

# Chapitre 1

## Résolution et Programmation logique

### 1.1 La programmation logique

#### 1.1.1 Clause de Horn

**Definition 1** 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 2** 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.

#### 1.1.2 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  $\text{solve}(\text{LG}, \text{P})$  correspond à la résolution sur  $\mathcal{C} \cup \{\neg g_1 \vee \dots \vee \neg g_n\}$ .

```

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 :

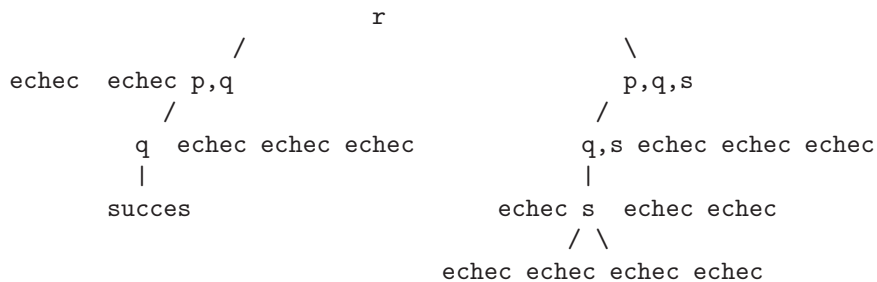
Le programme est

```

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

```

alors pour le but r l'arbre est



## 1.2 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 1** *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  $\mathcal{M}_\mu$  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  $\mathcal{M}_\mu$  (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.  $\mathcal{M}_\mu$  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  $\mathcal{M}_\mu$  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 2**  $\mathcal{M}_\mu$  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  $P$ .

### 1.3 Correction et complétude de la programmation logique

Soit  $P$  un programme logique.

**Proposition 1** 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 2** Soit  $p$  une conséquence logique de  $P$ . Alors une stratégie équitable retourne oui au but  $\neg p$ .