La *stratégie négative* consiste à restreindre l'application d'une résolution binaire au cas où une des deux prémisses ne contient que des littéraux négatives (l'application de la factorisation n'est pas restreinte). On note $S \vdash^*_{R1\neg} C$ quand une clause C peut être obtenue à partir d'un ensemble de clauses S par résolution binaire en stratégie négative.

Théorème 3.7.2 *La résolution binaire restreinte à la stratégie négative est réfutationnellement complète : Un ensemble S de clauses universelles closes est non satisfaisable si et seulement si $S \vdash^*_{R1\neg} \Box$.*

Preuve:
Dans le cas de la logique propositionnelle, la stratégie négative est réfutationnellement complète (c'est une exercice dans le chapitre sur la résolution dans la logique propositionnelle). Puisque S est non-satisfaisable, il y a une preuve $H(S) \vdash^*_{R\neg} \Box$. Relèvement (comme dans la preuve du lemme 3.7.3) donne une preuve par résolution du premier ordre qui suit la stratégie négative. $\Box$

Exercice 104
Montrer, qu'en général la preuve la plus courte par résolution négative est exponentiellement plus longue que la preuve la plus courte par résolution :

Donner une famille $(S_n)_{n\geq 1}$ d'ensembles finis de clauses universelles, telle que toute réfutation de S_n par résolution négative a la longueur $2^n + c_1$ pour une constante c_1, mais qu'il existe une réfutation de S_n par résolution de longueur $n + c_2$ pour une constante c_2.

Chapitre 4

Programmation logique

La *programmation logique* a été introduite au début des années 70 par les équipes d'Alain Colmerauer (Marseille) et de Bob Kowalski (Edinburgh). Il s'agit d'un formalisme de programmation radicalement différent de la programmation classique (impérative, fonctionnelle) : L'idée de base est celle de la *programmation déclarative*, qu'on peut résumer avec les mots de Bob Kowalski comme :

Programmation = *Logique* + *Contrôle*

Ce qui veut dire : Un programme consiste en une spécification du problème dans un langage logique. L'exécution du programme est un mécanisme de déduction, qui utilise un contrôle particulier pour la rendre suffisamment efficace.

En particulier dans la programmation logique,

- Le langage logique consiste des clauses de Horn du premier ordre.
- Le mécanisme de déduction est la résolution.
- Le contrôle utilisé est la stratégie SLD qui sera expliquée plus tard dans ce chapitre.

Nous donnons d'abord dans la Section 4.1 une introduction à la théorie de la programmation logique. Puis nous présenterons dans la section 4.2 le langage Prolog, qui est le langage prototypique de la programmation logique. Finalement nous reviendrons aux aspects théoriques de la programmation logique et discuterons dans la Section 4.3 la négation dans la programmation logique.

4.1 La programmation logique pure

4.1.1 Syntaxe et sémantique dénotationnelle

Definition 4.1.1 *Une* clause définie *(angl. : definite clause) (aussi appelée* clause universelle de Horn stricte*, ou encore* clause de programme*) est une formule universelle close de la forme*

$$\forall \bar{x}(A \vee \neg B_1 \vee \ldots \vee \neg B_n)$$

où $n \geq 0$*,* $A, B_1, \ldots, B_n$ *sont des atomes, et* $\bar{x} = Var(A) \cup \bigcup_{i=1}^{n} Var(B_i)$*.*

Un programme logique *est un ensemble fini de clauses définies. Nous disons que* P *est un* $\mathcal{F}, \mathcal{P}$*-programme logique quand* $\mathcal{F}$ *est l'ensemble des symboles de* *fonctions qui apparaissent en P, et $\mathcal{P}$ est l'ensemble des symboles de prédicat qui apparaissent en P.*

Traditionnellement, on n'écrit pas les quantificateurs (de toute façon, toutes les variables sont universellement quantifiées), et on écrit la clause en forme d'une implication où la virgule dénote la conjonction. Par exemple, la clause définie

$$\forall x, y, z((P(f(x), y) \vee \neg Q(x, g(y)) \vee \neg R(y, h(z)))$$

est notée comme

$$P(f(x), y) \leftarrow Q(x, g(y)), R(y, h(z))$$

On dit aussi que $P(f(x), y)$ est la *tête* de la clause, et $Q(x, g(y)), R(y, h(z))$ son *corps*.

