Unification de Martelli-Montanari

Creusel Victor - Heitzmann Maureen

14 décembre 2017

Part I
Code source

```
/* UNIFICATION DE MARTELLI-MONTANARI */


/* Prédicats utiles */

:- op(20,xfy,?=).


/* Predicat qui sert à vérifier si un terme est une fonction */
fonc(X)
:- nonvar(X), functor(X, _, A), A > 0 .


/* Predicat qui sert à vérifier si un terme est une constante */
const(X)
:- nonvar(X), functor(X, _, 0).


/* Predicat qui trouve toutes les occurences de E dans P et qui les supprimes pour donner
 ↪  le nouveau système Q. select et delete de prolog n'effectuent pas l'opération
 ↪  demandée*/
delete_p(_, [], []) :- !.

delete_p(X, [E | P], Qout)
:- X \== E, append([E], Q, Qout), delete_p(X, P, Q), !.

delete_p(X, [E | P], Q)
:- X == E, delete_p(X, P, Q), !.



:- dynamic echo_on/0.
% Prédicats d affichage fournis

% set_echo: ce prédicat active l affichage par le prédicat echo
set_echo :- assert(echo_on).

% clr_echo: ce prédicat inhibe l affichage par le prédicat echo
clr_echo :- retractall(echo_on).

% echo(T): si le flag echo_on est positionné, echo(T) affiche le terme T
%          sinon, echo(T) réussit simplement en ne faisant rien.

