Université De Lorraine

Faculté des Sciences
Département d'Informatique

Master I Informatique
Logique et Models de Calcul

Compte redu du TP

Implanter une variante de l'algorithme d'unification de Martelli-Montanari en PROLOG

Réalisé par :

Tarek Mokhtari
Oussama Zekri

14 décembre 2016

Introduction

Le but ce TP s'agit en l'implantation d'une variante de l'algorithme d'unification de Martelli-Montanari en PROLOG.

PROLOG est un langage de programmation logique très bien adapté à notre TP. Nous utiliserons l'implémentation du langage : SWI-Prolog.

Les règles de transformation utilisées par cette variante de l'algorithme de Martelli-Montanari sont les suivantes :

Rename $\{x \stackrel{?}{=} t\} \cup P'; S \rightsquigarrow P'[x/t]; S[x/t] \cup \{x = t\}$ si t est une variable.
Simplify $\{x \stackrel{?}{=} t\} \cup P'; S \rightsquigarrow P'[x/t]; S[x/t] \cup \{x = t\}$ si t est une constante.
Expand $\{x \stackrel{?}{=} t\} \cup P'; S \rightsquigarrow P'[x/t]; S[x/t] \cup \{x = t\}$ si t est composé et x n'apparaît pas dans t.
Check $\{x \stackrel{?}{=} t\} \cup P'; S \rightsquigarrow \perp$ si x $\neq$ t et x apparaît dans t.
Orient ;$\{t \stackrel{?}{=} x\} \cup P'; S \rightsquigarrow$; $\{x \stackrel{?}{=} t\} \cup P'; S$ si t n'est pas une variable.
Decompose $\{f(s_1, \cdots, s_n) \stackrel{?}{=} f(t_1, \cdots, t_n)\} \cup P'; S \rightsquigarrow \{s_1 \stackrel{?}{=} t_1, \cdots, s_n \stackrel{?}{=} t_n\} \cup P'; S$.
Clash $\{f(s_1, \cdots, s_n) \stackrel{?}{=} g(t_1, \cdots, t_m)\} \cup P'; S \rightsquigarrow \perp$ si $f \neq g$ ou $m \neq n$.

L'algorithme prend en entrée un système d'équations (liste d'équations) et tente de trouver un plus grand unificateur (pgu). Un pgu est un ensemble d'équations de la forme $x_i = t_i$ où x_i variable et t_i terme.

L'unification du système P réussit si l'algorithme trouve un pgu.

Conception

1 Question 1

1.1 Le prédicat regle(E,R)

Pour implanter le prédicat `regle(E,R)`, on doit implanter une règle logique pour chaque règle de notre variante de l'algorithme de Martelli-Montanari, Où :

— `R` est une règle parmi : `Rename, Simplify, Expand, Check, Orient, Decompose` et `Clash`.

— `E` est une équation sur laquelle `R` s'applique.

Avant de vérifier si l'équation `E` remplie bien les condition de la règle en question (`R` s'applique sur `E`), on vérifie que `E` est bien une équation bien formée (de la forme X ?=T, où X et T sont des termes quelconques). Pour cela, on a implanter le prédicat `equation`.

Chaque règle est suivie d'un ensemble de questions pour vérifier que le prédicat couvre les différents cas.

1.2 Le prédicat occur_check(V,T)

La façon dont le prédicat `occur_ceck/2` a été implanté l'oblige à parcourir l'arbre qui représente le terme passé en deuxième argument jusqu'aux feuilles, dans l'espoir de trouver une occurrence de la variable passée en premier argument.

Effectivement deux règles ont été implantées ; une qui capture le cas où on est sur un noeud, et l'autre le cas où on est sur une feuille [1].

Une question qui fait appel au prédicat `occur_ceck/2` échoue s'il ne parvient à trouver la variable passée en premier argument sur aucune des feuilles de l'arbre qui représente le terme passé en deuxième argument.

Le prédicat est suivi de quelques questions servant à vérifier s'il couvre les différents cas.

1.3 Le prédicat reduit(R,E,P,Q)

Une ou plusieurs règles logiques ont été implantées pour chaque règle de notre variante de l'algorithme de Martelli-Montanari.