Par le théorème 3.5.6, tout programme logique possède un modèle de Herbrand le plus petit.

Definition 4.1.2 *Soit P un $\mathcal{F}, \mathcal{P}$-programme logique. On suppose que $\mathcal{F}$ contient une constante, sinon on ajoute à $\mathcal{F}$ une constante a. La* sémantique *de P est son $\mathcal{F}, \mathcal{P}$-modèle de Herbrand le plus petit $\mathcal{M}_P$.*

Exemple 4.1.1 Soit le programme logique suivant :

$$\begin{array}{lcl} P(0, y, y) & & \\ P(s(x), y, s(z)) & \leftarrow & P(x, y, z) \end{array}$$

Sa sémantique est $\{P(s^n(0), s^m(0), s^{n+m}(0) | n, m \geq 0\}$.

Remarquons que, par conséquence de l'exercice 103, la sémantique $\mathcal{M}_P$ d'un programme logique est récursivement énumérable.

Definition 4.1.3 *Une* requête *(angl. :* query*) est une formule de la forme*

$$\exists \bar{x}(A_1 \wedge \ldots \wedge A_n)$$

où les A_i sont des atomes, et $\bar{x} = \bigcup_{i=1}^{n} Var(A_i)$.

Une clause de but *(angl. :* goal clause*), aussi appelée* clause universelle négative*, est une formule de la forme*

$$\forall \bar{x}(\neg A_1 \vee \ldots \vee \neg A_n)$$

où les A_i sont des atomes, et $\bar{x} = \bigcup_{i=1}^{n} Var(A_i)$

Évidemment, chaque requête est équivalente à la négation d'une clause de but, et inverse. Toute clause universelle de Horn est soit une clause définie, soit une clause de but.

On s'intéresse à la question :

> Étant donné un programme logique P et une requête R, est-ce que $P \models R$?

Cette question est équivalente à la question

> Étant donné un programme P et une requête R, est-ce que $P \cup \{\neg R\}$ est non-satisfaisable ?

Exemple 4.1.2 En continuation de l'exemple 4.1.1, on peut avoir par exemple des requêtes comme

$$
\begin{array}{l}
P(s^2(0), s^{17}(0), s^{42}(0)) \\
\exists x\, P(s^{17}(0), s^{42}(0), x) \\
\exists x\, P(x, s^{17}(0), s^{59}(0)) \\
\exists x, y\, (P(s(0), s^2(0), x) \wedge P(y, s^5(0), x))
\end{array}
$$

Quand la requête R est

$$\exists \bar{x}(A_1 \wedge \ldots \wedge A_n)$$

alors la clause de but correspondante, c'est-à-dire équivalente à $\neg R$, est

$$\forall \bar{x}(\neg A_1 \vee \ldots \vee \neg A_n)$$

Traditionnellement, on utilise les mêmes conventions que pour les programmes logiques, et on écrit la clause de but comme

$$\leftarrow A_1, \ldots, A_n$$

Proposition 4.1.1 *Soit P un programme logique et R une requête. Alors $P \models R$ si et seulement si $\mathcal{M}_P \models R$.*

Exercice 105
Démontrer Proposition 4.1.1

Plus généralement que la question posée au-dessus, on s'intéresse aussi au problème suivant :

> Étant donné un programme logique P et une requête $R = \exists \bar{x}(A_1 \wedge \ldots \wedge A_n)$, calculer l'ensemble des substitutions closes σ telles que $\mathcal{M}_P \models (A_1 \wedge \ldots \wedge A_n)\sigma$.

4.1.2 Sémantique de point fixe

Soient $\mathcal{F}$ un ensemble de symboles de fonction qui contient une constante, et $\mathcal{P}$ un ensemble de symboles de prédicat. Nous notons $HB(\mathcal{F}, \mathcal{P})$ la $\mathcal{F}, \mathcal{P}$-base de Herbrand.

Nous notons $HS(\mathcal{F}, \mathcal{P})$ pour l'ensemble des parties de $HB(\mathcal{F}, \mathcal{P})$. Nous avons vu dans la section 3.5.3 que $HS(\mathcal{F}, \mathcal{P})$ est essentiellement l'ensemble des $\mathcal{F}, \mathcal{P}$-structures de Herbrand. $(HS(\mathcal{F}, \mathcal{P}), \subseteq)$, où $\subseteq$ est l'ordre d'inclusion, est un cpo pointé (en fait, c'est même un treillis complet)[1].

