0.1 Annexes

Les sources de ce projet sont divisees en 2 fichiers. Le fichier principal est le fichier *main.pl*, il contient le programme demande. Le second fichier est le fichier *tests.pl* qui contient les tests unitaires que nous avons crees pour s'assurer que notre programme faisait bien le travail voulu, notamment les predicats *regle*, *reduit* et *unifie*. Les tests se lance avec le predicat *run_tests*. . Les sources sont aussi consultables en ligne a l'adresse `www.github.com/romainduc421/swipl-lmc-MartelliMontanari/` .

0.1.1 projet

```
% Definition de l'operateur "?=".
:- 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(_).

:- set_echo.

% Retire les warnings
:- style_check(-singleton).

% +---------------------------------------------------------------------+
%%%%%Question 1
% +---------------------------------------------------------------------+

%%regle(E, R) : Determine la regle de transformation R qui s'applique a l'equation E.
%%------ Def des conditions sur les regles ------%

% Regle clean qui permet d'enlever une equation composee
% des deux memes termes, telles que X ?= X, a ?= a, f(a) ?= f(a)
%
% Ce predicat ne figure pas dans les regles de bases, mais
% a ete ajoute suite a la decouverte des cas presentes.
regle(X?=T, clean) :- X == T,!.

/*
 * Renommage d une variable
 * Regle rename : renvoie true si X et T sont des variables.
 * E : equation donnee.
 * R : regle rename.
 */
regle(X?=T,rename) :- var(X), var(T), !.

/*
 * Simplification de constante
 * Regle simplify : renvoie true si X est une variable et A une
 * constante.
 * E : equation donnee.
 * R : regle simplify.
 */
regle(X?=T,simplify) :- var(X), atomic(T), !.

/*
 * Unification d'une variable avec un terme compose
```

```
 * Regle expand : renvoie true si X est une variable, T un terme
 * compose et si X n'est pas dans T.
 * E : equation donnee.
 * R : regle expand.
 */
regle(X?=T,expand) :- var(X), compound(T), \+occur_check(X,T), !.

/*
 * Verification de presence d occurrence
 * Regle check : renvoie true si X et T sont differents et si X est dans
 * T.
 * E : equation donnee.
 * R : regle check.
 */
regle(X?=T,check) :- \+X==T, occur_check(X,T), !.

/*
 * Echange
 * Regle orient : renvoie true si T n'est pas une variable et si X en
 * est une.
 * E : equation donnee.
 * R : regle orient.
 */
regle(T?=X,orient) :- nonvar(T), var(X), !.

/*
 * Decomposition de deux fonctions
 * Regle decompose : renvoie true si X et T sont des termes composes et
 * s'ils ont le meme nombre d'arguments et le meme nom.
 * E : equation donnee.
 * R : regle decompose.
 */
regle(X?=T,decompose) :- compound(X), compound(T), functor(X,F1,A1), functor(T,F2,A2), F1==F2, A1==A2, !.

/*
 * Gestion de conflit entre deux fonctions
 * Regle clash : renvoie true si X et T sont des termes composes (n'ont pas le meme symbole) et
 * s'ils n'ont pas le meme nombre d'arguments.
 * E : equation donnee.
 * R : regle clash.
 */
regle(X?=T,clash) :- compound(X), compound(T), functor(X,N,A), functor(T,M,B), not( ( N==M , A==B ) ), !.

% occur_check(V,T): teste si la variable V apparait dans le terme T
occur_check(V,T) :- var(V), compound(T), arg(_,T,X), compound(X), occur_check(V,X), !;
          var(V), compound(T), arg(_,T,X), V==X, !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicats annexes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% reduit(R,E,P,Q) : transforme le systeme d'equations P en le systeme d'equations Q par application de la regle de transformation R a l'equation E
% E est represente par X ?= T.

%Reduit au silence les warnings
:- discontiguous reduit/4.

% Predicat reduit pour la regle clean enlever une equation telle que X ?= X, ou a ?= a.
reduit(clean, _, P, Q) :- Q = P, !.

% Predicat reduit pour la regle rename
reduit(rename, X ?= T, P, Q) :-
  elimination(X ?= T, P,Q),
  !.
```

```
119
120 % Predicat reduit pour la regle expand
121 reduit(expand, X ?= T, P, Q) :-
122   elimination(X ?= T, P,Q),
123   !.
124
125
126 % Predicat reduit pour la regle simplify
127 reduit(simplify, X ?= T, P, Q) :-
128   elimination(X ?= T, P, Q),
129   !.
130
131 % Predicat elimination permettant d appliquer l unification necessaire
132 % aux regles rename, expand, simplify
133 elimination(X ?= T, P, Q) :-
134   X = T, % Unification avec la nouvelle valeur de X
135   Q = P,  % Q devient le reste du programme
136   !.