— Pour les règles `rename, simplify` et `expand` le prédicat `reduit/4` a été implanté de la même façon, et suit les étapes suivantes :

1. Parcourir tout le système d'équations par inférence sur P.
2. Ignorer les occurrences de E dans P si elles existent.
3. Unifier les deux arguments de `E` dans le reste du système d'équations.

1. Voir le source, ligne 186.

4. Remplir Q en remontant l'arbre .

— Pour les règles `check` et `clash`, `reduit/4` arrête le parcours de l'arbre construit par prolog et échoue.
— Pour la règle `orient` il suffit juste d'extraire l'équation en question du système d'équations et d'y insérer l'équation réorientée.
— Pour la règle `decompose` on génère deux listes contenant les arguments des deux termes de l'équation en question (à l'aide de l'opérateur `=..`). À partir de ces deux listes on construit une liste d'équations (à l'aide du prédicat `decompose/3`) qu'on insère dans le système d'équations de départ.

Pour implanter le prédicat `reduit/4` pour les deux règles `orient` et `decompose` on a eu besoin d'implanter le prédicat `delete_eq/3`. Ce dernier prend comme arguments une équation (premier argument) et supprime toutes ses occurrences d'un système d'équations (deuxième argument). Le système d'équations résultant peut être récupéré au biais du troisième argument.

Pour implanter le prédicat `reduit/4` pour la règle `decompose` on a eu besoin d'implanter le prédicat `decompose/3` qui prend comme paramètres deux listes de termes (premier et deuxième arguments). `decompose/3` génère dans son troisième argument une nouvelle liste contenant des équations dont le premier et le deuxième opérandes sont des termes pris respectivement de la première et la deuxièmes listes passées comme arguments au prédicat `decompose/3`.

Chaque règle ou groupe de règles est suivi d'un ensemble de questions pour vérifier que le prédicat couvre les différents cas.

1.4 Le prédicat unifie(P)

L'implantation du prédicat `unifie/1` se fait par inférence sur le système d'équations `P` jusqu'à ce qu'il ne reste aucune équation dans ce dernier. Dans ce cas, le prédicat réussit.

`unifie` trouve à l'aide de `regle/2` la règle de notre algorithme d'unification à appliquer sur la première équation du système d'équations passé comme argument. Ensuite, `unifie` applique la réduction à l'aide du prédicat `reduit/4`. Et enfin, `unifie/1` unifie le système résultat de la dernière réduction.

L'affichage du système d'équations courant, et des règles appliquées à chaque étape se fait à l'aide du prédicat `echo/1`. Avant de tenter d'afficher quelque chose, on teste si le prédicat `echo_on/0` a été défini en utilisant le prédicat `current_predicate/1`. L'affichage ne réussit que lorsque `echo_on/0` a bien été défini, ceci se traduit en la disjonction
`not(current_predicate(echo_on/0));echo(_)`

Si `unifie/1` réussit, La solution sera automatiquement affichée par prolog. Ce dernier affiche les assertions faites automatiquement à chaque étape d'unification.

2 Question 2

2.1 Le prédicat choix_premier(P,Q,E,R)

Le prédicat `choix_premier/4` choisi la première équation et la règle correspondante en utilisent simplement le prédicat `regle/2`.

2.2 Le prédicat choix_pondere(P,Q,E,R)

Pour implanter le prédicat `choix_pondere/4` on définit une règle pour chaque pour chaque poids différent affecté aux règles de l'algorithme. Et On ordonne les règles logique par poids de pondération décroissants pour que les règles du poids le plus fort seront privilégiées.

2.3 Le prédicat unifie(P,S)

L'implantation du prédicat `unifie(P,S)` est presque identique à l'implantation du prédicat `unifie(P)`, sauf pour le choix de l'équation en cours qui se fait par l'appel de `call(S,P,Q,E,R)`. ce dernier construit appel au prédicat `S(P,Q,E,R)` où S est le nom de la stratégie passée comme deuxième argument au prédicat `unifie/2`.

