Université de Lorraine

Faculté de Science et Technologie

M1 Informatique

Martelli Montanari

Algorithme d'unification

Projet de LMC

Auteurs :
Aurélien Thirion
Nicolas Blin

13 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 de 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 : **operateur.pl** , **predicatsRelais.pl** et **reglesTest.pl**.

operateur.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 nécessaire pour appliquer la réduction (prédicat regle(E, R)).

Question 1

Mise en place pour l'unification
On peut voir ici les instructions pour l'ouverture des différents fichiers.

```
:-
    [operateurs],
    [predicatsRelais],
    [reglesTest].
```

Listing 1 – Ouverture des fichiers dans *main.pl*

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

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

Listing 2 – regle 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*.

```
regle(E, rename):-
    splitEquation(E,X,T),
    var(T),
    var(X).
```

Listing 3 – regle 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.

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

Listing 4 – splitEquation 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.

```
occur_check(V,T):-
    compound(T),
    var(V),
    contains_var(V,T).
```

Listing 5 – occur_check dans *main.pl*

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

```
reduit(rename, E, P, Q):-
    splitEquation(E,X,T),
    X = T,
    P = [_|Q].
```

Listing 6 – reduit dans *main.pl*

Question 2

Stratégies
On s'intéresse désormais à la rapidité de l'exécution de l'algorithme d'unification. Pour cela, on vas mettre en place plusieurs strarégies.
La Première stratégie consiste à choisir la première équation de la liste d'équations et d'essayer de lui appliquer une régle. Cette Stratégie correspond à la méthode de résolution qui à etait mise en place dans la question 1. Elle est définit par le prédicat suivant :

```
1  % unification choix premier
2  unifie(P, choix_premier) :-
3      choix_premier(P, _, _, _),
4      !.
5
6      choix_premier(P, _, _, _) :-
7      unifie(P, regle),
8      !.
```

Listing 7 – Stratégie choix premier dans *main.pl*

La Deuxième stratégie consiste à appliquer les régles de transformation dans un certaine ordre. On cherche à appliquer certaines régles en priorité par rapport à d'autres. On essaie ainsi d'appliquer des régles sur l'ensemble du système d'équations avant d'en essayer d'autres. En d'autres termes, on essaie de trouver dans le systèmes d'équations une équation ou l'on peut appliquer notre régle au lieu de chercher une régle à appliquer à une équation.
On définit la priorité dans laquelle appliquer les régles comme suit :

1. Clash, Check
2. Rename, Simplify
3. Orient
4. Decompose
5. Expand

Par exemple, appliquer les régles Clash et Check en priorité permet de stopper l'éxécution plus rapidement si le système d'équation n'est pas unifiable.

```
1  % unification choix pondere
2  unifie(P, choix_pondere) :-
3      choix_pondere(P, P, _, 1),
4      !.
5
6  %%%% clash, check
7
8  %clash
9  choix_pondere(_, [Head|_], _, 1):-
10     regle(Head , clash),
11     !,
12     reduit(clash, Head, P, _),
13     !.
14
15 %check
16 choix_pondere(_, [Head|_], _, 1):-
17     regle(Head, check),
18     !,
```

```
    reduit(check, Head, P, _),
    !.

% regles non applicables dans le systeme d'equations
choix_pondere(P, [Head|Tail], _, 1):-
    \+regle(Head, clash),
    \+regle(Head, check),
    !,
    choix_pondere(P, Tail, _, 1),     % on essaye d'appliquer les transformations de niveau 1
    !.                                % sur l'equation suivante
```

Listing 8 – Stratégie choix pondere dans *main.pl* avec les régles de poids 1

```
choix_pondere(P, [], _, 2):-
    choix_pondere(P, P, _, 2),
    !.
```

Listing 9 – Stratégie choix pondere dans *main.pl*, passage aux régles de poids 1

La Troisième stratégie consiste à sélectionner une équation au hasard dans le système d'èquations. On tente ensuite d'appliquer toutes les transformations sur cette équation.

```
%unification choix aleatoire
unifie(P, choix_aleatoire) :-
    choix_equation_aleatoire(P,_,_,_),
    !.

choix_equation_aleatoire([],_,_,_):- !.

choix_equation_aleatoire(P,_,_,_):-
    random_member(E, P),                    % on choisit aleatoirement une equation
    deleteEquation(P, E, ListTemp),          % on supprime cette equation de la liste
    reduit_random([E|ListTemp], E, Q, regle),% on essaie d'appliquer les transformations E, nouvelle liste = Q
    choix_equation_aleatoire(Q,_,_,_).

reduit_random(ListTemp2, E, Q, regle) :- regle(E, rename), !, reduit(rename, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, simplify), !, reduit(simplify, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, expand), !, reduit(expand, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, check), !, reduit(check, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, orient), !, reduit(orient, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, decompose), !, reduit(decompose, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, clash), !, reduit(clash, E, ListTemp2, Q).
```