[1] Voir le cours *Programmation 1* pour les définitions d'un cpo pointé, d'un ensemble dirigé, etc.

Definition 4.1.4 *Soit P un $\mathcal{F},\mathcal{P}$-programme logique. L'*opérateur de conséquence immédiate associé à *P est $T_P\colon HS(\mathcal{F},\mathcal{P}) \to HS(\mathcal{F},\mathcal{P})$, défini par*

$$T_P(S) := \{A\sigma \mid (A \leftarrow B_1,\ldots,B_n) \in P; \sigma \in \Sigma_g; B_1\sigma,\ldots,B_n\sigma \in S\}$$

Exercice 106
Montrer pour tout programme logique P que T_P est continu.

Les modèles de Herbrand d'un programme logique P sont exactement les points fixes de T_P :

Proposition 4.1.2 *Soit P un $\mathcal{F},\mathcal{P}$-programme logique. Pour toute $\mathcal{F},\mathcal{P}$-structure de Herbrand $\mathcal{H}$: $\mathcal{H} \models P$ si et seulement si $T_P(\mathcal{H}) = \mathcal{H}$.*

Preuve:
Immédiate par la définition de T_P. □

Proposition 4.1.3 *Soit P un $\mathcal{F},\mathcal{P}$-programme de logique.*

$$\mathcal{M}_P = \bigcup_{i=0}^{\infty} T_P^i(\emptyset)$$

Preuve:
Par la continuité de T_P, et un théorème du cours *Programmation.* □

Exemple 4.1.3 Soit le programme logique suivant :

$$\begin{array}{rcl} & & P(0,y,y) \\ P(s(x),y,s(z)) & \leftarrow & P(x,y,z) \end{array}$$

On obtient pour $n \geq 1$:

$$T_P^n(\emptyset) = \{P(s^{n-1}(0), s^m(0), s^{m+n-1}(0)) \mid m \geq 0\}$$

et, donc, $\mathcal{M}_P = \{P(s^n(0), s^m(0), s^{n+m}(0)) \mid n,m \geq 0\}$.

Cette proposition justifie la notation $T_P \uparrow \omega$ pour le point fixe le plus petit de T_P. Ici, ω dénote l'ordinal qui est la limite de la suite des nombres naturels.

4.1.3 Sémantique opérationnelle : LD-résolution

Definition 4.1.5 *Soit P un programme logique et R une requête. Une* LD-dérivation *à partir de $P \cup \{\neg R\}$ est une séquence (finie ou infinie) $L_1,\ldots,$ de clauses universelles négatives, telle qu'il existe une séquence de substitutions $\theta_1,\ldots,\theta_{n-1}$ avec*

1. *$L_1 = \neg R$;*

2. *pour tout i, (où $i < n$ si la dérivation est finie et de longueur n) : L_i est de la forme*
$$L_i = (\leftarrow A_1, \ldots, A_n, P(\bar{t}), B_1, \ldots, B_m)$$
et il existe une clause du programme de la forme
$$P(\bar{s}) \leftarrow C_1, \ldots, C_k$$
telle que
$$L_{i+1} = (\leftarrow A_1, \ldots, A_n, C_1, \ldots, C_k, B_1, \ldots, B_m)\theta_i$$
où θ_i est un mgu de $\bar{s} \stackrel{?}{=} \bar{t}$.

On dit aussi que $P(\bar{t})$ est l'atome sélectionné *à l'étape i.*

Une LD-réfutation *est une LD-dérivation finie, et où la dernière clause est $\Box$. La* substitution de réponse *de cette réfutation est la substitution $\theta_1 \circ \ldots \circ \theta_{n-1}$.*

Le nom *LD*-résolution vient de l'anglais *Linear resolution for Definite clauses.*

Théorème 4.1.1 *Pour tout programme logique P et toute requête R : $P \models R$ si et seulement s'il y a une réfutation de $P \cup \{\neg R\}$ par LD-résolution.*

Preuve:
Une telle séquence est une dérivation de $P \cup \{\neg R\} \vdash^*_{R1} \Box$, donc si une telle séquence existe alors $P \models R$.

