```
 1 :- style_check(-singleton).
 2
 3 % Définition de l'opérateur ?=
 4 :- op(20,xfy,?=).
 5
 6 % Définition des règles de Martinelli-Montanari
 7
 8 % rename
 9 % regle (x?=t, rename) : true si x et t sont des variables
10 % regle (x?=t, rename) : false sinon
11 regle(E, rename) :- arg(1, E, X), arg(2, E, T), var(X), var(T), !.
12
13 % simplify
14 % regle (x?=t, simplify) : true si x est une variable et t est une constante
15 % regle (x?=t, simplify) : false sinon
16 regle(E, simplify) :- arg(1, E, X), arg(2, E, T), var(X), atomic(T), !.
17
18 % expand
19 % regle (x?=t, expand) : true si t est composé et x n'apparait pas dans t
20 % regle (x?=t, expand) : false sinon
21 regle(E, expand) :- arg(1, E, X), arg(2, E, T), compound(T), var(X), \+
   occur_check(X,T), !.
22
23 % check
24 % regle (x?=t, check) : true si x =/= t et x apparait dans t
25 % regle (x?=t, check) : false sinon
26 regle(E, check) :- arg(1, E, X), arg(2, E, T), var(X), X \== T, !, occur_check(X,
   T), !.
27
28 % orient
29 % regle (t?=x, orient) : true si x est une variable et t ne l'est pas
30 % regle (t?=x, orient) : false sinon
31 regle(E, orient) :- arg(1, E, T), arg(2, E, X), var(X), nonvar(T), !.
32
33 % decompose
34 % regle (x?=t, decompose) : true si x et t sont des fonctions de même symbole et
   la même arité
35 % regle (x?=t, decompose) : false sinon
36 regle(E, decompose) :- arg(1, E, X), arg(2, E, T), compound(X), compound(T),
   functor(X, N, A), functor(T, O, B), N==O, A==B, !.
37
38 % clash
39 % regle (x?=t, clash) : true si x et t sont des fonctions de même symbole et une
   arité différente ou x et t sont des fonctions de symboles différents
40 % regle (x?=t, clash) : false sinon
41 regle(E, clash) :- arg(1, E, X), arg(2, E, T), atomic(X), atomic(T), X\==T, !;
   arg(1, E, X), arg(2, E, T), not(var(X)), not(var(T)), functor(X, N, A), functor(T
   , O, B), (N \== O ; A \== B), !.
42
43 % clear
44 % regle (x?=t, clear) : true si x et t sont des constantes de même symbole
45 % regle (x?=t, clear) : false sinon
46 regle(E, clear) :- arg(1, E, X), arg(2, E, T), atomic(X), atomic(T), X==T, !.
47
48 % occur_check
49 % occur_check(X, T) : true si X apparait dans T
50 % occur_check(X, T) : false sinon
```

```
51 occur_check(X, T) :- compound(T), var(X), arg(_,T,V), (V==X ; (compound(V),
   occur_check(X, V)) ), !.
52
53 % reduit
54 % reduit(R, E, P, Q) : R est une règle de MM,
55 % E est une équation,
56 % P est un systeme d'équations,
57 % Q est un systeme d'équations résultant de l'application de R sur E
58
59 % application de la regle rename
60 % remplacement des occurrences de la variable X par la variable T
61 reduit(rename, E, P, Q) :- regle(E, rename), echo('rename: '), echo(E), nl, arg(
   1, E, X), arg(2, E, T), Q=P, X=T, !.
62
63 % application de la regle simplify
64 % remplacement des occurrences de la variable X par la constante T
65 reduit(simplify, E, P, Q) :- regle(E, simplify), echo('simplify: '), echo(E), nl
   , arg(1, E, X), arg(2, E, T), X=T, Q=P, !.
66
67 % application de la regle expand
68 % remplacement des occurrences de la variable X par la fonction T
69 reduit(expand, E, P, Q) :- regle(E, expand), echo('expand: '), echo(E), nl, arg(
   1, E, X), arg(2, E, T), X=T, Q=P, !.
70
71 % application de la regle orient
72 % inverse T et X
73 reduit(orient, E, P, Q) :- regle(E, orient), echo('orient: '), echo(E), nl, arg(
   1, E, T), arg(2, E, X), append([X?=T], P, Q), !.