3 Question 3

Pour implanter les prédicats `unif(P,S)` et `trace_unif(P,S)`, il suffit simplement de respectivement inhiber la trace d'affichage des règles (`clr_echo`) ou de l'activer (`set_echo`) avant de faire appel au prédicat `unifie(P,S)` avec les mêmes arguments.

Source

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

% Ce prédicat réussit si E est une équation de la forme: X ?= T.
% Où X et T sont des termes quelconques.
equation(E) :-
        % On teste que E n'est pas une variable ou une constante.
        compound(E),
        % On récupère le functor de l'équation (?=) dans F,
        % Et l'arité (nombre d'arguments) de l'équation dans A.
        functor(E,F,A),
        % On vérifie que le nombre d'arguments est bien 2.
        A == 2,
        % On vérifie aussi que le functor est bien ?=.
        F == '?='.
```

```
% equation(X?=Y).        --> true.
% equation(?=(X,T)).     --> true.
% equation(X=Y).         --> false.
% +--------------------------------------------------------------------------+

% Implantation des règles du prédicat: regle(E,R).
regle(E,rename) :-
        % On teste si E (passé en paramètre) est bien une équation.
        equation(E),
        % On récupère le premier argument de l'équation E dans X.
        arg(1,E,X),
        % On récupère le deuxième argument de l'équation E dans T.
        arg(2,E,T),
        % On teste si les deux arguments sont des variables.
        var(X),
        var(T).
% regle(X?=Y,rename).      --> true.
% regle(X?=X,rename).      --> true.
% regle(X?=x,rename).      --> false.
% regle(X?=Z,R).           --> R = rename.

regle(E,simplify) :-
        % On teste si E (passé en paramètre) est bien une équation.
        equation(E),
        % On récupère le premier argument de l'équation E dans X.
        arg(1,E,X),
        % On récupère le deuxième argument de l'équation E dans T.
        arg(2,E,T),
        % On vérifie que le premier argument est bien une variable.
        var(X),
        % On vérifie que le deuxième argument est bien une constante.
        atom(T).
% regle(X?=v,simplify).            --> true.
% regle(X?=Y,simplify).            --> false.
% regle(X?=y(x),simplify).         --> false.
% regle(X?=y,R).                   --> R = simplify.
% X = a, regle(Y?=X,simplify).     --> X = a ?

regle(E,expand) :-
        % On teste si E (passé en paramètre) est bien une équation.
        equation(E),
        % On récupère le premier argument de l'équation E dans X.
        arg(1,E,X),
        % On récupère le deuxième argument de l'équation E dans T.
        arg(2,E,T),
        % On vérifie que le premier argument est bien une variable.
        var(X),
        % On vérifie que le deuxième argument est bien un terme composé.
        compound(T),
```

```
        % On vérifie que X n'apparaît pas dans t.
        not(occur_check(X,T)).
% regle(X?=t(x),expand).    --> true.
% regle(x?=t(x),expand).    --> false.
% regle(X?=t, expand).      --> false.
% regle(X?=t(x),R).         --> R = expand.

regle(E,check) :-
        % On teste si E (passé en paramètre) est bien une équation.
        equation(E),
        % On récupère le premier argument de l'équation E dans X.
        arg(1,E,X),
        % On récupère le deuxième argument de l'équation E dans T.
        arg(2,E,T),
        % On vérifie que X et T ne sont pas identiques.
        not(X == T),
        % On vérifie si la variable X apparaît dans le terme T.
        % occur_check/2 s'occupera de vérifier que X est une variable.
        occur_check(X,T).
% regle(X?=t(X),check).        --> true.
% regle(X?=t,check).        --> false.
% regle(Y?=f(X),check).     --> false.
% regle(X?=X,check).        --> false.
% regle(X?=t(X),R).         --> R = check.

regle(E,orient) :-
        % On teste si E (passé en paramètre) est bien une équation.
        equation(E),
        % On récupère le premier argument de l'équation E dans T.
        arg(1,E,T),
        % On récupère le deuxième argument de l'équation E dans X.
        arg(2,E,X),
        % On vérifie que le premier argument n'est pas une variable.
        nonvar(T),
        % On récupère le deuxième argument est bien une variable.
        var(X).
% regle(t?=X,orient).       --> true.
% regle(f(X)?=X,orient).    --> true.
% regle(T?=x,orient).       --> false.
% regle(T?=X,orient).       --> false.
% regle(t?=X,R).            --> R = orient.