```

```
42  echo(T) :- echo_on, !, writeln(T).
43  echo(_).
44
45  /* Nous avons légerement modifié le prédicat echo(T) pour qu'il fasse automatiquement les
    ↪  retours à la ligne après l'appel de chaque echo(T) */
46
47
48
49  /* Vérifie si X apparait dans T */
50  occur_check(X, T)
51  :- var(X), term_variables(T, Arguments), occur_check_list(X, Arguments), !.
52
53  occur_check_list(X, [E | List])
54  :- var(X), same_term(X, E) ; occur_check(X, List).
55
56
57
58
59  /* REGLES : détermine la règle de transformation R qui s'applique à l'équation E
60  Le nom d'une règle est défini ainsi : X_r (ex : rename_r)
61  */
62
63  regle(X ?= T, rename_r)
64  :- var(X), var(T), !.
65
66  regle(X ?= T, simplify_r)
67  :- ((var(X), const(T)) ; (X==T, const(T))), !.
68
69  regle(X ?= T, expand_r)
70  :- var(X), fonc(T), \+occur_check(X, T), !.
71
72  regle(X ?= T, check_r)
73  :- var(X), fonc(T), occur_check(X, T), !.
74
75  regle(T ?= X, orient_r)
76  :- var(X), \+var(T), !.
77
78  regle(X ?= T, decompose_r)
79  :- fonc(X), fonc(T), functor(X, N1, A1), functor(T, N2, A2), A1 =:= A2, N1 == N2, !.
80
81  regle(X ?= T, clash_r)
82  :- fonc(X), fonc(T), functor(X, N1, A1), functor(T, N2, A2), (A1 =\= A2 ; N1 \== N2), !.
83
84
85
86  /* REDUIT : transforme le système d'équations P en le système d'équations Q par
    ↪  application de la règle de transformation R à l'équation E. */
87  /* On produit la constante bottom à la place de la liste de sortie lors d'un clash ou
    ↪  check, pour que bottom ne soit unifiable avec aucune liste par la suite */
88
89  /* rename */
90  reduit(rename_r, X ?= T, P, Q)
91  :- append(P, [], Q), X = T, !.
92
93  /*simplify*/
94  reduit(simplify_r, X ?= T, P, Q)
95  :- append(P, [], Q), X = T, !.
96
97  /*expand*/
```

```
98  reduit(expand_r, X ?= T, P, Q)
99  :- append(P, [], Q), X = T, !.
100
101 /*check */
102 reduit(check_r, _, _, bottom).
103
104 /*orient*/
105 reduit(orient_r, T ?= X, P, [X ?= T | P]).
106
107 /*clash*/
108 reduit(clash_r, _, _, bottom).
109
110 /*decompose*/
111 reduit(decompose_r, X ?= T, P, Pout)
112 :- X =.. [_ |F], T =.. [_ |G], decompose_aux(F, G, Q), append(P, Q, Pout).
113
114         /*Lout est la liste des équations créées à partir des éléments présents dans les
            ↪  listes L1 et L2*/
115         decompose_aux([], [], []).
116
117         decompose_aux([X1 | L1], [X2 | L2], Lout)
118         :- decompose_aux(L1, L2, Q), append([X1 ?= X2], Q, Lout).
119
120
121
122 /* Unifie(P) : utilise les prédicats regle(E, R) et reduit(R, E, P, Q) et effectue
    ↪  l'unification de Martelli-Montanari sur la liste d'équations P.
123 unifie(P) correspond à unifie(P, choix_premier) défini par la suite */
124
125 unifie([]).
126
127 unifie([E | P])
128 :- regle(E, R), reduit(R, E, P, Q), unifie(Q), !.
129
130
131 /* Unifie(P,S) : effectue l'unification de Martelli-Montanari sur la liste d'équations P
    ↪  en utilisant la stratégie S
132 S peut prendre comme valeur "choix_premier", "choix_pondere" ou "choix_aleatoire"
133 */
134 unifie([], _).
135
136 unifie(P, choix_premier)
137 :- choix_premier(P, Q, E, R), reduit(R, E, Q, Q1), echo(system: P), echo(R: E),
138 unifie(Q1, choix_premier), !.
139
140
141 unifie(P, choix_pondere)
142 :- choix_pondere(P, Q, E, R), reduit(R, E, Q, Q1), echo(system: P), echo(R: E),
143 unifie(Q1, choix_pondere), !.
144
145
146 unifie(P, choix_aleatoire)
147 :- choix_aleatoire(P, Q, E, R), reduit(R, E, Q, Q1), echo(system: P), echo(R: E),
148 unifie(Q1, choix_aleatoire), !.
149
150
151
152 /* affichages exécution */
```

```
153 /* trace_unif(P, S) appelle unifie(P, S) et autorise l'affichage des echo(T) présents
    ↪  dans ce-dernier. */
154 trace_unif(P, S)
155 :- set_echo, unifie(P, S), clr_echo, !.
156
157
158 /* unif(P, S) appelle unifie(P, S) mais n'autorise pas l'affichage des echo(T) présents
    ↪  dans ce-dernier. */
159 unif(P, S)
160 :- clr_echo, unifie(P, S), !.
161
162
163
164
165 /* choix_premier : sélectionne le premier élément E du système P et retourne le nouveau
    ↪  sytème P \ {E}, l'élément E sélectionné et la règle R à appliquer sur cet élément E
    ↪  */
166 choix_premier([E | P], P, E, R)
167 :- regle(E, R), !.
168
169
170
171 /* choix_pondere :
172 on donne maintenant un poid à chaque règle selon le modèle suivant : clash, check >
    ↪  rename, simplify > orient > decompose > expand
173 ainsi, clash et check sont prioritaires par rapport aux règles rename et simplify ... etc
    ↪  ...
174 On va donc chercher, dans le système P, l'élément E sur lequel la règle R avec la plus
    ↪  forte priorité peut s'appliquer.
175 On retourne cet élement E et sa règle associée R, ainsi que le nouveau système Q, où Q =
    ↪  P \ {E}.
176 */
177 choix_pondere(P, Q, E, R)
178 :- ( extrait_clash_check(P, E, R); extrait_rename_simplify(P, E, R); extrait_orient(P, E,
    ↪  R); extrait_decompose(P, E, R); extrait_expand(P, E, R) ),
179 delete_p(E, P, Q), !.
180
181
182     /*clash et check */
183     extrait_clash_check([E | _], E, R)
184     :- ( regle(E, clash_r) ; regle(E, check_r) ), regle(E, R).
185
186     extrait_clash_check([E | P], ESortie, RSortie)
187     :- \+regle(E, clash_r), \+regle(E, check_r),
188     extrait_clash_check(P, ESortie, RSortie).
189
190     /* rename et simplify */
191     extrait_rename_simplify([E | _], E, R)
192     :- ( regle(E, rename_r) ; regle(E, simplify_r) ), regle(E, R).
193
194     extrait_rename_simplify([E | P], ESortie, RSortie)
195     :- \+regle(E, rename_r), \+regle(E, simplify_r),
196     extrait_rename_simplify(P, ESortie, RSortie).
197
198     /* orient */
199     extrait_orient([E | _], E, orient_r)
200     :- regle(E, orient_r).
201
202     extrait_orient([E | P], ESortie, RSortie)
```

```
    :- \+regle(E, orient_r),
    extrait_orient(P, ESortie, RSortie).

    /* decompose */
    extrait_decompose([E | _], E, decompose_r)
    :- regle(E, decompose_r).

    extrait_decompose([E | P], ESortie, RSortie)
    :- \+regle(E, decompose_r),
    extrait_decompose(P, ESortie, RSortie).

    /* expand */
    extrait_expand([E | _], E, expand_r)
    :- regle(E, expand_r).

    extrait_expand([E | P], ESortie, RSortie)
    :- \+regle(E, expand_r),
    extrait_expand(P, ESortie, RSortie).


/* choix_aleatoire : sélectionne aléatoirement un élément E du système P et retourne le
 ↪  nouveau sytème P \ {E}, l'élément E sélectionné et la règle R à appliquer sur cet
 ↪  élément E*/
choix_aleatoire(P, Q, E, R)
:- random_member(E, P), regle(E, R), delete_p(E, P, Q), !.
```

