

UNIVERSITÉ DE LORRAINE



FACULTÉ DE SCIENCE ET TECHNOLOGIE

M1 INFORMATIQUE

Martelli Montanari

ALGORITHME D'UNIFICATION

PROJET DE LMC

Auteurs :
Aurélien THIRION
Nicolas BLIN

11 décembre 2016

Introduction

Nous avons voulu, dans ce projet, essayer d'être le plus clair et simple possible en disposant dans différents fichiers le code source de notre programme. Ainsi dans le code principal, il y a le minimum de ligne pour comprendre réellement le processus de l'unification de Martelli-Montanari. Notre code principal se situe dans **main.pl**, des prédicats sont appelés lors du processus dans les fichiers suivants : **opérateur.pl**, **predicatsRelais.pl** et **reglesTest.pl**.

opérateur.pl

Contient l'opérateur $?=$ et le code pour le echo contenu dans le mail.

predicatsRelais.pl

Fonction de décomposition d'une équation E pour obtenir la partie gauche de l'opérateur $?=$ et la partie droite.

reglesTest.pl

Toutes les conditions nécessaires pour appliquer la réduction ($\text{prédicat_reduit}(R,E,P,Q)$).

Question 1

Mise en place pour l'unification

On peut voir ici les instructions pour l'ouverture des différents fichiers.

```
1 :-  
2     [opérateurs],  
3     [predicatsRelais],  
4     [reglesTest].
```

Listing 1 – Dans le fichier *main.pl*

Un exemple du prédicat `regle(E,R)` dans le code.

```
1 regle(E, rename)
```

Listing 2 – Dans le fichier *main.pl*

Qui va appeler le prédicat dans le fichier **reglesTest.pl**. Ici on va séparer l'équation `E` en les termes `X` et `T`. Il va ensuite vérifier l'applicabilité de la règle *rename*.

```
1 regle(E, rename):-  
2     splitEquation(E,X,T),  
3     var(T),  
4     var(X).
```

Listing 3 – Dans le fichier *reglesTest.pl*

Qui va appeler le prédicat dans le fichier **predicatsRelais.pl**. On récupère la partie gauche et droite de `?=` pour le mettre dans `X` et `T` respectivement.

```
1 splitEquation(E,X,T):-  
2     arg(1,E,L),  
3     arg(2,E,R),  
4     X = L,  
5     T = R.
```

Listing 4 – Dans le fichier *predicatsRelais.pl*

Occur check

Le prédicat *occur_{check}* a été assez simple à mettre en oeuvre. Il sera chargé de vérifier si `V` apparaît dans le terme composé `T`.

```
1 occur_check(V,T):-  
2     compound(T),  
3     var(V),  
4     contains_var(V,T).
```

Listing 5 – Dans le fichier *main.pl*

Reduit

Voici un exemple de *reduit* qui se charge d'appliquer la règle *rename*.

```
1 réduit(rename, E, P, Q):-  
2     splitEquation(E,X,T),  
3     X = T,  
4     P = [_|Q].
```

Listing 6 – *reduit* dans *main.pl*

Question 2

Dans le main

Question 3

Dans le main

Pour le prédicat *unif*, il s'agit simplement de désactiver le echo puis d'appeler le prédicat *unifie*.

```
1 unif(P,S) :-  
2   clr_echo,  
3   unifie(P,S).
```

Listing 7 – unif dans *main.pl*

Pour le prédicat *trace_unif*, il faut activer le echo en précisant que si l'unification marche il y aura affiché un "Yes" ou un "No" dans le cas contraire.

```
1 trace_unif(P,S) :-  
2   set_echo,  
3   (unifie(P,S),  
4   echo("Yes"),  
5   !;  
6   echo("No")).
```

Listing 8 – trace unif dans *main.pl*

Une fois que ces prédcats sont fait, il suffit de mettre les echo dans le code qui sera commun aux différentes stratégies. Ainsi nous aurons par exemple dans le prédicat *reduit* :

```
1 réduit(rename, E, P, Q):-  
2   echo("system: "),echo(P),nl,  
3   echo("rename: "),echo(E),nl,  
4   splitEquation(E,X,T),  
5   X = T,  
6   P = [_|Q].
```

Listing 9 – Les echos dans *main.pl*

Code source

Dans le main

```
1 occur_check(V,T):-
2     compound(T),
3     var(V),
4     contains_var(V,T).
5
6 unif(P,S) :-
7     clr_echo,
8     unify(P,S).
9
10 trace_unif(P,S) :-
11     set_echo,
12     (unify(P,S),
13     echo("Yes"),
14     !;
15     echo("No")).
16
17 unify([], _) :- !.
18 unify([]) :- !.
19
20
21 unify(P):-
22     unify(P, regle),
23     !.
24
25 unify(P, regle):- unify(P, rename).
26 unify(P, regle):- unify(P, simplify).
27 unify(P, regle):- unify(P, expand).
28 unify(P, regle):- unify(P, check).
29 unify(P, regle):- unify(P, orient).
30 unify(P, regle):- unify(P, decompose).
31 unify(P, regle):- unify(P, clash).
32
33
34 unify(P, rename) :-
35     P = [E | _],
36     regle(E, rename),
37     reduit(rename, E, P, Q),
38     unify(Q, regle),!.
39
40
41 unify(P, simplify):-
42     P = [E | _],
43     regle(E, simplify),
44     reduit(simplify, E, P, Q),
45     unify(Q, regle),!.
46
47 unify(P, expand):-
48     P = [E | _],
49     regle(E, expand),
50     reduit(expand, E, P, Q),
51     unify(Q, regle),!.
52
```

```

53 unify(P, check):-
54     P = [E | _],
55     regle(E, check),
56     reduit(check, E, P, Q),
57     unify(Q, regle),!.
58
59 unify(P, orient):-
60     P = [E | _],
61     regle(E, orient),
62     reduit(orient, E, P, Q),
63     unify(Q, regle),!.
64
65 unify(P, decompose):-
66     P = [E | _],
67     regle(E, decompose),
68     reduit(decompose, E, P, Q),
69     unify(Q, regle),!.
70
71 unify(P, clash):-
72     P = [E | _],
73     regle(E, clash),
74     reduit(clash, E, P, Q),
75     unify(Q, regle),!.
76
77
78 reduit(decompose, E, P, Q):-
79     splitEquation(E,X,T),
80     functor(X,_,ArityX),
81     functor(T,_,_),
82     P = [_|Tail],
83     repet(X,T,ArityX,Tail,Q),
84     echo("decompose: "),echo(Q),nl.
85
86 repet(_,_,0,T,Q):- Q = T, !.
87 repet(X,T,N,Tail,Q) :-
88     N > 0,
89     arg(N,X,ValX),
90     arg(N,T,ValT),
91     Var = [ValX?=ValT|Tail],
92     N1 is N - 1,
93     repet(X,T,N1,Var,Q).
94
95 reduit(rename, E, P, Q):-
96     splitEquation(E,X,T),
97     X = T,
98     P = [_|Q].
99
100 reduit(simplify, E, P, Q):-
101     splitEquation(E,X,T),
102     X = T,
103     P = [_|Q].
104
105 reduit(expand, E, P, Q):-
106     splitEquation(E,X,T),
107     X = T,
108     P = [_|Q].
109
110 reduit(check, _, _, _):-
111     fail,
112     !.
113
114 reduit(orient, E, P, Q):-
115     splitEquation(E,X,T),
116     P = [_|Tail],
117     Q = [T ?= X | Tail].
118

```

```
119
120 reduit(clash, _, _, _):-
121     fail,
122     !.
```

Listing 10 – *main.pl*