si $\mathcal{P} \cup \{g\}$ est insatisfaisable alors la résolution avec une stratégie bien choisie permet de déduire la clause vide. L'interpréteur Prolog utilise la SLD résolution qui est la résolution avec une règle de sélection particulière des clauses à résoudre. Cette stratégie est complète lorsque la règle de sélection est équitable mais l'implémentation usuelle dans les interpréteurs Prolog n'est pas complète.

5.1.2 Interpréteur Prolog

La résolution est cachée dans PROLOG par la notation et la stratégie utilisée, ce qui permet de donner un interpréteur PROLOG sans faire référence à la règle de résolution.

L'interpréteur prolog est donné par la fonction *solve*. Les buts mentionnés ici sont des atomes, car le calcul par résolution est ici implicite. Le calcul de *solve*(LG,P) correspond à la résolution sur $C \cup \{\neg g_1 \vee \dots \vee \neg g_n\}$.

```

solve(LG,P)
/* LG=liste de buts g1,...,gn (ou vide)
   P programme=suite de clauses (faits ou règles) C1,...,Cn
   solve resoud la liste de but LG avec le programme P
*/
si LG est vide alors retourner Oui
sinon
    g=premier but de LG
    PP=P
    Tant que PP n'est pas vide faire
        C=premiere clause de PP
        PP=PP \ C
        si identique(tete(C),g) alors solve(LG {g<-corps(C)},P) fsi
    fait
fsi

```

Déroulement de l'interpréteur L'arbre de résolution est l'arbre

- dont la racine est étiquetée par le but initial,
- si un noeud est étiqueté par une liste de but g_1, \dots, g_n , alors les fils de ce noeud sont les noeuds étiquetés par $B_1, \dots, B_m, g_2, \dots, g_n$ avec $A : -B_1, \dots, B_m$ une règle ou un fait ($m=0$ dans ce cas) tel que $A = g_1$. L'ordre des noeuds est l'ordre des clauses dans le programme.

Exemple :

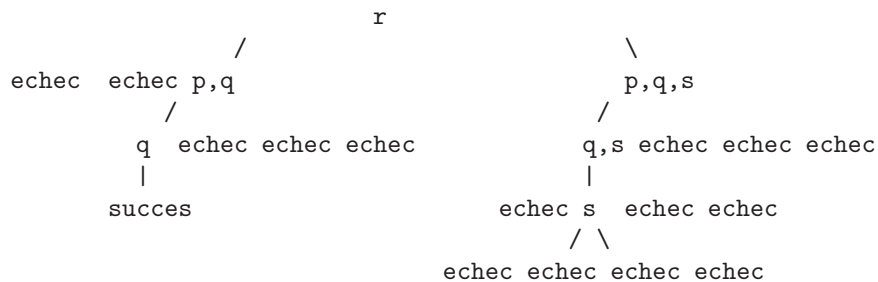
Le programme est

```

p.
q.
r : -p, q.
r : -p, q, s.

```

alors pour le but r l'arbre est



5.2 Sémantique par Point Fixe

Clauses de Horn et valuation.

Un programme prolog P propositionnel est formé de *règles*

$$p_1 \wedge \dots \wedge p_n \Rightarrow p$$

($n \geq 1$) et de *faits* :

$$p$$

Un modèle v est déterminé par l'ensemble \mathcal{M}_v des symboles propositionnels qui valent 1 et on peut identifier v avec cet ensemble. Cela permet de dire qu'un modèle v est plus petit que v' si $\mathcal{M}_v \subseteq \mathcal{M}_{v'}$.

Exemple : si P est le programme

$$\{p, p \Rightarrow q, p \wedge q \Rightarrow r, r \Rightarrow q, q \wedge u \Rightarrow rq \wedge r \Rightarrow s\}$$

sur $PROP = \{p, q, r, s, t, u, v\}$ il a un modèle $\{p, q, r, s\}$ inclus dans tout autre modèle.

Si on considère des formules quelconques, les modèles n'ont pas de relation particulières entre eux. Par exemple $(p \vee \neg q)$ a comme modèle $\{p\}, \{q\}, \{p, q\}$. Pour les programmes Prolog on a un résultat très fort : l'existence d'un plus petit modèle. Noter que ce résultat est faux pour l'ensemble des clauses de Horn : $\neg p \wedge p$ n'a pas de modèle.

Theorème 10 *Tout programme prolog possède un plus petit modèle.*

Soit P un programme Prolog.

1. Si p est un fait du programme P et v un modèle de P alors $p \in \mathcal{M}_v$ par définition.
2. L'opérateur $T_P : PROP \rightarrow PROP$ est l'opérateur de *conséquence immédiate* associé à P) définit par

$$T_P(E) = \{p \mid \exists p_1 \wedge \dots \wedge p_m \Rightarrow p \in P, p_1, \dots, p_m \in E\}$$

On considère la suite :

$$\begin{aligned} T_0 &= \emptyset \\ T_1 &= T_P(T_0) = \{p \mid p \text{ fait de } P\} \\ &\dots \\ T_n &= T_P(T_{n-1}) \cup T_{n-1} \end{aligned}$$

$T_n(P)$ représente ce qu'on peut déduire en au plus $n - 1$ utilisations de règles de P .

Il existe toujours un n_0 (dépendant du programme P) tel que

$$T_{n_0}(P) = T_{n_0+1}(P) = \dots$$

On appelle \mathcal{M}_μ l'ensemble $T_{n_0}(P)$ (c'est le point stationnaire de la suite (T_n) ou point fixe de l'opérateur T définit par $T(E) = T_P(E) \cup E$).

3. Tout modèle \mathcal{M} de P contient \mathcal{M}_μ (montrer par récurrence sur n que \mathcal{M} contient $T_n(P)$ pour tout n .
 - Vrai pour $n = 0$.
 - On suppose que c'est vrai à l'étape n . Soit \mathcal{M} un modèle de P . Il est modèle de $p_1 \wedge \dots \wedge p_m \Rightarrow p$ et si $p_i \in T_n$ pour $i = 1, \dots, m$ alors nécessairement $p \in \mathcal{M}$. Donc $T_{n+1} \subseteq \mathcal{M}$.
4. \mathcal{M}_μ est un modèle. Par construction il est modèle de tous les fait de P . Soit $p_1 \wedge \dots \wedge p_m \Rightarrow p$ une règle de P . Alors soit un des $p_i \notin \mathcal{M}_\mu$ et donc \mathcal{M}_μ est un modèle de la règle, soit pour tout $i = 1, \dots, m$, $p_i \in \mathcal{M}_\mu$ et donc par définition du point fixe $p \in \mathcal{M}_\mu$.

On en déduit :

Theorème 11 \mathcal{M}_μ est l'ensemble des conséquences logiques de P qui sont des symboles propositionnels.

Il suffit de se rappeler que les conséquences logiques de P sont les formules vraies dans tous les modèles de P .

5.3 Correction et complétude de la programmation logique

Soit P un programme logique.

Proposition 15 Si l'interpréteur prolog répond oui au but $\neg p$ alors p est dans le plus petit modèle de P

Preuve par induction sur l'arbre de résolution.

Une stratégie équitable pour la résolution est une stratégie qui assure qu'un sous-but qui peut être résolu le sera par l'interpréteur. L'implémentation de l'interpréteur vue ci-dessus n'est pas équitable.

Proposition 16 Soit p une conséquence logique de P . Alors une stratégie équitable retourne oui au but $\neg p$.