Part II
Tests

```
/* tests de fonc(X) */
?- fonc(f(a)).
true.

?- fonc(f(X)).
true.

?- fonc(c).
false.

?- fonc(X).
false.


/* tests de const(X) */
?- const(a).
true.

?- const(X).
false.

?- const(f(a)).
false.

?- const(f(X)).
false.


```

```
/* tests de delete_p(Elem, Lin, Lout) */
?- delete_p(a, [a, b, c], Lout).
Lout = [b, c].

?- delete_p(a, [a, b, c, d, a], Lout).
Lout = [b, c, d].

?- delete_p(a, [b, c, d], Lout).
Lout = [b, c, d].

?- delete_p(a, [], Lout).
Lout = [].

?- delete_p(a, [X, Y, Z], Lout).
Lout = [X, Y, Z].

?- delete_p(X, [a, b, X], Lout).
Lout = [a, b].

?- delete_p(f(X), [a, b, f(X)], Lout).
Lout = [a, b].


/* tests de occur_check(X, T) */
?- occur_check(X, f(X)).
true.

?- occur_check(X, f(a, b, c)).
false.

?- occur_check(X, f(Y, X, Z)).
true.

?- occur_check(X, f(Y)).
false.


/*tests des règles */
/* rename */
?- regle(X ?=T, R).
R = rename_r.

?- regle(X ?= a, rename_r).
false.

?- regle(X ?= f(a), rename_r).
false.

?- regle(X ?= f(X), rename_r).
false.


/* simplify */
?- regle(X ?=t, R).
R = simplify_r.

?- regle(X ?= a, simplify_r).
true.