137
138 % Predicat reduit permettant d'appliquer la regle de decomposition de deux fonctions sur l equation E
139 reduit(decompose, Fonct1?= Fonct2, P, Q) :-
140   functor(Fonct1, _, A),  % recuperer le nombre d'arguments
141   decompose(Fonct1, Fonct2, A, Liste), % recuperer les nouvelles eq
142   append(Liste,P,Q), % ajout des eq dans le prog P
143   !.
144
145 % Predicat de decomposition, cas ou l'argument des 2 fct
146 % parcourues est 0 (arret de la recursion)
147 decompose(_,_,0,_) :-
148   !.
149 % Predicat de decomposition, on prend deux fct et on recupere le
150 % ieme argument afin d ajouter l equation Arg1 ?= Arg2 au programme
151 decompose(Fonct1, Fonct2, A, Liste) :-
152   % on decremente le no de l'argument parcouru
153   New is A - 1,
154   % ajout de l'equation liee au (i-1) -ieme argument
155   decompose(Fonct1, Fonct2, New, Res),
156   % obtention de l'argument courant pour les deux fct
157   arg(A, Fonct1, Arg1),
158   arg(A, Fonct2, Arg2),
159   % ajout de l'eq arg1 ?= arg2
160   append(Res, [Arg1 ?= Arg2], Liste),
161   !.
162
163 % Predicat reduit pour la regle orient, le predicat prend l equation E et l inverse
164 % puis l ajoute au programme P, le resultat est alors stocke dans Q
165 reduit(orient, T ?= X, P, Q) :-
166   % ajout de l'equation inversee dans P
167   append([X ?= T], P, Q), !.
168
169 % facultatifs (systeme dequation est incorrect)
170 %occurence
171 reduit(check, _, _, bottom) :- fail.
172 %gestion de conflits
173 reduit(clash, _, _, bottom) :- fail.
174
175 % Predicat unifie (P)
176 % % Unifie sans strategie (Question 1)
177 unifie([]) :- echo("\nUnification terminee."), echo("Resultat: \n\n"),!.
178
179 unifie([X|P]) :-
180   aff_syst([X|P]),
181   regle(X,R),
182   aff_regle(R,X),
183   reduit(R,X,P,Q), !, unifie(Q).
184
```

```
185
186 % +---------------------------------------------------------------+
187 %%%%%Question 2
188 % +---------------------------------------------------------------+
189
190 % Predicats annexes
191 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
192
193 % Permet d'extraire un element d'une liste .
194 extraction( _, [], []).
195 extraction( R, [R|T], T).
196 extraction( R, [H|T], [H|T2]) :- H \= R, extraction( R, T, T2).
197
198 % Permet de tester l' applicabilite des regles de la liste "Regles" sur l'equation X.
199 % Cherche la regle R que l'on peut appliquer dans une liste de regles Regles a une liste d'equations d'
      unification X
200 % R1 est le premier element de la liste "Regles"
201 regle_applicable(X, [R1|Regles], R) :- regle(X,R1), R = R1, !.
202 regle_applicable(X, [R1|Regles], R) :- regle_applicable(X, Regles, R), !.
203
204 % Permet de choisir sur quelle equation appliquer les regles dans la liste Regles
205 choix_equation([X|P],Q,E,[R1|Regles],R):-
206   regle_applicable(X, [R1|Regles], R), E = X, !.
207 choix_equation([X|P], Q, E,[R1|Regles],R) :-
208   choix_equation(P, Q, E,[R1|Regles],R), !.
209
210
211 choix_premier([X|P],Q,E,R) :- regle(E,R), aff_regle(R,E), !, reduit(R,E,P,Q).
212 choix_dernier(P, L, E, R) :- reverse(P, [E|L]), regle(E, R), !.
213
214 % Liste des "Strategies"
215 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
216 % Choix_pondere_1 (Exemple du sujet)
217 % Poids des regles
218 % on donne maintenant un poids a chaque regle selon le modele suivant :
219 % clean; clash; check > rename; simplify > orient > decompose > expand
220 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
221
222 choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [clean, check, clash], R), !.
223 choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [rename, simplify], R), !.
224 choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [orient], R), !.
225 choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [decompose], R), !.
226 choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [expand], R), !.
227
228 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
229 % Choix_pondere_2
230 % Poids des regles
231 % on donne maintenant un poids a chaque regle selon le 2eme modele suivant :
232 % clash; check > orient > decompose > rename; simplify > expand > clean
233 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
234
235 choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [clash, check], R), !.
236 choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [orient], R), !.
237 choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [decompose], R), !.
238 choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [rename, simplify], R), !.
239 choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [expand], R), !.
240 choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [clean], R), !.
241
242 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
243 % Predicats pour unifier
244
245 % choix_premier, aucun poids sur les regles
246 unifie([],_) :- echo("\nUnification terminee."), echo("Resultat: \n\n"),!.