regle(E,decompose) :-
        % On teste si E (passé en paramètre) est bien une équation.
        equation(E),
        % On récupère le premier argument de l'équation E dans E1.
        arg(1,E,E1),
        % On récupère le deuxième argument de l'équation E dans E2.
        arg(2,E,E2),
```

```
        % On vérifie que les deux arguments sont des termes composés.
        compound(E1),
        compound(E2),
        % On récupère le functor et l'arité de chaque terme.
        functor(E1,F1,A1),
        functor(E2,F2,A2),
        % On vérifie que les deux termes ont le même functor
        % (symbole de fonction).
        F1 == F2,
        % On vérifie finalement qu'ils ont le même arité.
        A1 == A2.
% regle(f(a,g(Y),X,Z)?=f(W,b,f(a),S),decompose).    --> true.
% regle(f(a,g(Y),X,Z)?=f(W,b,f(a)),decompose).      --> false.
% regle(g(a,g(Y),X,Z)?=f(W,b,f(a),S),decompose).    --> false.
% regle(X?=T,decompose).                            --> false.
% regle(X?=t,decompose).                            --> false.
% regle(x?=t,decompose).                            --> false.
% regle(x?=T,decompose).                            --> false.
% regle(f(a,g(Y),X,Z)?=f(W,b,f(a),S),R).            --> R= decompose.

regle(E,clash) :-
        % On teste si E (passé en paramètre) est bien une équation.
        equation(E),
        % On récupère le premier argument de l'équation E dans E1.
        arg(1,E,E1),
        % On récupère le deuxième argument de l'équation E dans E2.
        arg(2,E,E2),
        % On vérifie que les deux arguments sont des termes composés.
        compound(E1),
        compound(E2),
        % On récupère le functor et l'arité de chaque terme.
        functor(E1,F1,A1),
        functor(E2,F2,A2),
        % Cette disjonction réussit si:
        %    - Les deux termes n'ont pas le même functor.
        %    - Ou les deux termes n'ont pas le même arité.
        (not(F1 == F2);not(A1 == A2)).
% regle(f(a,g(Y),X,Z)?=f(W,b,f(a)),clash).      --> true.
% regle(f(a,g(Y),X,Z)?=g(W,b,f(a),S),clash).    --> true.
% regle(f(a,g(Y),X,Z)?=g(W,b,f(a)),clash).      --> true.
% regle(f(a,g(Y),X,Z)?=f(W,b,f(a),S),clash).    --> false.
% regle(f(t)?=X,clash).                         --> false.
% regle(a?=b,clash).                            --> false.
% regle(X?=Y,clash).                            --> false.
% regle(f(a,g(Y),X,Z)?=f(W,b,f(a),S),clash).    --> false.
% regle(f(a,g(Y),X,Z)?=f(W,b,f(a)),R).          --> R = clash.
% regle(f(a,g(Y),X,Z)?=g(W,b,f(a),S),R).        --> R = clash.
% regle(f(a,g(Y),X,Z)?=g(W,b,f(a)),R).          --> R = clash.
% +----------------------------------------------------------------------------+
```

```prolog

% Implantationdu prédicat: occur_check(V,T).
% Cette règle capture le cas où on est sur un noeud de l'arbre représentant T.
occur_check(V,T) :-
        % On vérifie que V est une variable.
        var(V),
        % On vérifie que T est un terme composé.
        compound(T),
        % En ne précisant pas le 1er argument de arg/3,
        % Prolog cherchera à rendre vraie la conjonction suivante:
        % (Ce qui revient à tester tous les sous-termes jusqu'à trouer celui qui
        % rend vraie la conjonction).
        arg(_,T,ST),
        occur_check(V,ST).
        % Etape suivant: on refait la même chose avec ST s'il est composé.