```

```
?- regle(X ?= f(a), simplify_r).
false.

?- regle(X ?= f(X), simplify_r).
false.


/* expand */
?- regle(X ?=f(t), R).
R = expand_r.

?- regle(X ?= a, expand_r).
false.

?- regle(X ?= f(a), expand_r).
true.

?- regle(X ?= f(X), expand_r).
false.


/* check */
?- regle(X ?= f(X), R).
R = check_r.

?- regle(X ?= a, check_r).
false.

?- regle(X ?= f(a), check_r).
false.

?- regle(X ?= f(X), check_r).
true.


/* orient */
?- regle(t ?= X, R).
R = orient_r.

?- regle(t ?= X, orient_r).
true.

?- regle(X ?= f(a), orient_r).
false.

?- regle(X ?= a, orient_r).
false.


/* decompose */
?- regle(f(t) ?= f(X), R).
R = decompose_r.

?- regle(f(X) ?= X, decompose_r).
false.

?- regle(f(X) ?= f(X, Y), decompose_r).
false.

```

```
?- regle(f(X) ?= f(a), decompose_r).
true.

?- regle(f(X) ?= g(Y), decompose_r).
false.


/* clash */
?- regle(f(t) ?= g(X, a), R).
R = clash_r.

?- regle(f(X) ?= X, clash_r).
false.

?- regle(f(X) ?= f(X, Y), clash_r).
true.

?- regle(f(X) ?= f(a), clash_r).
false.

?- regle(f(X) ?= g(Y), clash_r).
true.


/* tests reduit */
/* rename */
?- reduit(rename_r, X ?= T, [], Q).
X = T,
Q = [].

?- reduit(rename_r, X ?= T, [f(X) ?= X], Q).
X = T,
Q = [f(T)?=T].


/* simplify */
?- reduit(simplify_r, X ?= t, [], Q).
X = t,
Q = [].

?- reduit(simplify_r, X ?= t, [f(X) ?= X], Q).
X = t,
Q = [f(t)?=t].


/* expand */
?- reduit(expand_r, X ?= f(t), [], Q).
X = f(t),
Q = [].

?- reduit(expand_r, X ?= f(t), [f(X) ?= X], Q).
X = f(t),
Q = [f(f(t))?=f(t)].


/* check */
?- reduit(check_r, X ?= f(t, Y, X, k), [], Q).
Q = bottom.

