Fonctionnement de Prolog

Pierron Théo

13 avril 2014

Prolog est un langage de programmation logique, qui se base sur la méthode de résolution. On va expliquer ici les grandes lignes de son fonctionnement. On va travailler ici avec des formules sous FNC prénexes skolémisées, qu'on va donc pouvoir décomposer en conjonction de clauses closes.

Définition Une clause de Horn est une clause ayant au plus un littéral positif, c'est-à-dire qu'on peut les écrire sous une des formes suivantes :

- $A_1 \wedge \ldots \wedge A_n \Rightarrow B$: si $n = 0$, B est un fait. Sinon, on parle de règle.
- $A_1 \wedge \ldots \wedge A_n \Rightarrow \bot$: on parle de question ou de but.

On appelle programme un ensemble de règles et de faits.

On utilise la méthode de résolution avec des clauses de Horn pour plusieurs raisons : on a une stratégie de preuve (qu'on va détailler) qui est complète et constructive. De plus, la coupure de deux clauses de Horn reste une clause de Horn.

Si Q est une question, elle correspond à

$$\forall x_1 \ldots \forall x_n(\neg A_1 \vee \ldots \vee \neg A_n) \equiv \neg\exists x_1 \ldots \exists x_n(A_1 \wedge \ldots \wedge A_n)$$

Si on arrive à prouver, étant donné un programme P que $Q \cup P \vdash \bot$, on saura que

$$P \vdash \exists x_1 \ldots \exists x_n A_1 \wedge \ldots \wedge A_n$$

Le fonctionnement de prolog repose sur l'algorithme général suivant :

Algorithme 1: Stratégie de preuve

```
1 L est une liste de buts, P un programme.
2 tant que L ≠ ∅ faire
3 |   Soit B ∈ L.
4 |   si il existe une clause A_1 ∧ ... ∧ A_n → A_0 ∈ P tel que A_0 et B soient unifiables alors
5 |   |   remplacer B par A_1, ..., A_n dans L
6 |   |   L ← σ(L) avec σ un mgu de A_0 et B
```

Cet stratégie n'est pas déterministe car il faut choisir un ordre sur les buts à prendre et les clauses de P à utiliser. On va modéliser la recherche d'une preuve par un arbre. Les nœuds de cet arbre sont étiquetés par des listes de buts. L' est fils de L ssi on a une clause de P qui permet d'obtenir (après coupure et unification) L' à partir de L.

On va donner maintenant un exemple :

$$succ(a, b).$$
$$succ(a, d).$$
$$succ(b, c).$$
$$ch(X, Y) \text{ :- } succ(X, Y).$$
$$ch(X, Z) \text{ :- } succ(X, Y), ch(Y, Z).$$

On veut savoir s'il existe un ch de a à c, i.e. si $ch(a, c)$. On part donc du prédicat $\neg ch(a, c)$ qu'on place en racine de l'arbre de recherche de preuve.

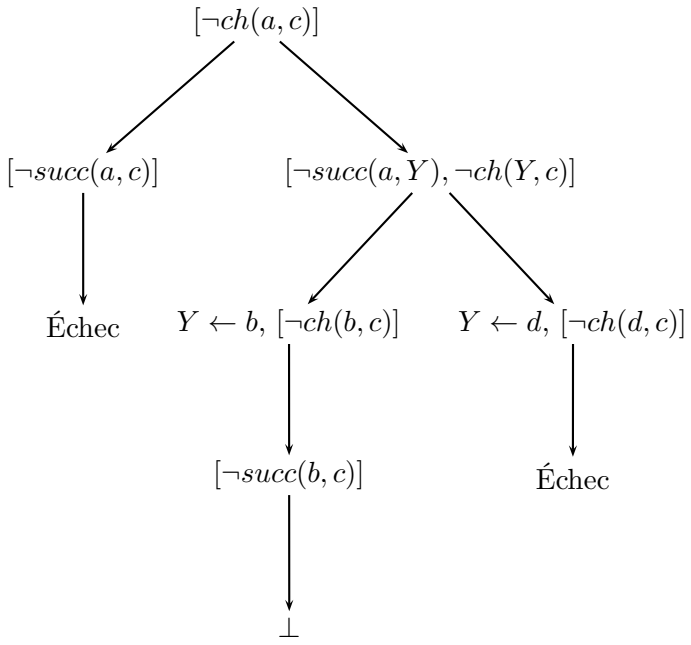

FIGURE 1 – L'arbre de recherche

Il suffit donc de faire un parcours en largeur de l'arbre de recherche. Comme la méthode de résolution est complète, si $P \cup \{Q\}$ est contradictoire, on finira nécessairement par trouver un noeud de l'arbre étiqueté par $[\bot]$. Cependant, faire une parcours en largeur n'est pas très efficace.

