Cours de programmation fonctionnelle et logique

Cours 10 : l'unification
... ou pourquoi 1+1 n'est pas égal à 2

Table des matières

1 Retour sur le principe d'effacement

1.1 Calcul de la longueur d'une liste

Le prédicat qui calcule la longueur d'une liste est défini au moyen de deux règles :

1. `longueur([],0).`
2. `longueur([T|R], N) :- longueur(R, M), N is M+1.`

Question :

```
?- longueur([3],1).
```

Effacement :

1. On teste la règle 1. Les contraintes sont `[] = [3]` et `0 = 1`. Générer de telles contraintes semble assez facile. Ici, on n'a pas de solution, l'interpréteur renvoie `No`.
2. On teste la règle 2. Les contraintes sont ici : `T=3`, `R = []` et `N = 1`. Comment obtient-on la décomposition de la liste argument ?

   Le but devient `longueur(R,M), N is M+1`.

Unifier c'est

1. Tester les égalités `[] = [1]` et `0 = 1`.
2. Tester les égalités `[T|R] = [1]` et `N = 1`. Ici on essaye d'unifier `[T|R]` et `[1]`. Autrement dit, on essaye de rendre « un », identique, les deux expressions.

1.2 1+1 $\neq$ 2

On définit le prédicat suivant :

```
égal_à_deux(X) :- X = 1+1.
```

où « = » est le test général d'égalité.

```
?- égal_à_deux(2).

No
?- égal_à_deux(1+1).

Yes
?- égal_à_deux(X).

X = 1+1 ;

No
```

Pourquoi ?

2 Représentation des prédicats par des arbres

- Expression arithmétique : $23 = 43 * 60$ (figure 1).

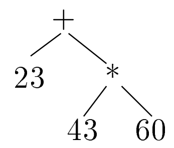

FIG. 1: Représentation d'une expression arithmétique par un arbre.

- Relation (prédicat) : `longueur([],0)` (2).
- Cas général : prédicat R d'arité n (figure 3).

Avec une telle représentation, les deux expressions $23 + (43 * 60)$ et $23 + (60 * 43)$ sont *a priori* différente (cf. figure 4).

Les structures-arbres présentées ici ne sont pas forcément complètement définies. Prenons l'exemple

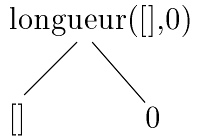

FIG. 2: Représentation d'un prédicat.

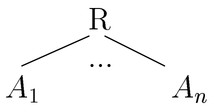

FIG. 3: Représentation par un arbre : cas général.

de l'expression arithmétique : $8 + X + 3$. L'arbre présenté figure 5 représente l'infinité des arbres obtenus en remplaçant la variable X par un arbre quelconque. Ici, X peut être remplacé par une valeur ou, plus généralement, par un arbre.

Les arbres élémentaires sont des constantes :

- entier
- chaîne de caractères ;
- identificateur ;
- ...

3 Unification

Prolog est un langage qui manipule des arbres qui peuvent contenir des parties variables. Un mécanisme de base consiste à décider si deux arbres *peuvent être rendus égaux*, ou non. Le cas échéant, l'égalité est réalisée en affectant des variables aux valeurs figurant dans l'un ou l'autre des arbres considérés.

Algorithme classique d'unification

Soient deux arbres t_1 et t_2.
Unification(t_1, t_2)

1. t_1 est constant ou est une variable de valeur connue, t_2 est constant ou est variable de valeur connue. Alors $t_1 = t_2$.
2. t_1 est une variable de valeur inconnue et t_2 est constant ou est variable de valeur connue. Alors `valeur(`t_1`)` $\rightarrow$ t_2`, vrai`.

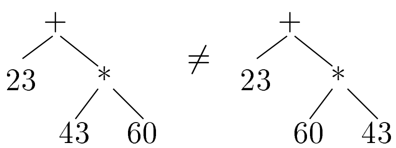

FIG. 4: Deux expressions *a priori* différentes.