74
75 % application de la regle decompose
76 % decompose les fonctions X et T en une liste d'équations
77 reduit(decompose, E, P, Q) :- regle(E, decompose), echo('decompose: '), echo(E
   ), nl, arg(1, E, X), arg(2, E, T), X=..[_|L], T=..[_|K], union_list(L, K, R),
   append(R, P, Q), !.
78
79 % application de la regle clear
80 % supprime l'équation E
81 reduit(clear, E, P, Q) :- regle(E, clear), echo('clear: '), echo(E), nl, Q=P, !.
82
83 % application de la regle clash
84 reduit(clash, E, _, _) :- regle(E, clash), echo('clash: '), echo(E), nl, !, fail
   .
85
86 % application de la regle check
87 reduit(check, E, _, _) :- regle(E, check), echo('check: '), echo(E), nl, !, fail
   .
88
89 % union_list
90 % union des termes de deux listes pour appliquer la regle decompose
91 union_list([X|A], [Y|B], C) :- union_list(A, B, D), append([X?=Y], D, C).
92 union_list([], [], C) :- C=[].
93
94
95 % unifie
96 % unifie(P) : P est un systeme d'équations à résoudre sous la forme d'une liste
97 % unifie(P, S) : P est un systeme d'équations à résoudre sous la forme d'une
   liste et S la stratégie utilisé pour le résoudre
```

```
 98 % unifie([]) : termine l'éxécution du programme
 99
100 unifie([E|P], choix_premier) :- echo('system:'), echo([E|P]), nl, choix_premier(
    P, Q, E, _), !, unifie(Q, choix_premier).
101 unifie([E|P], choix_pondere_1) :- echo('system:'), echo([E|P]), nl,
    choix_pondere_1(P, Q, E, _), !, unifie(Q, choix_pondere_1).
102 unifie([E|P], choix_pondere_2) :- echo('system:'), echo([E|P]), nl,
    choix_pondere_2(P, Q, E, _), !, unifie(Q, choix_pondere_2).
103 unifie(P, choix_aleatoire) :- echo('system:'), echo(P), nl, choix_aleatoire(P, Q
    ), !, unifie(Q, choix_aleatoire).
104
105 unifie([], _) :- write("L'unification est termine avec succes").
106
107 %Unifie avec un seul parametre (choix_premier)
108 unifie([E|P]) :- choix_premier(P, Q, E, _), !, unifie(Q).
109 unifie([]) :- write("L'unification est termine avec succes").
110
111
112 % Les différentes stratégies possibles pour le résolution de l'équation
113
114 choix_premier([], Q, [], R).
115
116 % Stratégie de choix premier
117 % choix_premier(P, Q, E, R)
118 choix_premier(P, Q, E, R) :- reduit(R, E, P, Q).
119
120 % Stratégies de choix pondérés
121 % choix pondéré 1
122 choix_pondere_1([], Q, E, R) :- choix_reduit_1(E, R, _), reduit(R, E, [], Q).
123 choix_pondere_1(P, Q, E, R) :- parcours_liste_1(P, E, C, L), choix_reduit_1(C, R
    , _), reduit(R, C, L, Q).
124
125
126 % choix pondéré 2
127 choix_pondere_2([], Q, E, R) :- choix_reduit_2(E, R, _), reduit(R, E, [], Q).
128 choix_pondere_2(P, Q, E, R) :- parcours_liste_2(P, E, C, L), choix_reduit_2(C, R
    , _), reduit(R, C, L, Q).
129
130 %Choix de l'équation de façon aléatoire dans la liste
131 %Si la liste à un seul élément, on choisit cette équation
132 %Si la liste à plusieurs éléments, on choisit une équation aléatoirement
133 choix_aleatoire([E|[]], Q) :- reduit(_, E, [], Q).
134 choix_aleatoire(L, Q) :- random_select(E, L, R), reduit(_, E, R, Q).
135
136
137 %Choix de la règle à appliquer avec poids
138 %E représente l'équation à résoudre
139 %P représente le poids de la règle
140
141 % clash > check > rename > simplify > orient > expand > decompose
142 choix_reduit_1(E, R, P):- regle(E, clear), P=8, R='clear', !;
143 regle(E, clash), P=7, R='clash', !;
144 regle(E, check), P=6, R='check', !;
145 regle(E, rename), P=5, R='rename', !;
146 regle(E, simplify), P=4, R='simplify', !;
147 regle(E, orient), P=3, R='orient', !;
148 regle(E, expand), P=2, R='expand', !;
```

```
149 regle(E, decompose), P=1, R='decompose', !.