On peut utiliser aussi Prolog pour trouver les sommets reliés à un sommet donné. Il suffit de demander $ch(a, X)$: quels sont les sommets accessibles depuis a ? On construit alors un arbre de recherche de preuve similaire et il suffira de renvoyer les images de X par les substitutions qui auront permis de prouver $\bot$ à partir de P et $\neg ch(a, X)$.

Donc b, c et d sont les successeurs de a.

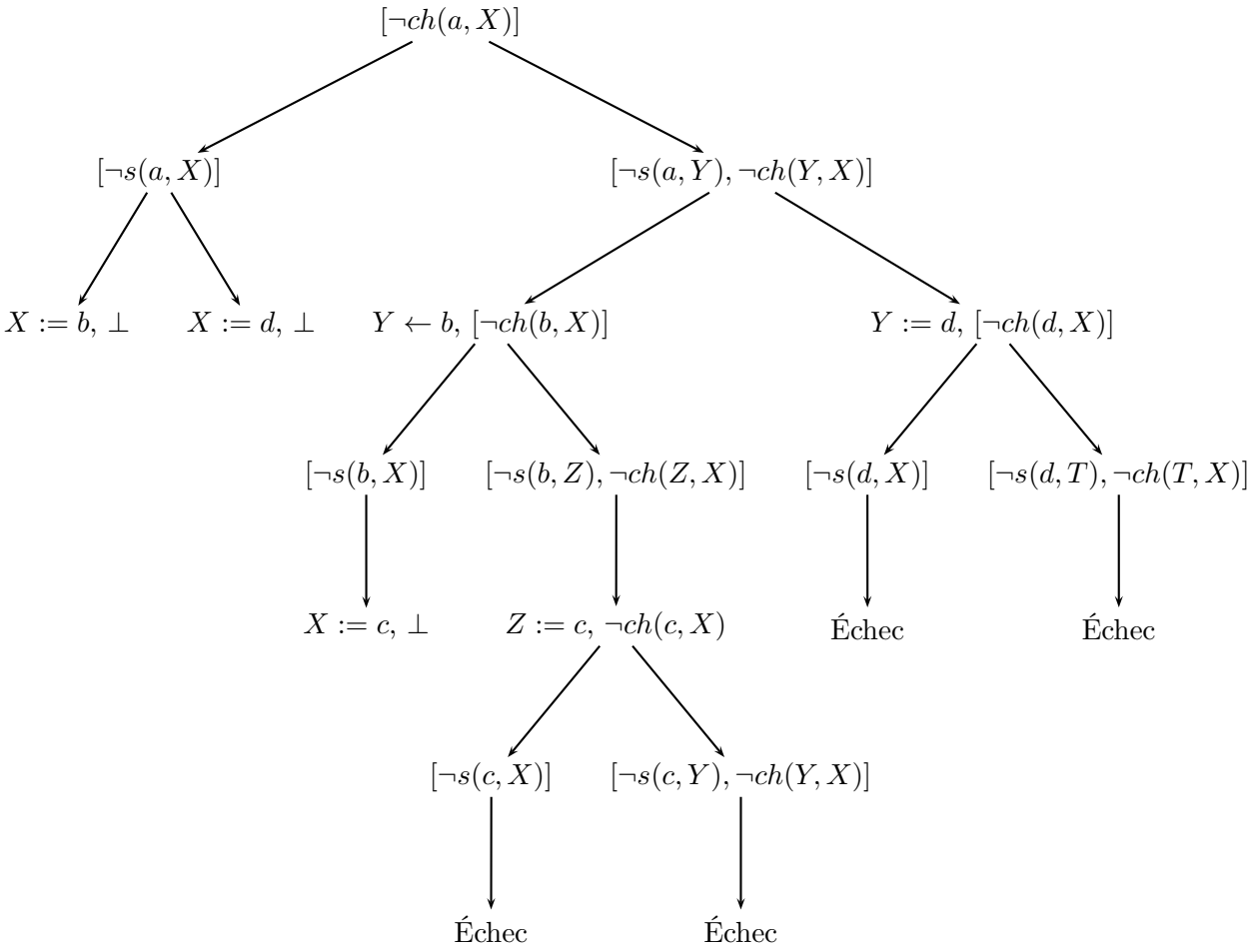

FIGURE 2 – L'arbre de recherche des successeurs de a (avec $s = succ$)