Listing 10 – Stratégie choix aléatoire dans *main.pl*

Comparaison des Stratégies :

Grace à ces exemples, on peut constater que le choix pondere permet de remarquer rapidement si une unification est possible ou non :

```
?- trace_unif([c ?= Z, E?= Z, f(X,Y) ?= f(U,V,W)], choix_premier).
    system: [c?=_G1732,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    orient: c?=_G1732
    system: [_G1732?=c,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    simplify: _G1732?=c
    system: [_G1737?=c,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    simplify: _G1737?=c
    system: [f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    clash: f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)
    [c?=_G1732,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    No
```

Listing 11 – Stratégie choix premier dans *main.pl*

```
?- trace_unif([c ?= Z, E?= Z, f(X,Y) ?= f(U,V,W)], choix_pondere).
    system: [c?=_G1732,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    clash: f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)
    [c?=_G1732,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    No
```

Listing 12 – Stratégie choix pondere dans *main.pl*

La Troisième stratégie permet en général de faire mieux que la stratégie choix premier :

```
?- trace_unif([c ?= Z, E?= Z, f(X,Y) ?= f(U,V,W)], choix_aleatoire).
    system: [f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748),c?=_G1732,_G1737?=_G1732]
    clash: f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)
    [c?=_G1732,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    No

?- trace_unif([c ?= Z, E?= Z, f(X,Y) ?= f(U,V,W)], choix_aleatoire).
    system: [c?=_G1732,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    orient: c?=_G1732
    system: [_G1737?=_G1732,_G1732?=c,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    rename: _G1737?=_G1732
    system: [_G1732?=c,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    simplify: _G1732?=c
    system: [f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    clash: f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)
    [c?=_G1732,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    No

?- trace_unif([c ?= Z, E?= Z, f(X,Y) ?= f(U,V,W)], choix_aleatoire).
    system: [_G1737?=_G1732,c?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    rename: _G1737?=_G1732
    system: [f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748),c?=_G1732]
    clash: f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)
    [c?=_G1732,_G1737?=_G1732,f(_G1743,_G1744)?=f(_G1746,_G1747,_G1748)]
    No
```

Listing 13 – Stratégie choix pondere dans *main.pl*

Question 3

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

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

Listing 14 – 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('\tYes'),
5      !;
6      echo('\t'),
7      echo(P),
8      echo('\n'),
9      echo('\tNo')).
```

Listing 15 – 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 reduit(rename, E, P, Q):-
2     echo('\tsystem: '),echo(P),nl,
3     echo('\trename: '),echo(E),nl,
4     splitEquation(E,X,T),
5     X = T,
6     P = [_|Q].
```

Listing 16 – Les echos dans *main.pl*

Tests

Tests vus en cours
On a décidé de présenter une liste de tests présents dans le cours pour illustrer la résolution de l'algorithme implanté.
(Un exemple qui devrait réussir)

```
?- trace_unif([f(X,a) ?= f(g(Y),Y)],choix_pondere).
    system: [f(_384,a)?=f(g(_390),_390)]
    decompose: f(_384,a)?=f(g(_390),_390)
    system: [a?=_390,_384?=g(_390)]
    orient: a?=_390
    system: [_390?=a,_384?=g(_390)]
    simplify: _390?=a
    system: [_384?=g(a)]
    expand: _384?=g(a)

    Yes
X = g(a),
Y = a.
```

Listing 17 – Des tests de cours

(Un exemple qui devrait échouer)

```
?- trace_unif([f(b,a) ?= f(g(Y),Y)],choix_pondere).
    system: [f(b,a)?=f(g(_390),_390)]
    decompose: f(b,a)?=f(g(_390),_390)
    system: [a?=_390,b?=g(_390)]
    orient: a?=_390
    system: [_390?=a,b?=g(_390)]
    simplify: _390?=a
    [f(b,a)?=f(g(_390),_390)]

    No
```

Listing 18 – Des tests de cours

(Un exemple qui devrait échouer)

```
?- trace_unif([f(X,X) ?= f(g(Y),Y)],choix_pondere).
    system: [f(_384,_384)?=f(g(_390),_390)]
    decompose: f(_384,_384)?=f(g(_390),_390)
    system: [_384?=_390,_384?=g(_390)]
    rename: _384?=_390
    system: [_384?=g(_384)]
    check: _384?=g(_384)
    [f(_384,_384)?=f(g(_390),_390)]

    No
```