150
151 % clash > check > decompose > orient > rename > simplify > expand
152 choix_reduit_2(E, R, P):- regle(E, clear), P=8, R='clear', !;
153 regle(E, clash), P=7, R='clash', !;
154 regle(E, check), P=6, R='check', !;
155 regle(E, decompose), P=5, R='decompose', !;
156 regle(E, orient), P=4, R='orient', !;
157 regle(E, rename), P=3, R='rename', !;
158 regle(E, simplify), P=2, R='simplify', !;
159 regle(E, expand), P=1, R='expand', !.
160
161 %Comparaison des equations pour le choix pondéré 1
162 %Si les deux équations sont de poids différents, on choisit celle qui a le plus
    grand poids
163 %Si les deux équations sont de poids identiques, on choisit la première
164 compare_equation_1(E1, E2, E, O):- choix_reduit_1(E1, _, P1), choix_reduit_1(E2
    , _, P2), P1>=P2, E=E1, O=[E2], !; E=E2, O=[E1].
165
166 %Comparaison des equations pour le choix pondéré 2
167 %Si les deux équations sont de poids différents, on choisit celle qui a le plus
    grand poids
168 %Si les deux équations sont de poids identiques, on choisit la première
169 compare_equation_2(E1, E2, E, O):- choix_reduit_2(E1, _, P1), choix_reduit_2(E2
    , _, P2), P1>=P2, E=E1, O=[E2], !; E=E2, O=[E1].
170
171 %Parcour de la liste des équations pour le choix pondéré 1
172 %Si la liste à un seul élément, on compare les équations
173 %Si la liste à plusieurs éléments, on compare les équations et on parcour la
    liste jusqu'à la fin
174 parcours_liste_1([E1|[]], E2, E, L) :- compare_equation_1(E1, E2, E, L), !.
175 parcours_liste_1([E1|P], E2, E, L):- parcours_liste_1(P, E2, E3, L2),
    compare_equation_1(E1, E3, E4, C), E=E4, append(C, L2, L), !.
176
177
178 %Parcour de la liste des équations pour le choix pondéré 2
179 %Si la liste à un seul élément, on compare les équations
180 %Si la liste à plusieurs éléments, on compare les équations et on parcour la
    liste jusqu'à la fin
181 parcours_liste_2([E1|[]], E2, E, L) :- compare_equation_2(E1, E2, E, L), !.
182 parcours_liste_2([E1|P], E2, E, L):- parcours_liste_2(P, E2, E3, L2),
    compare_equation_2(E1, E3, E4, C), E=E4, append(C, L2, L), !.
183
184
185 % Prédicats d'affichage fournis
186
187 % set_echo: ce prédicat active l'affichage par le prédicat echo
188 set_echo :- assert(echo_on).
189
190 % clr_echo: ce prédicat inhibe l'affichage par le prédicat echo
191 clr_echo :- retractall(echo_on).
192
193 % echo(T): si le flag echo_on est positionné, echo(T) affiche le terme T
194 %          sinon, echo(T) réussit simplement en ne faisant rien.
195 echo(T) :- echo_on, !, write(T).
196 echo(_).
197
```

```
198 % trace_unif(P, S): trace_unif(P, S) affiche les étapes de la résolution de la
    liste d'équations P avec la stratégie S
199 trace_unif(P, S) :- set_echo, echo('unifie('), echo(P), echo(').\n'), unifie(P,
    S), clr_echo.
200
201 % unif(P, S): unif(P, S) indique si la liste d'équations P avec la stratégie S
    peut être résolue
202 unif(P, S) :- clr_echo, unifie(P, S), !.
203
204
205 main :- write("Projet prolog Martelli-Montanari (Briot, Brancati)\n
206 Pour lancer le programme vous avez le choix entre differentes commandes : \n
207 - trace_unif([votre formule], strategie possible) : Affiche les traces d'
    execution de l'unification
208 - unif([votre formule], strategie possible) : Affiche seulement le resultat de l
    'unification \n
209 Les differentes strategies possibles sont :\n
210 - choix_premier : choisi toujours la premiere equation de la liste
211 - choix_pondere_1 : choisi les formules selon un poids defini (clash > check >
    rename > simplify > orient > expand > decompose)
212 - choix_pondere_2 : choisi les formules selon un poids defini (clash > check >
    decompose > orient > rename > simplify > expand)
213 - choix_aleatoire : choisi les equations de facon aleatoire").
214
215 % Lance le programme
216 :- main.
```