Si $P \models R$, alors par complétude réfutationelle de la résolution négative (théorème 3.7.2) il existe une déduction par résolution négative $P \cup \{\neg R\} \vdash^*_{R1\neg} \Box$. Remarquez qu'une clause qu'on peut obtenir par résolution binaire à partir d'une clause de programme et une clause négative est toujours une clause négative. Donc, la règle de factorisation n'est jamais appliquée. $\Box$

Donc, si $P \cup \{\neg R\} \vdash^*_{R1\neg} \Box$, alors la déduction a la forme comme indiquée dans la figure 4.1.

Exemple 4.1.4 Un terme de la forme suivante est appelé une *liste*
$$cons(t_1, cons(t_2, \ldots, cons(t_n, nil) \ldots))$$
où les t_i sont des termes quelconques. Soit le programme P_1 suivant :
$$\begin{array}{rcl} List(nil) & \leftarrow & \\ List(cons(x, y)) & \leftarrow & List(y) \end{array}$$
Pour tout terme clos $t \in T(\mathcal{F})$, $P_1 \models List(t)$ ssi t est une liste.

Soit P_2 le programme suivant :
$$\begin{array}{rcl} Nat(0) & \leftarrow & \\ Nat(s(x)) & \leftarrow & Nat(x) \\ IncList(nil) & \leftarrow & \\ IncList(cons(x, nil)) & \leftarrow & Nat(x) \\ IncList(cons(x, cons(s(x), z))) & \leftarrow & IncList(cons(s(x)), z) \end{array}$$

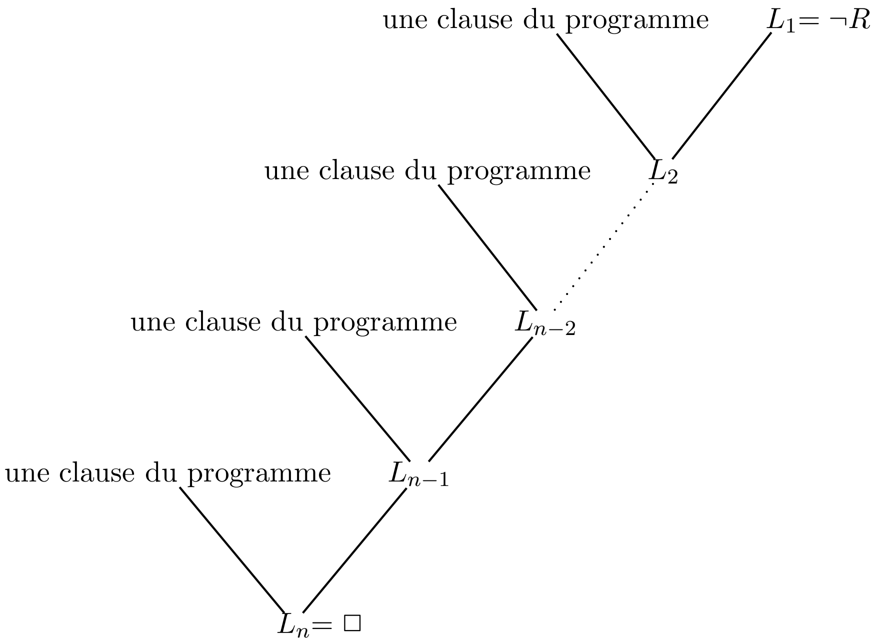

FIG. 4.1 – Une déduction par résolution linéaire.

En confondant un entier n avec le terme $s^n(0)$, on a pour tout terme clos $t \in T(\mathcal{F})$ que $P_2 \models IncList(t)$ si et seulement si t est une liste de nombres naturels $n, n+1, n+2, \ldots$.

Exercice 107

1. Écrire un programme logique P_3 définissant un prédicat $SortedList$ tel que $P_3 \models SortedList(t)$ si et seulement si t est une liste croissante de nombres naturels.
2. Écrire un programme logique P_4 définissant un prédicat $Permutation$ tel que $P_4 \models Permutation(s, t)$ si et seulement si s et t sont des listes, et t est une permutation de s. Définir d'abord un prédicat pour les permutations élémentaires (échange de deux positions voisin), puis exprimer la rélation *permutation* comme clôture réflexive et transitive de la rélation *permutatin élémentaire*.
3. En déduire un programme logique P_5 définissant un prédicat $Sort$, tel que $P_5 \models Sort(s, t)$ si et seulement si s est une liste de nombres naturels, et t est la version triée de s.