Listing 19 – Des tests de cours

(Un exemple qui devrait réussir)

```
?- trace_unif([f(X,Y) ?= f(Y,X)],choix_pondere).
    system: [f(_384,_386)?=f(_386,_384)]
    decompose: f(_384,_386)?=f(_386,_384)
    system: [_384?=_386,_386?=_384]
    rename: _384?=_386
    system: [_384?=_384]
    rename: _384?=_384

    Yes
X = Y.
```

Listing 20 – Des tests de cours

(Un exemple qui devrait échouer)

```
?- trace_unif([f(X,Y) ?= f(U,V,W)],choix_pondere).
    system: [f(_384,_386)?=f(_390,_392,_394)]
    clash: f(_384,_386)?=f(_390,_392,_394)
    [f(_384,_386)?=f(_390,_392,_394)]

    No
```

Listing 21 – Des tests de cours

Code source

Dans operateurs

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

% Predicats d'affichage fournis

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

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

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

echo(T) :-
    echo_on,
    !,
    write(T).

echo(_).
```

Listing 22 – *operateurs.pl*

Dans predicatsRelais

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

Listing 23 – *predicatsRelais.pl*

Dans reglesTest

```
regle(E, rename):-
    splitEquation(E,X,T),
    var(T),
    var(X).

regle(E, simplify):-
    splitEquation(E,X,T),
    atomic(T),
    var(X).

regle(E, orient):-
    splitEquation(E,T,X),
```

```
    var(X),
    nonvar(T).

regle(E, check):-
    splitEquation(E,X,T),
    var(X),
    not(X==T),
    occur_check(X,T).

regle(E, expand):-
    splitEquation(E,X,T),
    var(X),
    not(atomic(T)),
    nonvar(T),
    not(occur_check(X,T)).

regle(E, decompose):-
    splitEquation(E,S,T),
    compound(S),
    compound(T),
    functor(S,NameS,ArityS),
    functor(T,NameT,ArityT),
    NameS == NameT,
    ArityS == ArityT.

regle(S ?= T, clash):-
    compound(S),
    compound(T),
    functor(S,NameS,ArityS),
    functor(T,NameT,ArityT),
    \+((NameS == NameT, ArityS == ArityT)).
```

Listing 24 – *reglesTest.pl*

Dans le main

```
% Ajout d'autres fichiers pour simplier le code principal
:-
    [operateurs], % ?= , echo
    [predicatsRelais], % splitEquation
    [reglesTest]. % test de validite sur chaque regle

% Predicats

% Occur_check
occur_check(V,T):-
    compound(T),
    var(V),
    contains_var(V,T).

% unif
unif(P,S) :-
    clr_echo,
    unifie(P,S).

% trace_unif
trace_unif(P,S) :-
    set_echo,
    (unifie(P,S),
     echo('\tYes'),
     !;
     echo('\t'),
     echo(P),
     echo('\n'),
     echo('\tNo')).
```

```
% unification
unifie([], _) :- !.
unifie([]) :- !.

% unification choix premier
unifie(P, choix_premier) :-
    choix_premier(P, _, _, _),
    !.

%unification choix pondere
unifie(P, choix_pondere) :-
    choix_pondere(P, P, _, 1),
    !.

%unification choix aleatoire
unifie(P, choix_aleatoire) :-
    choix_equation_aleatoire(P,_,_,_),
    !.

unifie(P, regle):- unifie(P, rename).
unifie(P, regle):- unifie(P, simplify).
unifie(P, regle):- unifie(P, expand).
unifie(P, regle):- unifie(P, check).
unifie(P, regle):- unifie(P, orient).
unifie(P, regle):- unifie(P, decompose).
unifie(P, regle):- unifie(P, clash).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Fonction qui supprime l'equation du systeme d'equation
deleteEquation([], _, []):- !.          % si liste vide, on renvoie une liste vide
deleteEquation([Element | Tail], E, [Element| Tail2]):-
    not(Element == E),
    deleteEquation(Tail, E, Tail2),!.

deleteEquation([E | Tail], E, L):-
    deleteEquation(Tail, E, L),!.    % on supprime l'element E de la liste

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

choix_premier(P, _, _, _):-
    unifie(P, regle),
    !.