% Cette règle capture le cas où on est sur une feuille de l'arbre représentant T.
occur_check(V,T) :-
        % On vérifie que V et T sont des variables.
        var(V),
        var(T),
        % Puis on teste si V et T sont identiques.
        V == T.
% occur_check(X,f(a,g(Y,Z,f(X)))).    --> true.
% occur_check(X,f(a,g(Y,Z,f(W)))).    --> false.
% +-----------------------------------------------------------------------------+

% delete_eq/3:
% Supprime une equation et toutes ses occurrences d'un système d'équations.
delete_eq(_,[],[]).

delete_eq(E,[Head|TailL],[Head|TailM]) :-
        delete_eq(E,TailL,TailM),
        E \== Head.

delete_eq(E,[E|TailL],M) :- delete_eq(E,TailL,M).
% delete_eq(X?=T,[X?=T,Z?=F,X?=Y,T?=X,X?=T],L).    --> L = [Z?=F, X?=Y, T?=X].
% +-----------------------------------------------------------------------------+

% Implantation du prédicat: reduit(R,E,P,Q).
% Règle logique représentant la condition d'arrêt de reduit/4 avec rename.
reduit(rename,_,[],[]).

% Le cas où E et la première équation de P sont les mêmes.
% E ne sera pas dans le système d'équations résultant (Q).
reduit(rename,X?=T,[E1?=E2|TailL],TailM) :-
        % Les opérands des deux équations doivent être les mêmes.
        X==E1,
        T==E2,
```

```
        % On continue de parcourir le système d'équations avant d'unifier.
        reduit(rename,X?=T,TailL,TailM),
        % On demande à prolog d'unifier X et T.
        X=T.

% Le cas où E et la première équation de P sont différentes.
% La même équation est dans Q aussi.
reduit(rename,X?=T,[E1?=E2|TailL],[E1?=E2|TailM]) :-
        % On continue de parcourir le système d'équations
        reduit(rename,X?=T,TailL,TailM).
% reduit(rename,X?=Y,[X?=Y,Y?=X,Y?=Z,X?=Y],Q).
% --> X = Y,
%     Q = [Y?=Y, Y?=Z].

% reduit/4 pour simplify est implanté de la même façon que pour rename.
reduit(simplify,_,[],[]).

reduit(simplify,X?=T,[E1?=E2|TailL],TailM) :-
        X==E1,
        T==E2,
        reduit(simplify,X?=T,TailL,TailM),
        X=T.

reduit(simplify,X?=T,[E1?=E2|TailL],[E1?=E2|TailM]) :-
        reduit(simplify,X?=T,TailL,TailM).
% reduit(simplify,X?=a,[X?=a,Y?=X,Y?=Z,X?=a],Q).
% --> X = a,
%     Q = [Y?=a, Y?=Z] .

% reduit/4 pour expand est implanté de la même façon que pour rename.
reduit(expand,_,[],[]).

reduit(expand,X?=T,[E1?=E2|TailL],TailM) :-
        X==E1,
        T==E2,
        reduit(expand,X?=T,TailL,TailM),
        X=T.

reduit(expand,X?=T,[E1?=E2|TailL],[E1?=E2|TailM]) :-
        reduit(expand,X?=T,TailL,TailM).
% reduit(expand,X?=f(Y,g(a)),[X?=f(Y,g(a)),Y?=X,Y?=Z,X?=f(Y,g(a))],Q).
% --> X = f(Y, g(a)),
%     Q = [Y?=f(Y, g(a)), Y?=Z].
% reduit(expand,X?=f(h(a)),[X?=f(h(a))],Q).
% --> X = f(h(a)),
%     Q = [] .

