```
% Definition de l'operateur "?=".
:- op(20,xfy,?=).

% 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.

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

echoln(T) :- echo_on, echo(T), nl.
echoln(_).

:- set_echo.

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

% +------------------------------------------------------------------------+
%          ____                       _   _                __
%         / __ \                     | | (_)              /_ |  _
%        | |  | |  _   _   ___   ___ | |_  _   ___   _ __   | | (_)
%        | |  | | | | | | / _ \ / __|| __|| | / _ \ | '_ \  | |
%        | |__| | | |_| ||  __/ \__ \| |_ | || (_) || | | | | |  _
%         \___\_\  \__,_| \___| |___/ \__||_| \___/ |_| |_| |_| (_)
%
% +------------------------------------------------------------------------+

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

/*
 * Renommage d une variable
 * Regle rename : renvoie true si X et T sont des variables.
 * E : equation donnee.
 * R : regle rename.
 * Rename {x ?= t}∪P′;S -> P′[x/t];S[x/t]∪{x=t} si t est une variable
 */
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.
 * Simplify {x ?= t}∪P′;S -> P′[x/t];S[x/t]∪{x=t} si t est une constante
 */
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.
 * Expand {x ?= t}∪P′;S -> P′[x/t];S[x/t]∪{x=t} si t est composé et x n’apparaît
pas dans t
 */
regle(X?=T,expand) :- var(X), compound(T), \+occur_check(X,T), !.

```

```
68  /*
69   * Verification de presence d occurrence
70   * Regle check : renvoie true si X et T sont differents et si X est dans
71   * T.
72   * E : equation donnee.
73   * R : regle check.
74   * Check {x?=t}∪P′;S->⊥ si x!=t et x apparaît dans t
75   */
76  regle(X?=T,check) :- \+X==T, occur_check(X,T), !.
77
78  /*
79   * Echange
80   * Regle orient : renvoie true si T n'est pas une variable et si X en
81   * est une.
82   * E : equation donnee.
83   * R : regle orient.
84   * Orient {t?=x}∪P′;S->{x=?t}∪P′;S si t n’est pas une variable
85   */
86  regle(T?=X,orient) :- nonvar(T), var(X), !.
87
88  /*
89   * Decomposition de deux fonctions
90   * Regle decompose : renvoie true si X et T sont des termes composes et
91   * s'ils ont le meme nombre d'arguments et le meme nom.
92   * E : equation donnee.
93   * R : regle decompose.
94   * Decompose {f(s,···,s)?=f(t,···,t)}∪P′;S->{s?=t,···,s?=t}∪P′;S
95   */
96  regle(X?=T,decompose) :- compound(X), compound(T), functor(X,F1,A1),
    functor(T,F2,A2), F1==F2, A1==A2, !.
97
98  /*
99   * Gestion de conflit entre deux fonctions
100  * Regle clash : renvoie true si X et T sont des termes composes (n'ont pas le meme
    symbole) et
101  * s'ils n'ont pas le meme nombre d'arguments.
102  * E : equation donnee.
103  * R : regle clash.
104  * Clash {f(s,···,s)?=g(t,···,t)}∪P′;S->⊥ si f!=g ou m!=n
105  */
106 regle(X?=T,clash) :- compound(X), compound(T), functor(X,N,A), functor(T,M,B),
    not( ( N==M , A==B ) ), !.
107
108 % occur_check(V,T): teste si la variable V apparait dans le terme T
109 occur_check(V,T) :- var(V), compound(T), arg(_,T,X), compound(X),
    occur_check(V,X), !;
110                                         var(V), compound(T), arg(_,T,X), V==X, !.
111
112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
113 % Predicats annexes
114 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
115
116 % reduit(R,E,P,Q) : transforme le système d'équations P en le système d'équations Q
    par application de la règle de transformation R à l'équation E
117 % E est représenté par X ?= T.
118
119 % Predicat reduit pour la regle rename
120 reduit(rename, X ?= T, P, Q) :-
121         elimination(X ?= T, P, Q),
122         !.
123
124 % Predicat reduit pour la regle expand
125 reduit(expand, X ?= T, P, Q) :-
126         elimination(X ?= T, P,Q),
127         !.
128
129
130 % Predicat reduit pour la regle simplify
131 reduit(simplify, X ?= T, P, Q) :-
```

```
        elimination(X ?= T, P, Q),
        !.

% Predicat elimination permettant d appliquer l unification necessaire
% aux regles rename, expand, simplify
elimination(X ?= T, P, Q) :-
        X = T,  % Unification avec la nouvelle valeur de X
        Q = P,  % Q devient le reste du programme
        !.

% Predicat reduit permettant d'appliquer la regle de decomposition de deux
fonctions sur l equation E
reduit(decompose, Fonct1?= Fonct2, P, Q) :-
        functor(Fonct1, _, A),          % recuperer le nombre d'arguments
        decompose(Fonct1, Fonct2, A, Liste), % recuperer les nouvelles eq
        append(Liste,P,Q), % ajout des eq dans le prog P
        !.

% Predicat de decomposition, cas où l'argument des 2 fct
% parcourues est 0 (arrêt de la récursion)
decompose(_,_,0,_) :-
        !.