247 unifie([X|P],choix_premier):- aff_syst([X|P]), choix_premier([X|P],Q,X,_), !, unifie(Q, choix_premier)
      .
248 unifie(P, choix_dernier):-
```

```
  aff_syst(P),choix_dernier(P, P2, E, R), aff_regle(R,E), reduit(R, E, P2,Q), !,unifie(Q,choix_dernier).

% Applique la strategie S pour l'algorithme
unifie(P,S) :-
  aff_syst(P), %On affiche le systeme
  choix(S, P, Q, E, R), %On effectue le choix de la regle a appliquer + sur quelle equation
  aff_regle(R,E), %On affiche la regle utilisee .
  regle(E,R), %On applique la regle.
  extraction(E, P, U), %On extrait l'element
  reduit(R,E,U,Q), %On applique reduit
  unifie(Q, S), %Recursion
  !.

% +---------------------------------------------------------------+
%%%%%Question 3
% +---------------------------------------------------------------+

%appelle unifie apres avoir desactive les affichages
unif(P, S) :- clr_echo, unifie(P, S).

%appelle unifie apres avoir active les affichages
trace_unif(P, S) :- set_echo, unifie(P,S).


% PREDICATS POUR L AFFICHAGE
aff_syst(W) :- echo('\nsystem: '), echo(W), echo('\n').
aff_regle(R,E) :- echo(R),echo(': '),echo(E).

% +---------------------------------------------------------------+
% Lancement du programme
:- initialization
       manual.


manual :- write("Unification Martelli-Montanari\n"),
    write("\n\nUtilisez trace_unif(P,S) pour executer l'algorithme de Martelli-Montanari avec les traces d'execution a chaque etape."),
    write("\nUtilisez unif(P,S) pour executer l'algorithme de Martelli-Montanari sans les traces d'execution."),
    write("\nP est le systeme a unifier. S represente une strategie a employer: choix_premier, choix_pondere_1, choix_pondere_2, choix_dernier."),
    set_echo, !.
```

Listing 1: le fichier main.pl

0.1.2 tests

```
include(main).
:- consult(main).
:- style_check(-singleton).
%test rules

:- begin_tests(regle).
%rename
test(rename) :- regle(X ?= T, R). %R = rename
test(rename, [fail]) :- regle(X ?= a, rename).
test(rename, [fail]) :- regle(X ?= f(a), rename).
test(rename, [fail]) :- regle(X ?= f(X), rename).

%simplify
test(simplify) :- regle(X ?=t, R). %R = simplify
test(simplify) :- regle(X?=a, simplify). %true
test(simplify, [fail]) :- regle(X?=f(a), simplify).
test(simplify, [fail]) :- regle(X ?= f(X), simplify).

%check
```

```
test(check) :- regle(X?=f(X),R). %R = check.
test(check, [fail]):- regle(X ?= a, check).
test(check, [fail]):- regle(X ?= f(a), check).
test(check):- regle(X ?= f(X), check). %true

%orient
test(orient) :- regle(t ?= X, R). %R = orient
test(orient) :- regle(t ?= X, orient). %true
test(orient, [fail]) :- regle(X ?= f(a), orient).
test(orient, [fail]) :- regle(X ?= a, orient).

%decompose
test(decompose) :- regle(f(t) ?= f(X), R). %R = decompose
test(decompose, [fail]) :- regle(f(X) ?= X, decompose).
test(decompose, [fail]) :- regle(f(X) ?= f(X, Y), decompose).
test(decompose) :- regle(f(X) ?= f(a), decompose). %true
test(decompose, [fail]) :- regle(f(X) ?= g(Y), decompose).

%clash
test(clash) :- regle(f(t) ?= g(X, a), R). %R = clash
test(clash, [fail]) :- regle(f(X) ?= X, clash).
test(clash) :- regle(f(X) ?= f(X, Y), clash). %true
test(clash, [fail]):- regle(f(X) ?= f(a), clash).
test(clash) :- regle(f(X) ?= g(Y), clash). %true

%expand
test(expand) :- regle(X?=f(a,b),expand). %true
test(expand, [fail]) :- regle(X?=f(a,b,X),expand).
test(expand, [fail]) :- regle(X?=f(f(a),b,X),expand).
test(expand, [fail]) :- regle(X?=f(f(X,b,a),b,a),expand).
test(expand) :- regle(X?=f(f(Y,b,a),b,a),expand). %true
test(expand, [fail]) :- regle(X?=f(g(X,b,a),b,a),expand).
test(expand, [fail]) :- regle(X?=f(g(h(X),b,a),b,a),expand).
test(expand) :- regle(X?=f(g(h(Y),b,a),b,a),expand). %true

:- end_tests(regle).