% reduit/4 pour check arrête de parcourir l'arbre construit par prolog et échoue.
reduit(check,_,P,P) :- !, fail.
```

```
% reduit(check, Y ?= f(a,X,Y) ,[Y ?= f(a,X,Y),f(a) ?= X, Z ?= f(Y)], Q).
% --> false.

reduit(orient,E,P,Q) :-
        % On récupère les deux arguments de l'équation.
        arg(1,E,T),
        arg(2,E,X),
        % On supprime les occurrences de l'équation de notre système.
        delete_eq(E,P,Ptemp),
        % On réoriente l'équation et on l'insère dans le nouveau système.
        append([(X ?= T)|[]],Ptemp,Q).
% reduit(orient,f(a) ?= Y,[f(a) ?= X, Z ?= f(Y)], Q).
% --> Q = [Y?=f(a), f(a)?=X, Z?=f(Y)].

reduit(decompose,X?=T,P,Q) :-
        % On récupère dans des listes les arguments des deux termes X et T,
        % et on ignore leurs functors (premiers éléments des deux listes).
        X=..[_|ArgsX],
        T=..[_|ArgsT],
        % ON décompose les deux termes à l'aide du prédicat decompose/3.
        decompose(ArgsX,ArgsT,Decomposed),
        % On supprime les occurrences de l'équation de notre système.
        delete_eq(X?=T,P,Ptemp),
        % On insère les équations résaltants de la décomposition
        % dans le nouveau système.
        append(Decomposed,Ptemp,Q).
% reduit(decompose,f(X,Y)?=f(g(Z),h(a)),
% [f(X,Y)?=f(g(Z),h(a)),Z?=f(Y),f(Y,X)?=f(g(Z),h(a)),f(X,Y)?=f(g(Z),h(a))], Q).
% --> Q = [X?=g(Z), Y?=h(a), Z?=f(Y), f(Y, X)?=f(g(Z), h(a))] .

% reduit/4 pour clash arrête de parcourir l'arbre construit par prolog et échoue.
reduit(clash,_,P,P) :- !, fail.
% reduit(clash, f(X,Y) ?= g(y), [f(X,Y) ?= g(y), Z?= f(Y)], Q). --> false.
% +--------------------------------------------------------------------------+

% le prédicat decompose/3 associe les éléments des deux listes (dans l'ordre).
decompose([],[],[]).
decompose([X|TailL],[T|TailM],[X?=T|TailQ]) :- decompose(TailL,TailM,TailQ).
% +--------------------------------------------------------------------------+

% Implantation du prédicat: unifie(P).
% Règle logique exprimant la condition d'arrêt du prédicat unifie/1
% L'affichage ne s'éffectue que lorsque le prédicat echo_on/0 est défini.
unifie([]) :- (not(current_predicate(echo_on/0));echo('\nYes\n')).

unifie([E|Tail]) :-
        % Affichage du système d'équations.
        (not(current_predicate(echo_on/0));
            (echo('system :\t'), echo([E|Tail]), echo('\n'))),
```

```
        regle(E,R),!,
        % Affichage de la règle et de l'équation correspondante.
        (not(current_predicate(echo_on/0));
            (echo(R), echo(' :\t'), echo(E), echo('\n'))),
        % On applique la réduction.
        reduit(R,E,[E|Tail],Q),
        % Si le système d'équations résultant est différent de celui de départ,
        % la deuxième tranche de la disjonction suivante ne sera pas évaluée, et
        % le prédicat continue à parcourir l'arbre construit par prolog.
        % sinon si le système résultant et celui de départ sont les mêmes, (le
        % cas de check et clash), le prédicat arrête de parcourir l'arbre
        % construit par prolog et échoue.
        (Q \== [E|Tail];(!,fail)),
        % On continue avec le nouveau système.
        unifie(Q).

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

% Implantation du prédicat: choix_premier(P,Q,E,R).
choix_premier([E|Tail],Tail,E,R) :-    regle(E,R).


% Implantation du prédicat: choix_pondere(P,Q,E,R).
choix_pondere(P,Q,E,R) :-
        % La première équation incluse dans le système d'équations sur laquelle
        % une des règles du poids actuel s'applique, est selectionnée.
        member(E,P),
        (regle(E,clash);regle(E,check)),
        % On récupère le nom de la règle qui s'applique
        % sur l'équation selectionnée.
        regle(E,R),
        % On supprime les occurrences de cette règle du système d'équations P.
        delete_eq(E,P,Q).

choix_pondere(P,Q,E,R) :-
        member(E,P),
```

```
        (regle(E,rename);regle(E,simplify)),
        regle(E,R),
        delete_eq(E,P,Q).