```

```
?- reduit(check_r, X ?= f(t, Y, X, k), [f(X) ?= X], Q).
Q = bottom.


/* orient */
?- reduit(orient_r, t ?= X, [], Q).
Q = [X?=t].

?- reduit(orient_r, t ?= X, [f(X) ?= X], Q).
Q = [X?=t, f(X)?=X].


/* clash */
?- reduit(clash_r, f(t) ?= g(X), [], Q).
Q = bottom.

?- reduit(clash_r, f(t) ?= f(X, m), [], Q).
Q = bottom.

?- reduit(clash_r, f(t) ?= g(X), [f(X) ?= X], Q).
Q = bottom.


/* decompose */
?- reduit(decompose_r, f(t, C, X) ?= f(X, k, Y), [], Q).
Q = [t?=X, C?=k, X?=Y].

?- reduit(decompose_r, f(t) ?= f(X), [f(X) ?= X], Q).
Q = [f(X)?=X, t?=X].


/* unifie(P) */
?- unifie([X ?= b]).
X = b.

?- unifie([X ?= X]).
true.

?- unifie([X ?= Y]).
X = Y.

?- unifie([X ?= f(X)]).
false.

?- unifie([X ?= f(a)]).
X = f(a).

?- unifie([a ?= a]).
true.

?- unifie([a ?= b]).
false.

?- unifie([a ?= f(a)]).
false.

?- unifie([f(X) ?= X]).
false.
```

```
?- unifie([f(X) ?= a]).
false.

?- unifie([f(X) ?= f(a)]).
X = a.

?- unifie([f(X) ?= f(X)]).
true.

?- unifie([f(a) ?= f(a)]).
true.

?- unifie([f(X, k) ?= f(X,n)]).
false.

?- unifie([f(a) ?= f(X)]).
X = a.

?- unifie([f(a, b) ?= f(X, Y)]).
X = a,
Y = b.

?- unifie([f(a, X) ?= f(a, b)]).
X = b.

?- unifie([f(X) ?= f(Y)]).
X = Y.

?- unifie([f(X) ?= f(g(Y))]).
X = g(Y).

?- unifie([f(X) ?= f(X,Y)]).
false.

?- unifie([f(X, Y) ?= f(g(Z), h(a)), Z ?= f(Y)]).
X = g(f(h(a))),
Y = h(a),
Z = f(h(a)).

?- unifie([f(X, Y) ?= f(g(Z), h(a)), Z ?= f(X)]).
false.


/* tests trace_unif(P, S) */
/* choix premier */
?- trace_unif([Z ?= f(Y), f(X, Y) ?= f(g(Z), h(a))], choix_premier).
system:[f(_G4306)?=f(_G4306),f(_G4314,_G4306)?=f(g(f(_G4306)),h(a))]
expand_r:f(_G4306)?=f(_G4306)
system:[f(_G4314,_G4306)?=f(g(f(_G4306)),h(a))]
decompose_r:f(_G4314,_G4306)?=f(g(f(_G4306)),h(a))
system:[g(f(_G4306))?=g(f(_G4306)),_G4306?=h(a)]
expand_r:g(f(_G4306))?=g(f(_G4306))
system:[h(a)?=h(a)]
expand_r:h(a)?=h(a)
Z = f(h(a)),
Y = h(a),
X = g(f(h(a))).

/* choix pondere */
```

```
?- trace_unif([Z ?= f(Y), f(X, Y) ?= f(g(Z), h(a))], choix_pondere).
system:[_G4308?=f(_G4306),f(_G4314,_G4306)?=f(g(_G4308),h(a))]
decompose_r:f(_G4314,_G4306)?=f(g(_G4308),h(a))
system:[f(_G4306)?=f(_G4306),_G4314?=g(f(_G4306)),_G4306?=h(a)]
expand_r:f(_G4306)?=f(_G4306)
system:[g(f(_G4306))?=g(f(_G4306)),_G4306?=h(a)]
expand_r:g(f(_G4306))?=g(f(_G4306))
system:[h(a)?=h(a)]
expand_r:h(a)?=h(a)
Z = f(h(a)),
Y = h(a),
X = g(f(h(a))).

/* choix aleatoire */
?- trace_unif([Z ?= f(Y), f(X, Y) ?= f(g(Z), h(a))], choix_aleatoire).
system:[f(_G4306)?=f(_G4306),f(_G4314,_G4306)?=f(g(f(_G4306)),h(a))]
expand_r:f(_G4306)?=f(_G4306)
system:[f(_G4314,_G4306)?=f(g(f(_G4306)),h(a))]
decompose_r:f(_G4314,_G4306)?=f(g(f(_G4306)),h(a))
system:[g(f(_G4306))?=g(f(_G4306)),_G4306?=h(a)]
expand_r:g(f(_G4306))?=g(f(_G4306))
system:[h(a)?=h(a)]
expand_r:h(a)?=h(a)
Z = f(h(a)),
Y = h(a),
X = g(f(h(a))).

?- trace_unif([Z ?= f(Y), f(X, Y) ?= f(g(Z), h(a))], choix_aleatoire).
system:[_G4308?=f(_G4306),f(_G4314,_G4306)?=f(g(_G4308),h(a))]
decompose_r:f(_G4314,_G4306)?=f(g(_G4308),h(a))
system:[f(_G4306)?=f(_G4306),_G4314?=g(f(_G4306)),_G4306?=h(a)]
expand_r:f(_G4306)?=f(_G4306)
system:[g(f(_G4306))?=g(f(_G4306)),_G4306?=h(a)]
expand_r:g(f(_G4306))?=g(f(_G4306))
system:[h(a)?=h(a)]
expand_r:h(a)?=h(a)
Z = f(h(a)),
Y = h(a),
X = g(f(h(a))).
```