UPMC/master/info/4I503 APS
PROLOG

P. Manoury

Janvier 2018

Le langage PROLOG

Le langage PROLOG est une *langage déclaratif* basé sur la logique du premier ordre; d'où son nom *PROG[rammation] LOG[ique].*

Définir

Un programme en PROLOG est une suite de définitions (en général inductives) de relations; typiquement une relation entre des valeurs d'entrée et des valeurs de sortie. Le format des définitions est celui des *clauses de Horn.* Une telle clause est une formule de la forme $(F_1 \wedge \ldots \wedge F_n) \Rightarrow F$ (qui est équivalente à la relation de conséquence $F_1, \ldots, F_n \models F$). Les formules $F_1, \ldots, F_N$ et F sont des *litéraux* ou des *formules atomiques*: application de symboles de prédicats (ou de relation) à des *termes.* Les clauses de Horn s'écrivent en PROLOG de la manière suivante: F `:-` F_1 `,` $\ldots$ `,` F_n `.`

Par exemple on définira une relation binaire `fib` telle que si `fib(N,M)` est vérifié alors `M` est le `N`-ième nombre de la suite de Fibonacci:

```
fib(N,1) :- N < 2, !.
fib(N,M) :- fib(N-1,M1), fib(N-2,M2), M is M1+M2, !.
```

Notez l'utilisation des majuscules qui en PROLOG marquent les *variables*, c'est-à-dire les identificateurs dont on cherchera à déterminer des *instances.*

Calculer

En effet, à partir de cette définition de `fib`, on peut déterminer pour quelles valeurs de `X` la relation `fib(2,X)` est vérifiée. Le principe de cette détermination est de chercher une *démonstration* de la formule: $\exists$`X.fib(2,X)`. L'algorithme utilisé pour chercher cette preuve est basé sur la stratégie *résolution.* Ainsi, on peut dire que le mécanisme d'évaluation de PROLOG est un moteur de recherche de preuve.

Intuitivement, chaque clause est une règle dont la conclusion est à gauche du symbole `:-` et les conditions (ou prémisses) sont à droite de ce symbole. Pour trouver `X` tel que `fib(2,X)` on cherche une règle applicable à `fib(2,X)`, c'est-à-dire une règle dont la conclusion est *unifiable* avec le *but* à prouver (ici `fib(2,X)`).

Unifier deux termes, c'est trouver une instance commune de leurs variables. Ici, en remplaçant `X` par `1` et `N` par `2`, on obtient `fib(2,1)` qui est une instance commune à `fib(2,X)` et `fib(N,1)`.

La première règle est donc applicable, mais elle dit que pour que `fib(2,1)` il faut que `2 < 2`. Ce qui n'est pas vérifié. La première règle ne nous donne donc pas la valeur recherchée.

On passe à la suivante, qui est applicable et qui nous dit qu'il faut trouver `M` tel que `fib(1,M1)` et `fib(0,M2)` et `M is M1+M2` (c'est-à-dire que `M` est a pour valeur la somme des valeurs de `M1` et `M2`).

On trouvera que `M1` et `M2` peuvent valoir `1` en appliquant la première règle puisque `1<2` et `0<2`. Ce qui nous donne que `M`, et donc `X`, peut prendre la valeur `2`.

Pour avoir le `X` tel que `fib(3,X)`, la première règle *échoue* et la seconde nous demande de trouver `M1` tel que `fib(2,M1)` et `M2` tel que `fib(1,M2)`. On a vu comment obtenir ces valeurs et nous auront `3` pour valeur de `X`.

etc.

Listes et couples

PROLOG connait également les structures de listes. La liste vide est notée `[]`, la liste commençant par `X` et se poursuivant par `XS` est notée `[X|XS]`. Par exemple, on définit que «`S` est la somme des valeurs de la liste `NS`» en définissant le prédicat binaire `sum` de la manière suivante:

```
sum(0,[]).
sum(S,[N|NS]) :- sum(N1,NS), N is N1+1.
```

On peut également noter les listes *en extension*, par exemple `[1,2,3,4]` et chercher le `X` tel que `sum(X,[1,2,3,4])`.

Autre exemple: le prédicat d'appartenance à une liste se définit comme:

```
mem(X,[X|_]).
mem(X,[_|XS]) :- mem(X,XS).
```

Notez comment l'égalité est simplement signifiée par l'usage d'un même nom et comment on utilise l'ordre de définition des clauses.

Enfin, on peut former des paires de termes avec la notation usuelle `(X,Y)`.

SWI prolog

Pour réaliser nos programmes PROLOG, on pourra utiliser SWI prolog. La commande `swipl` nous donne une boucle d'interaction avec l'environnement de SWI prolog:

```
manoury@ssh:~$ swipl
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 6.6.6)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

?-
```

L'invite `?-` nous invite à formuler une requête comme `fib(5,X)` pour obtenir la valeur de `X`.

Naturellement, cette requête n'a de chance d'aboutir que si nous avons informé le système de nos définitions. Supposons que celles-ci ont été enregistrées dans un fichier appelé `prog.pl` (extension obligatoire). Nous «chargeons» ce fichier avec la requête:

```
?- [prog].
true.

?-
```

La réponse du système est `true`, les définitions sont maintenant connues et nous pouvons demander:

```
?- fib(5,X).
X = 8.

?-
```

La réponse du système est ici la valeur qu'il a pu déterminer pour la variable présente dans notre requête.

Script

La commande `swipl` peut prendre des arguments comme le nom d'un fichier de définitions et le but initial à satisfaire. Par exemple

```
swipl -s Typage/typeChecker.pl -g main_stdin
```

est équivalente à

```
swipl
?- [Typage/typeChecker].
?- main_stdin.
```

Nous avons défini le prédicat `main_stdin` de cette manière:

```
main_stdin :-
    read(user_input,T),
    typeProg(T,R),
    print(R),
    nl,
    exitCode(R).
```

où

1. `read(user_input,T)` lit un terme PROLOG (sensé être la traduction d'un programme APS) sur l'entrée standard et le lie à `T`,
2. puis, `typeProg(T,R)` applique les règles de typage au terme `T` pour construire le résultat `R` (atome `ok` ou `ko`),
3. ce résultat est affiché puis le programme est stopé avec un code de retour (O ou 1) déterminé par la valeur de `R`.

Si `translate` est un programme qui produit sur la sortie standard la traduction en terme PROLOG d'un programme APS contenu dans un fichier est passé sur la ligne de commande, le script

```
#!/usr/bin/env bash
Syntaxe/translate $1 | swipl -s Typage/typeChecker.pl -g main_stdin
```

permet d'enchaîner la traduction d'un programme APS et sa vérification de type. On pourra utiliser le code de retour du typage pour enchaîner ou non sur l'évaluation.