Exercice 108

Montrer que la programmation logique est Turing-complète : Construire, pour chaque machine de Turing T un programme logique P_T, et pour chaque mot m de l'alphabet d'entrée une requête R_m, telle que $P_T \models R_m$ si et seulement si la machine T accepte l'entrée m.

Le théorème suivant dit que l'ensemble des substitutions de réponse obtenu par LD-résolution est correct est complet :

Théorème 4.1.2 *Soit P un programme logique, $R = \exists\bar{x}(A_1 \wedge \ldots \wedge A_n)$ une requête, et $\sigma \in \Sigma_g$ une substitution close. Alors les deux énoncés suivants sont équivalents :*

1. $P \models (A_1 \wedge \ldots \wedge A_n)\sigma$
2. *il y a une LD-réfutation de $P \cup \{\neg R\}$ avec substitution de réponse θ et une substitution close $\tau \in \Sigma_g$ telle que $\sigma = \tau \circ \theta$.*

Pour montrer ce théorème nous avons d'abord besoin de deux lemmes. Dans la suite nous écrivons $\tilde{\forall}\phi$ pour la *clôture universelle* d'une formule ϕ, c'est-à-dire $\forall\bar{x}\ \phi$ où $\bar{x} = \mathrm{Var}(\phi)$. (Analogue $\tilde{\exists}\phi$ pour la clôture existentielle d'une formule ϕ). Remarquez que, pour toute substitution θ et formula ϕ, on a que $\tilde{\forall}\phi \models \tilde{\forall}\phi\theta$.

Lemme 4.1.1 *Soit P un programme et $\tilde{\exists}(A_1 \wedge \ldots \wedge A_n)$ une requête. S'il y a une LD-réfutation de $P \cup \{\tilde{\forall}(\neg A_1 \vee \ldots \vee A_n)\}$ avec substitution de réponse θ, alors $P \models \tilde{\forall}((A_1 \vee \ldots \vee A_n)\theta)$.*

Preuve:
On montre d'abord que, quand μ est un mgu de $\bar{s} \stackrel{?}{=} \bar{t}$, alors

$$\tilde{\forall}(P(\bar{s}) \vee L) \models \tilde{\forall}(\neg(L \vee K)\theta \to \neg(\neg P(\bar{t}) \vee K)\theta)$$

L'énoncé en suit par induction. □

Exercice 109
Compléter la preuve du lemme 4.1.1.

Lemme 4.1.2 *Soit P un programme, $\tilde{\exists}(A_1 \wedge \ldots \wedge A_n)$ une requête et σ une substitution. S'il y a une LD-réfutation de $P \cup \{\tilde{\forall}((\neg A_1 \vee \ldots \vee \neg A_n)\sigma)\}$ avec substitution de réponse θ, alors il y a aussi une LD-réfutation de $P \cup \{\tilde{\forall}(\neg A_1 \vee \ldots \vee \neg A_n)\}$ avec substitution de réponse $\theta' \geq \sigma \circ \theta$.*

Exercice 110
Démontrer le lemme 4.1.2.

Preuve:
(du Théorème 4.1.2)
Pour la direction (1) $\Rightarrow$ (2) : Soit $P \models (A_1 \wedge \ldots \wedge A_n)\sigma$. Donc, $P \cup \{(\neg A_1 \vee \ldots \vee \neg A_n)\sigma\}$ n'a pas de modèle. Par la complétude réfutationelle de la LD-résolution, il y a une LD-réfutation de $P \cup \{(\neg A_1 \vee \ldots \vee \neg A_n)\sigma\}$. Par lemme 4.1.2, il y a une LD-réfutation de $P \cup \{\forall\bar{x}(\neg A_1 \vee \ldots \vee \neg A_n)\}$ avec substitution de réponse θ telle que $\theta \geq \sigma$.
Pour la direction (2) $\Rightarrow$ (1) : Soit une LD-réfutation de $P \cup \{\forall\bar{x}(\neg A_1 \vee \ldots \vee \neg A_n)\}$ avec substitution de réponse θ. Par le lemme 4.1.1, $P \models \tilde{\forall}((A_1 \wedge \ldots \wedge A_n)\theta)$. Donc, on a aussi pour toute substitution close $\tau \in \Sigma_g$ que $P \models ((A_1 \wedge \ldots \wedge A_n)\tau)\theta$. □