%%tests reduit
:- begin_tests(reduit).

%rename
test(rename) :- reduit(rename, X ?= T, [], Q).
/* X = T,
Q = [].
*/
test(rename) :- reduit(rename, X ?= T, [f(X) ?= X], Q).
/* X = T,
Q = [f(T)?=T].
*/

%simplify
test(simplify) :- reduit(simplify, X ?= t, [], Q).
/* X = t,
Q = [].
*/
test(simplify) :- reduit(simplify, X ?= t, [f(X) ?= X], Q).
/* X = t,
Q = [f(t)?=t].
*/

%expand
test(expand) :- reduit(expand, X ?= f(t), [], Q).
/* X = f(t),
Q = [].
*/
test(expand) :- reduit(expand, X ?= f(t), [f(X) ?= X], Q).
```

```
/* X = f(t),
Q = [f(f(t))?=f(t)].
*/

%check
test(check, [fail]) :- reduit(check, X ?= f(t, Y, X, k), [f(X) ?= X], Q).
%Q = bottom.
test(check, [fail]) :- reduit(check, X ?= f(t, Y, X, k), [], Q).
%Q = bottom.

%orient
test(orient) :- reduit(orient, t ?= X, [f(X) ?= X], Q).
%Q = [X?=t, f(X)?=X].
test(orient) :- reduit(orient, t ?= X, [], Q).
%Q = [X?=t].

%clash
test(clash, [fail]) :- reduit(clash, f(t) ?= f(X, m), [], Q).
%Q = bottom.
test(clash, [fail]) :- reduit(clash, f(t) ?= g(X), [f(X) ?= X], Q).
%Q = bottom.
test(clash, [fail]) :- reduit(clash, f(t) ?= g(X), [], Q).
%Q = bottom.

%decompose
test(decompose):- reduit(decompose, f(t) ?= f(X), [f(X) ?= X], Q).
%Q = [t?=X, f(X)?=X].
test(decompose):- reduit(decompose, f(t, C, X) ?= f(X, k, Y), [], Q).
%Q = [t?=X, C?=k, X?=Y].
:- end_tests(reduit).

%% tests unifie
:- begin_tests(unifiebase_).
test(unifie) :- unifie([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(Y)]).
/* trace_unif([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(Y)], choix_premier).

system: [f(_396,_398)?=f(g(_402),h(a)),_402?=f(_398)]
decompose: f(_396,_398)?=f(g(_402),h(a))
system: [_396?=g(_402),_398?=h(a),_402?=f(_398)]
expand: _396?=g(_402)
system: [_398?=h(a),_402?=f(_398)]
expand: _398?=h(a)
system: [_402?=f(h(a))]
expand: _402?=f(h(a))
Unification terminee. Resultat :

X = g(f(h(a))),
Y = h(a),
Z = f(h(a)).
*/

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

:- end_tests(unifiebase_).

/* trace_unif([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(X)], choix_premier).

system: [f(_8,_10)?=f(g(_14),h(a)),_14?=f(_8)]
decompose: f(_8,_10)?=f(g(_14),h(a))
system: [_8?=g(_14),_10?=h(a),_14?=f(_8)]
expand: _8?=g(_14)
system: [_10?=h(a),_14?=f(g(_14))]
expand: _10?=h(a)
system: [_14?=f(g(_14))]
check: _14?=f(g(_14))
system: [_14?=f(g(_14))]
```

```
152 false .
153 */
154
155 :- begin_tests(unifie_).
156 test(unifie, [forall(member(STRATEGY ,[choix_premier, choix_pondere_1, choix_pondere_2]))]) :-
157     unifie([f(X,Y)?= f(g(Z), h(a)), Z ?= f(Y)], STRATEGY).
158 test(unifie, [forall(member(STRATEGY ,[choix_premier, choix_pondere_1, choix_pondere_2])), fail]) :-
159     unifie([f(X,Y)?= f(g(Z), h(a)), Z ?= f(Y), X ?= f(X)], STRATEGY).
160 test(unifie,[forall(member(STRATEGY,[choix_premier, choix_pondere_2]))]) :-
161    unifie([p(g(u,Z),X,Z) ?= p(X,g(Y,Z),b)],STRATEGY).
162
163 test(unifie,[forall(member(STRATEGY,[choix_pondere_1])), fail]) :-
164    unifie([p(g(u,Z),X,Z) ?= p(X,g(Y,Z),b)],STRATEGY).
165 test(unifie,[forall(member(STRATEGY ,[choix_premier, choix_pondere_1,choix_pondere_2])), fail]) :-
166    unifie([p(X,f(X),h(f(X),X)) ?= p(Z,f(f(a)),h(f(g(a,Z)),v))],STRATEGY).
167
168 :- end_tests(unifie_).
```

Listing 2: le fichier test.pl