%%%%%%%%

choix_equation_aleatoire([],_,_,_):- !.

choix_equation_aleatoire(P,_,_,_):-
    random_member(E, P),              % on choisit aleatoirement une equation
    deleteEquation(P, E, ListTemp),      % on supprime cette equation de la liste
    reduit_random([E|ListTemp], E, Q, regle), % on essaie d'appliquer les transformations E, nouvelle liste = Q
    choix_equation_aleatoire(Q,_,_,_).

reduit_random(ListTemp2, E, Q, regle) :- regle(E, rename), !, reduit(rename, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, simplify), !, reduit(simplify, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, expand), !, reduit(expand, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, check), !, reduit(check, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, orient), !, reduit(orient, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, decompose), !, reduit(decompose, E, ListTemp2, Q).
reduit_random(ListTemp2, E, Q, regle) :- regle(E, clash), !, reduit(clash, E, ListTemp2, Q).

%%%%%%%%
```

```
95      % niveau 1: clash, check
96      % niveau 2: rename, simplify
97      % niveau 3: orient
98      % niveau 4: decompose
99      % niveau 5: expand
100
101 choix_pondere([], _, _, _):-
102     true.
103
104 choix_pondere(P, [], _, 1):-      % passage au niveau 2
105     choix_pondere(P, P, _, 2),
106     !.
107
108 choix_pondere(P, [], _, 2):-      % passage au niveau 3
109     choix_pondere(P, P, _, 3),
110     !.
111
112 choix_pondere(P, [], _, 3):-      % passage au niveau 4
113     choix_pondere(P, P, _, 4),
114     !.
115
116 choix_pondere(P, [], _, 4):-      % passage au niveau 5
117     choix_pondere(P, P, _, 5),
118     !.
119
120 % on a applique toutes les regles sans succes, echec de l'unification
121 choix_pondere(_, [], _, 5):-
122     fail,
123     !.
124
125 %%%%%%%%
126
127 %%%% clash, check (niveau 1)
128
129 %clash
130 choix_pondere(P, [Head|_], _, 1):-
131     regle(Head , clash),
132     !,
133     reduit(clash, Head, P, _),
134     !.
135
136 %check
137 choix_pondere(P, [Head|_], _, 1):-
138     regle(Head, check),
139     !,
140     reduit(check, Head, P, _),
141     !.
142
143 % regles non applicables dans le systeme d'equations
144 choix_pondere(P, [Head|Tail], _, 1):-
145     \+regle(Head, clash),
146     \+regle(Head, check),
147     !,
148     choix_pondere(P, Tail, _, 1),    % on essaye d'appliquer les transformations de niveau 1
149     !.                 % sur l'equation suivante
150
151 %%%% rename, simplify (niveau 2)
152
153 % rename
154 choix_pondere(P, [Head|_], _, 2):-
155     regle(Head, rename),
156     !,
157     deleteEquation(P, Head, ListTemp), % on supprime l'equation E du systeme d'equation
158     reduit(rename, Head, [Head|ListTemp], Q),
159     choix_pondere(Q, Q, _, 1),    % on applique la strategie choix_pondere sur le nouveau
160     !.                 % systeme d'equation
```

```
161
162 % simplify
163 choix_pondere(P, [Head|_], _, 2):-
164     regle(Head, simplify),
165     !,
166     deleteEquation(P, Head, ListTemp),
167     reduit(simplify, Head, [Head|ListTemp], Q),
168     choix_pondere(Q, Q, _, 1),
169     !.
170
171 % regles non applicables dans le systeme d'equations
172 choix_pondere(P, [Head|Tail], _, 2):-
173     \+regle(Head, rename),
174     \+regle(Head, simplify),
175     !,
176     choix_pondere(P, Tail, _, 2),
177     !.
178
179 %%%% orient (niveau 3)
180
181 % orient
182 choix_pondere(P, [Head|_], _, 3):-
183     regle(Head, orient),
184     !,
185     deleteEquation(P, Head, ListTemp),
186     reduit(orient, Head, [Head|ListTemp], Q),
187     choix_pondere(Q, Q, _, 1),
188     !.
189
190 % regles non applicables dans le systeme d'equations
191 choix_pondere(P, [Head|Tail], _, 3):-
192     \+regle(Head, orient),
193     !,
194     choix_pondere(P, Tail, _, 3),
195     !.
196
197 %%%% decompose (niveau 4)
198
199 % decompose
200 choix_pondere(P, [Head|_], _, 4):-
201     regle(Head, decompose),
202     !,
203     deleteEquation(P, Head, ListTemp),
204     reduit(decompose, Head, [Head|ListTemp], Q),
205     choix_pondere(Q, Q, _, 1),
206     !.
207
208 % regles non applicables dans le systeme d'equations
209 choix_pondere(P, [Head|Tail], _, 4):-
210     \+regle(Head, decompose),
211     !,
212     choix_pondere(P, Tail, _, 4),
213     !.
214
215 %%%% expand (niveau 5)
216
217 % expand
218 choix_pondere(P, [Head|_], _, 5):-
219     regle(Head, expand),
220     !,
221     deleteEquation(P, Head, ListTemp),
222     reduit(expand, Head, [Head|ListTemp], Q),
223     choix_pondere(Q, Q, _, 1),
224     !.
225
226 % regles non applicables dans le systeme d'equations
```

