SU/FS/master/info/MU4IN503 APS
Prolog

P. Manoury

Février 2021

Langage des termes pour Prolog

Lexique:

undef _

num [0-9]+

const [a-z][a-zA-Z0-9]*

var [A-Z][a-zA-Z0-9]*

Grammaire:

Term ::= Atom | App | Nuplet | List

Atom ::= undef | const | var | num

App ::= const(Term,...,Term)

Nuplet ::= (Term,...,Term)

List ::= [] | [Term,...,Term] | [Term|Term]

Définitions:

Un *terme clos* ne contient ni undef, ni var.

Dans const(Term,...,Term), const est le *symbole de tête*.

Les termes non atomiques énoncent des *prédicats*, ou *relations* entre termes. Le nom de la relation est le symbole de tête.

Clauses

Syntaxe:

Clause ::= Term . | Term :- Term, ..., Term .

Les clauses se terminent par un point.
Les clauses de la forme Term . sont *immédiates*.
Les clauses de la forme Term :- Term , ... , Term . sont *conditionnelles*.
Les clauses immédiates sont des *faits*.
Les clauses conditionnelles sont des *règles* qui permettent de déduire de nouveaux faits.
La règle t :- t_1, ..., t_n . se lit comme: si t_1 et ...et t_n alors t.

Dans un «programme» prolog, les clauses sont regroupées par symbole de tête.

Exemple Un graphe et ses chemins

```
arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).

arc(x,y).
arc(y,x).

existsPath(X,Y) :- arc(X,Y).
existsPath(X,Y) :- arc(X,Z), existsPath(Z,Y).

pathList(X,Y,[X,Y]) :- arc(X,Y).
pathList(X,Y,[X|R]) :- arc(X,Z), pathList(Z,Y,R).
```

Interprétation Le prédicat `arc` définit les *arêtes* entre sommets du graphe.
Les clauses immédiates (`arc`) définissent **un** graphe: *relation* binaire entre les sommets {`a,b,c,d,x,y`}.
Les éléments `a` et `c` satisfont la relation `arc` car on a, parmi les faits que `arc(a,c)`.
Les éléments `a` et `d` ne satisfont pas la relation `arc` car:

1. `arc(a,d)` ne figure parmi les faits.
2. aucune règle de permet de le déduire.

Le prédicat `existsPath` définit l'existence d'un chemin entre deux sommets: clôture transitive de `arc`.
Les éléments `a` et `c` satisfont la relation `existsPath` car:

- la clause conditionnelle `existsPath(X,Y) :- arc(X,Y).` dit que *pour tout terme `X` et tout terme `Y`, si `arc(X,Y)` alors `existsPath(X,Y)`*;
- or `arc(a,c)` est un fait;
- donc `existsPath(a,c)`.

Les éléments `a` et `d` satisfont la relation `existsPath` car

- la clause `existsPath(X,Y) :- arc(X,Z), existsPath(Z,Y).` dit que *pour tout `x, Y`, si il existe `Z` tel que `arc(X,Z)` et `existsPath(Z,Y)` alors `existsPath(X,Y)`.*
- or `arc(a,c)`
- et `existsPath(c,d)` car
  - `arc(c,d)`.
- donc `existsPath(a,d)`.

C'est une première raison. Il y en a d'autres

- `arc(a,b)`
- et `existsPath(b,d)` car
  - `arc(b,d)`

- donc `existsPath(a,d)`.

Le système prolog sait «calculer» toutes ces raisons.

Il y a une infinité de raisons d'avoir `existsPath(x,y)`:

1. `arc(x,y)`.
2. `arc(x,y)` et `existsPath(y,y)` car
   - `arc(y,x)` et `existsPath(x,y)` car
     - `arc(x,y)`.
3. `arc(x,y)` et `existsPath(y,y)` car
   - `arc(y,x)` et `existsPath(x,y)` car
     - `arc(x,y)` et `existsPath(y,y)` car
       - `arc(y,x)` et `existsPath(x,y)` car
         - `arc(x,y)`.
4. etc.

Le prédicat ternaire `pathList(X,Y,Z)` est satisfait lorsque `Z` est la liste des sommets d'un chemin de `X` à `Y`. Notons que `pathList(X,Y,Z)` est satisfait si et seulement si `existsPath(X,Y)` l'est aussi. L'ajout du 3ème terme au prédicat permet de «calculer» le chemin: une liste de sommets.

A-t-on `P` tel que `pathList(a,d,P)` ? Oui car `pathList(a,c,[a,c,d])`, c-à-d, `pathList(a,d,[a|[c,d]])`. En effet

- `arc(a,c)` et `pathList(c,d,[c,d]])` car
  - `arc(c,d)`

Autres exemples Manipulation de listes: concaténation et association.

```
%% append
append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).

%% assoc
assoc(X,[(X,V)|_],V).
assoc(X,[_|XVs],V) :- assoc(X,XVs,V).
```

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. Il est installé sur les machines de la PPTI:

```
manoury@ppti-14-407-05:~$ swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 8.0.2)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
```

L'invite `?-` nous invite à formuler une requête comme `listPath(a,d,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:

```
?- pathList(a,d,X).
X = [a,b, d]
```

Si nous entrons un retour-chariot après cette réponse, le système s'arrête là (pour cette requête). Pour obtenir d'autre réponses, il faut entrer une espace. On obtiendra ainsi une suite de réponses:

```
?- pathList(a,d,X).
X = [a, b, d] ;
X = [a, b, c, d] ;
X = [a, c, d] ;
false.
```

La suite se termine par `false` lorsqu'il n'y a plus de réponse possibles.

Pour sortir de `swipl`, tapez `^D` ou `halt.` (avec un point à la fin).

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),
    typeCheck(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`.

On définit également

```
typeCheck(P,ok) :- typeProg(P).
typeCheck(_,ko).
```

et

```
exitCode(ok) :- halt(0).
exitCode(_) :- halt(1).
```

Si `prologTerm` 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/prologTerm $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.