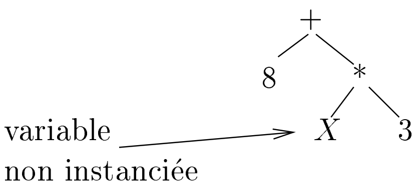

FIG. 5: Arbre non complètement défini.

3. t_1 et t_2 sont des variables de valeur inconnue. Alors `valeur(`t_1`)` $\rightarrow$ t_2`, vrai`.
4. $t_1 = f(u_1, ..., u_n)$ et $t_2 = f(u'_1, ..., u'_n)$. Alors Unification(u_1, u'_1), ...,Unification(u_n, u'_n).

Exemples

1. ```
   nul(0).
   ?- nul(1).

   No
   ?- nul(X).

   X = 0 ;

   No
   ?
   ```
   Unification(0,1) $\Leftrightarrow$ $0 = 1$: faux.
   Unification(X,0) $\Leftrightarrow$ `valeur(X) = 0` : vrai.
2. ```
   nul(X), un(X).

   No
   ```
   Unification(X,0) $\Leftrightarrow$ `valeur(X) = 0` : vrai. Unification(X,1) $\Leftrightarrow$ $0 = 1$: faux.
3. (a) ```
      egal(X,X).
      ?- egal(Y,Z).

      Y = _G153
      Z = _G153

      Yes
      ```
      _G153 est un nom de variable.

   (b) ```
      ?- egal(f(0,X),f(Y,1)).

      X = 1
      Y = 0 ;

      No
      ```
      Première unification : `f(0,X)` avec `f(Y,1)`.
      Deuxième unification : 0 avec `Y`.
      Troisième unification : `X` avec 1.

   (c) `?- egal(f(X,2,g(Z)),f(g(1),Y,g(4))).`

```
X = g(1)
Z = 4
Y = 2 ;

No
```

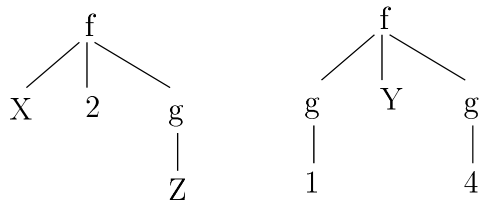

FIG. 6: Deux arbres à unifier.

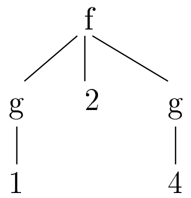

FIG. 7: Résultat de l'unification.

Première unification : `f(X,2,g(Z))` avec `f(g(1),Y,g(4))`.

Deuxième unification : `X = g(1)`.

Troisième unification : `Y = 2`.

Quatrième unification : `g(Z)` avec `g(4)`.

Cinquième unification : `Z=4`.

(d) `egal(f(Y,1,g(X)),f(X,Y,g(2))).`

```
No
```

Première unification : `f(Y,1,g(X))` avec `f(X,Y,g(2))`.

Deuxième unification : `X = Y`.

Troisième unification : `Y = 1` (donc `X=1`).

Quatrième unification : `g(X)` avec `g(2)`.

Cinquième unification : `X=2`, or `X=1`, donc `1=2` : faux.

4. `inf(X,f(aa,X)).`

```
?- inf(Y,Z).

Y = _G147
Z = f(aa, _G147) ;

No
?- inf(Y,Y).
```

Première unification : `X = Y` (`X` et `Y` non connus).

Deuxième unification : `Y = f(aa,X)` et `Y` non connu. Donc `Y` de la forme `Y = f(Z,T)`. `Z = aa`, `T = X`, avec ici `X = Y` et `Y = f (Z,T)`. D'où `T = f(aa,T)`. Ça part à l'infini : objet à définition récursive.

5. `mult(X,Y,Z) :- 2*(X+5) = Y * Z.`

```
?- mult(X,Y,Z).

X = _G165
Y = 2
Z = _G165+5

Yes
?
```

6. `égal_à_deux(X) :- X = 1+1.`

```
?- égal_à_deux(2).

No
```

On tente l'unification de `2` et de `1+1`, qui sont deux expressions de racines différentes : `2` et `+`, d'où l'échec.