% Predicat de decomposition, on prend deux fct et on recupere le
% ieme argument afin d ajouter l equation Arg1 ?= Arg2 au programme
decompose(Fonct1, Fonct2, A, Liste) :-
        % on decremente le no de l'argument parcouru
        New is A - 1,
        % ajout de l'equation liee au (i-1) -ieme argument
        decompose(Fonct1, Fonct2, New, Res),
        % obtention de l'argument courant pour les deux fct
        arg(A, Fonct1, Arg1),
        arg(A, Fonct2, Arg2),
        % ajout de l'eq arg1 ?= arg2
        append(Res, [Arg1 ?= Arg2], Liste),
        !.

% Predicat reduit pour la regle orient, le predicat prend l equation E et l inverse
% puis l ajoute au programme P, le resultat est alors stocke dans Q
reduit(orient, T ?= X, P, Q) :-
        % ajout de l'equation inversee dans P
        append([X ?= T], P, Q), !.

% facultatifs (systeme dequation est incorrect)
%occurence
reduit(check, _, _, bottom) :- fail.
%gestion de conflits
reduit(clash, _, _, bottom) :- fail.

% Predicat unifie(P)
% % Unifie sans stratégie (Question 1)
unifie([]) :- echo("\nUnification terminee."), echo("Resultat: \n\n"),!.

unifie([X|P]) :-
        aff_syst([X|P]),
        regle(X,R),
        aff_regle(R,X),
        reduit(R,X,P,Q), !, unifie(Q).


% +----------------------------------------------------------------------------+
%
%         ____                                  _     _                      ___
%        / __ \                                | |   (_)                    |__ \    _
%       | |  | |  _   _    ___    ___  | |_    _    ___    _ __         ) |  (_)
%       | |  | | | | | |  / _ \  / __| | __|  | |  / _ \  | '_ \       / /
%       | |__| | | |_| | |  __/  \__ \ | |_   | | | (_) | | | | |     / /_    _
%        \___\_\  \__,_|  \___|  |___/  \__|  |_|  \___/  |_| |_|    |____| (_)
%
% +----------------------------------------------------------------------------+
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicats annexes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Permet d'extraire un element d'une liste.
extraction( _, [], []).
extraction( R, [R|T], T).
extraction( R, [H|T], [H|T2]) :- H \= R, extraction( R, T, T2).

% Permet de tester l'applicabilité des regles de la liste "Regles" sur l'equation X.
% Cherche la regle R que l'on peut appliquer dans une liste de regles Regles a une liste d'equations d'unification X
% R1 est le premier element de la liste "Regles"
regle_applicable(X, [R1|Regles], R) :- regle(X,R1), R = R1, !.
regle_applicable(X, [R1|Regles], R) :- regle_applicable(X, Regles, R), !.

% Permet de choisir sur quelle equation appliquer les regles dans la liste Regles
choix_equation([X|P],Q,E,[R1|Regles],R):-
        regle_applicable(X, [R1|Regles], R), E = X, !.
choix_equation([X|P], Q, E,[R1|Regles],R) :-
        choix_equation(P, Q, E,[R1|Regles],R), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Liste des "Strategies"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choix_pondere_1 (Exemple du sujet)
% Poids des regles
% on donne maintenant un poids à chaque règle selon le modèle suivant :
% clash; check > rename; simplify > orient > decompose > expand
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [check, clash], R), !.
choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [decompose], R), !.
choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [rename, simplify], R), !.
choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [orient], R), !.
choix(choix_pondere_1, P,Q,E,R) :- choix_equation(P, Q, E, [expand], R), !.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choix_pondere_2
% Poids des regles
% on donne maintenant un poids à chaque règle selon le 2eme modèle suivant :
% clash; check > decompose; simplify > orient; expand > rename
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [check, clash], R), !.
choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [decompose, simplify], R), !.
choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [orient,expand], R), !.
choix(choix_pondere_2, P,Q,E,R) :- choix_equation(P, Q, E, [rename], R), !.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicats pour unifier
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% choix_premier, aucun poids sur les regles
unifie([],_) :- echo("\nUnification terminee."), echo("Resultat: \n\n"),!.
unifie([X|P],choix_premier):- unifie([X|P]), !.

% 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
        !.

% +----------------------------------------------------------------------------+
%         ____                         _     _                     ____
%        / __ \                       | |   (_)                   |___ \   _
%       | |  | |  _   _    ___   ___  | |_   _    ___   _ __        __) | (_)
%       | |  | | | | | |  / _ \ / __| | __| | |  / _ \  | '_ \     |__ <
%       | |__| | | |_| | |  __/ \__ \ | |_  | | | (_) | | | | |    ___) |  _
%        \___\_\  \__,_|  \___| |___/  \__| |_|  \___/  |_| |_|   |____/  (_)
%
% +----------------------------------------------------------------------------+

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

%appelle unifie apres avoir active les affichages, affiche "Yes" si on peut unifier "No" sinon (il n'y a donc pas d'echec de la procedure.
%trace_unif(P, S) :- set_echo, unifie(P,S).
trace_unif(P,S) :- set_echo, (unifie(P, S), echo('Yes'), ! ;
                echo('No') ) .

% PREDICATS POUR L AFFICHAGE
aff_syst(W) :- echo('system: '), echoln(W).
aff_regle(R,E) :- echo(R),echo(': '),echoln(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."),
                set_echo, !.
```