```
choix_pondere(P, [Head|Tail], _, 5):-
    \+regle(Head, expand),
    !,
    choix_pondere(P, Tail, _, 5),
    !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

unifie(P, rename) :- % Placer dans Q le resultat de l'unification avec la transformation rename
    P = [E |_], % Placer dans E la tete du systeme d'equation P
    regle(E, rename), % Test de l'applicabilite de la regle rename sur l'equation E
    reduit(rename, E, P, Q), % Application de la regle
    unifie(Q, regle),!. % Appel recursif sur l'unification de Q avec une nouvelle regle


unifie(P, simplify):- % Meme raisonnement que precedemment
    P = [E |_],
    regle(E, simplify),
    reduit(simplify, E, P, Q),
    unifie(Q, regle),!.

unifie(P, expand):-
    P = [E |_],
    regle(E, expand),
    reduit(expand, E, P, Q),
    unifie(Q, regle),!.

unifie(P, check):-
    P = [E |_],
    regle(E, check),
    reduit(check, E, P, Q),
    unifie(Q, regle),!.

unifie(P, orient):-
    P = [E |_],
    regle(E, orient),
    reduit(orient, E, P, Q),
    unifie(Q, regle),!.

unifie(P, decompose):-
    P = [E |_],
    regle(E, decompose),
    reduit(decompose, E, P, Q),
    unifie(Q, regle),!.

unifie(P, clash):-
    P = [E |_],
    regle(E, clash),
    reduit(clash, E, P, Q),
    unifie(Q, regle),!.


% Transformation du systeme d'equations P en un systeme d'equations Q par application de la regle de transformation a l'equation E


% reduit sur regle decompose
reduit(decompose, E, P, Q):-
    echo('\tsystem: '),echo(P),nl, % Affichage des etapes pour le trace_unif
    echo('\tdecompose: '),echo(E),nl,
    splitEquation(E,X,T), % Separe E en X et T avec comme separateur ?=
    functor(X,_,ArityX), % Recuperation de l'arite de X
    functor(T,_,_),
    P = [_|Tail], % Recup de la queue de la liste P dans Tail
    repet(X,T,ArityX,Tail,Q). % Boucle iterative pour verifier l' unification sur tous les arguments des fonctions

```

```
292 repet(_,_,0,T,Q):- Q = T, !. % Arret du predicat repet et affectation du resultat dans Q
293 repet(X,T,N,Tail,Q) :-
294     N > 0, % Condition d'arret
295     arg(N,X,ValX), % Recuperer l'argument a l'indice N dans X et le mettre dans ValX
296     arg(N,T,ValT),
297     Var = [ValX?=ValT|Tail], % Var va desormais contenir ValX ?= ValT en plus dans la liste Tail
298     N1 is N - 1, % Decrementation de la boucle
299     repet(X,T,N1,Var,Q).
300
301 reduit(rename, E, P, Q):-
302     echo('\tsystem: '),echo(P),nl,
303     echo('\trename: '),echo(E),nl,
304     splitEquation(E,X,T),
305     X = T,
306     P = [_|Q].
307
308 reduit(simplify, E, P, Q):-
309     echo('\tsystem: '),echo(P),nl,
310     echo('\tsimplify: '),echo(E),nl,
311     splitEquation(E,X,T),
312     X = T,
313     P = [_|Q].
314
315 reduit(expand, E, P, Q):-
316     echo('\tsystem: '),echo(P),nl,
317     echo('\texpand: '),echo(E),nl,
318     splitEquation(E,X,T),
319     X = T,
320     P = [_|Q].
321
322 reduit(check, E, P, _):-
323     echo('\tsystem: '),echo(P),nl,
324     echo('\tcheck: '),echo(E),nl,
325     fail,
326     !.
327
328 reduit(orient, E, P, Q):-
329     echo('\tsystem: '),echo(P),nl,
330     echo('\torient: '),echo(E),nl,
331     splitEquation(E,X,T),
332     P = [_|Tail],
333     Q = [T ?= X | Tail].
334
335
336 reduit(clash, E, P, _):-
337     echo('\tsystem: '),echo(P),nl,
338     echo('\tclash: '),echo(E),nl,
339     fail,
340     !.
```

Listing 25 – *main.pl*