choix_pondere(P,Q,E,R) :-
        member(E,P),
        regle(E,orient),
        regle(E,R),
        delete_eq(E,P,Q).

choix_pondere(P,Q,E,R) :-
        member(E,P),
        regle(E,decompose),
        regle(E,R),
        delete_eq(E,P,Q).

choix_pondere(P,Q,E,R) :-
        member(E,P),
        regle(E,expand),
        regle(E,R),
        delete_eq(E,P,Q).
% ?- choix_pondere([f(X,Y,Z)?=f(a,b,c),a?=W,X?=a,X?=t(a)],Q,E,R).
% Q = [f(X, Y, Z)?=f(a, b, c), a?=W, X?=t(a)],
% E = X?=a,
% R = simplify .
% +----------------------------------------------------------------------------+

% Implantation du prédicat: unifie(P,S).
unifie([],_).

unifie(P,S) :-
        (not(current_predicate(echo_on/0));
            (echo('system :\t'), echo(P), echo('\n'))),
        % Appellera: S(P,Q,E,R) où S est le nom de la stratégie.
        call(S,P,Q,E,R),
        (not(current_predicate(echo_on/0));
            (echo(R), echo(' :\t'), echo(E), echo('\n'))),
        reduit(R,E,[E|Q],Q2),
        ((R\==check,R\==clash);(!,fail)),
        unifie(Q2,S).
% +----------------------------------------------------------------------------+
%         ____                  _   _               ____
%        / __ \                | | (_)             |___ \ _
%       | |  | |_   _  ___  ___| |_ _  ___  _ __     __) (_)
%       | |  | | | | |/ _ \/ __| __| |/ _ \| '_ \   |__ <
%       | |__| | |_| |  __/\__ \ |_| | (_) | | | |  ___) |_
%        \___\_\\__,_|\___||___/\__|_|\___/|_| |_| |____/(_)
%
% +----------------------------------------------------------------------------+
```

```

% Implantation des prédicats: unif(P,S) et trace_unif(P,S).
% On désactive la trace d'affichage et on fait appel à unifie(P,S)
unif(P,S) :- clr_echo,unifie(P,S).
% On active la trace d'affichage et on fait appel à unifie(P,S)
trace_unif(P,S) :- set_echo,unifie(P,S).

% unif([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(Y)],choix_premier).
% -- > X = g(f(h(a))),
%       Y = h(a),
%       Z = f(h(a)) .
% unif([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(X)],choix_premier). --> false.

% unif([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(Y)],choix_pondere).
% --> X = g(f(h(a))),
%     Y = h(a),
%     Z = f(h(a)) .
% unif([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(X)],choix_pondere). --> false.

% trace_unif([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(Y)],choix_pondere).
% --> system :        [f(_1072,_1074)?=f(g(_1078),h(a)),_1078?=f(_1074)]
%     decompose :     f(_1072,_1074)?=f(g(_1078),h(a))
%     system :        [_1072?=g(_1078),_1074?=h(a),_1078?=f(_1074)]
%     expand :        _1072?=g(_1078)
%     system :        [_1074?=h(a),_1078?=f(_1074)]
%     expand :        _1074?=h(a)
%     system :        [_1078?=f(h(a))]
%     expand :        _1078?=f(h(a))
%     X = g(f(h(a))),
%     Y = h(a),
%     Z = f(h(a)) .
% trace_unif([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(X)],choix_pondere).
% --> system :        [f(_1072,_1074)?=f(g(_1078),h(a)),_1078?=f(_1072)]
%     decompose :     f(_1072,_1074)?=f(g(_1078),h(a))
%     system :        [_1072?=g(_1078),_1074?=h(a),_1078?=f(_1072)]
%     expand :        _1072?=g(_1078)
%     system :        [_1074?=h(a),_1078?=f(g(_1078))]
%     check : _1078?=f(g(_1078))
%     expand :        _1074?=h(a)
%     system :        [_1078?=f(g(_1078))]
%     check : _1078?=f(g(_1078))
%     false.

% +------------------------------------------------------------------------------+
```