Un exemple de programme Prolog

Référence : Fondements mathématiques de l'informatique.
Jacques Stern

2011-2012

Définition 0.1
On appelle clause de Horn *une clause dont* au plus *un littéral est positif.*

Si exactement *un littéral est positif, on parle de clause définie.*

Si aucun *littéral n'est positif, on parle de but.*

On a alors

Proposition 0.2
Soit Σ un ensemble de clauses de Horn, et soit Σ_0 le sous-ensemble formé des clauses définies. Alors :
– *Σ_0 n'est pas contradictoire ;*
– *si Σ est contradictoire, il existe un but G de Σ tel que $\Sigma_0 \cup G$ soit contradictoire.*

Définition 0.3
*On dit qu'un arbre de résolution est de type LD (*Linéaire et Défini*) s'il se compose d'une branche étiquetée de la feuille vers la racine par des buts $G_0, \ldots, G_m$ et de nœuds terminaux étiquetés par des clauses définies qui sont fils d'éléments de cette branche, $C_0, \ldots, C_{m-1}$.*

Si $C_0, \ldots, C_{m-1} \in \Sigma$ et $G_m = \square$, on dit que l'arbre réfute G_0 à partir de Σ.

Théorème 0.4
Soient Σ un ensemble de clauses définies et G un but. Si $\Sigma \cup \{G\}$ est contradictoire, il existe un arbre de réfutation de type LD qui réfute G à partir de Σ.

De plus, on peut remarquer que la recherche d'un arbre de réfutation LD donne explicitement un contre-exemple. C'est cette propriété qui est utilisée par PROLOG.

PROLOG construit en fait un arbre de réfutation LD par *effacement des buts*.

On opère sur des suites de littéraux positifs $Q_1 \ldots Q_n$, appelées *clauses ordonnées*.

S'il existe une variante d'une règle $L \to P_1 \ldots P_k$, où L s'unifie à Q_1 par l'unificateur principal σ, on dit que $P_1\sigma \ldots P_k\sigma Q_2\sigma \ldots Q_n\sigma$ est obtenue par *effacement* à partir de $Q_1 \ldots Q_n$.

Si la règle est un fait ($k = 0$), on obtient donc $Q_2\sigma \ldots Q_n\sigma$: on a effacé Q_1.

Le choix de la règle à unifier se fait de façon non déterministe. On contruit un *arbre d'effacement de la façon*

suivante :

On note $R_1, \ldots, R_\ell$ les règles dans l'ordre où elles sont écrites, et $Q_1 \ldots Q_n$ le but.

Chaque nœud de l'arbre correspond au but courant, et a autant de fils que de règle qui peuvent s'unifier avec le premier but, dans l'ordre de l'indice.

Regardons maintenant le programme PROLOG suivant :

```
homme(jacques).
homme(julien).
homme(aymeric).
homme(françois).
homme(didier).
femme(brigitte).
femme(martine).
femme(vanessa).
parent(jacques,julien).
parent(jacques,aymeric).
parent(brigitte,julien).
parent(brigitte,aymeric).
parent(martine,françois).
parent(didier,vanessa).
soeur(martine,brigitte).
soeur(brigitte,martine).
soeur(martine,didier).
soeur(brigitte,didier).

fils(X,Y) :- parent(Y,X), homme(X).
fille(X,Y) :- parent(Y,X), femme(X).
frere(X,Y) :- soeur(Y,X), femme(X).
cousin(X,Y) :- fils(X,T), soeur(T,Z), parent(Z,Y).
cousin(X,Y) :- fils(X,T), frere(T,Z), parent(Z,Y).
cousine(X,Y) :- fille(X,T), frere(T,Z), parent(Z,Y).
cousine(X,Y) :- fille(X,T), soeur(T,Z), parent(Z,Y).
```

On pose la question (le but) à PROLOG :

```
cousin(françois,A).
```

PROLOG va donc construire l'arbre d'effacement.

On commence par chercher à unifier `cousin(françois,A)` avec une règle. Les deux règles possibles sont

```
cousin(X,Y) :- fils(X,T), soeur(T,Z), parent(Z,Y).}
cousin(X,Y) :- fils(X,T), frere(T,Z), parent(Z,Y).}
```

dans cet ordre là, avec l'unificateur `X` $\rightarrow$ `françois`.

Le fils gauche de l'arbre est donc

```
fils(françois,T).
```

```
soeur(T,Z).
parent(Z,A).
```

et le fils droit

```
fils(françois,T).
frere(T,Z).
parent(Z,A).
```

On continue ainsi de suite pour obtenir l'arbre d'effacement :