

SU/FS/master/info/MU4IN503 APS

Prolog

P. MANOURY

Janvier 2020

Langage des termes

undef _

num [0-9]+

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

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

TERM

ATOM undef | const | var | num

APP const(TERM, ..., TERM)

NUPLET (TERM, ..., TERM)

LIST [] | [TERM, ..., TERM] | [TERM|TERM]

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

immédiate TERM. clos;

conditionnelle TERM :- TERM, ..., TERM.

Les clauses se terminent par un point. Les clauses immédiates sont des *faits*. Les clauses conditionnelles sont des *règles* qui permettent de déduire d'autres de nouveaux faits.

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 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.

Les éléments `a` et `c` satisfont le 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 `pathList(X,Y,[X|R]) :- arc(X,Z), pathList(Z,Y,R)` 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.

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:

```
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 `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 entrns 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.

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 (0 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.