|  |
| --- |
| Projet LMC  Implémentation de l’algorithme d’unification de Martelli-Montanari en Prolog |
| V. DAMM – F. NOSARI - 7-déc.-16 |
| Table des matières  [Règles d’unification 3](#_Toc468898990)  [Teste d’applicabilité des règles 4](#_Toc468898991)  [Rename 4](#_Toc468898992)  [Simplify 4](#_Toc468898993)  [Expand 4](#_Toc468898994)  [Check 5](#_Toc468898995)  [Orient 5](#_Toc468898996)  [Decompose 5](#_Toc468898997)  [Clash 5](#_Toc468898998)  [Application des règles 7](#_Toc468898999)  [Rename 7](#_Toc468899000)  [Simplify 7](#_Toc468899001)  [Expand 8](#_Toc468899002)  [Check 8](#_Toc468899003)  [Orient 8](#_Toc468899004)  [Decompose 8](#_Toc468899005)  [Clash 9](#_Toc468899006)  [Unification 10](#_Toc468899007)  [Stratégies 11](#_Toc468899008)  [Choix de la première équation 11](#_Toc468899009)  [Choix pondéré 12](#_Toc468899010)  [Choix aléatoire 13](#_Toc468899011)  [Prédicats annexes 14](#_Toc468899012)  [Teste d’occurrence 14](#_Toc468899013)  [Création de liste d’équations 14](#_Toc468899014)  [Liste des paramètres d’un terme 14](#_Toc468899015)  [Programme 16](#_Toc468899016)  [Webographie 19](#_Toc468899017) |

# Règles d’unification

La liste des règles applicables est la suivante :

1. Rename {x ?= t} ∪ P ′ ; S → P ′ [x/t]; S[x/t] ∪ {x = t} si t est une variable
2. Simplify {x ?= t} ∪ P ′ ; S → P ′ [x/t]; S[x/t] ∪ {x = t} si t est une constante
3. Expand {x ?= t} ∪ P ′ ; S → P ′ [x/t]; S[x/t] ∪ {x = t} si t est composé et x n’apparaît pas dans t
4. Check {x ?= t} ∪ P ′ ; S → ⊥ si x 6= t et x apparaît dans t
5. Orient {t ?= x} ∪ P ′ ; S → {x ?= t} ∪ P ′ ; S si t n’est pas une variable
6. Decompose {f(s1, · · · , sn) ?= f(t1, · · · , tn)} ∪ P ′ ; S → {s1 ?= t1, · · · , sn ?= tn} ∪ P ′ ; S
7. Clash {f(s1, · · · , sn) ?= g(t1, · · · , tm)} ∪ P ′ ; S → ⊥ si f 6= g ou m 6= n

## Teste d’applicabilité des règles

Pour chaque règle on créer un prédicat regle(E, r) tel que le prédicat est vrai si on peut appliquer la règle r à l’équation E.

### Rename

Pour la règle rename on doit vérifier si t est une variable et si x en est une également.

regle(X?=T,simplify) :- **var**(X), **var**(T).

### Simplify

Pour la règle simplify on doit vérifier si t et une constante et si x est une variable.

regle(X?=T,simplify) :- **var**(X), **atomic**(T).

### Expand

Pour la règle expand on doit vérifier si t et composé, si x est une variable et si x n’apparait pas dans t.

regle(X?=T,expand) :- **var**(X), **compound**(T), \+occur\_check(X,T).

Le prédicat occur\_check(X,T) est décrit dans la section « Prédicats annexes ».

### Check

Pour la règle check on doit vérifier si x est une variable et si x apparait dans t.

regle(X?=T,check) :- X \== T, **var**(X), !, occur\_check(X,T).

Le prédicat occur\_check(X,T) est décrit dans la section « Prédicats annexes », il est vrai si X apparait dans T.

### Orient

Pour la règle orient on doit vérifier si x est une variable et si t n’est pas une variable.

regle(T?=X,orient) :- **var**(X), **nonvar**(T).

### Decompose

Pour la règle decompose on doit vérifier si f(s1, ···, sn) et f(t1, ···, tn) sont deux fonctions de même symbole et de même arité.

regle(S?=T,decompose) :- compound(S), compound(T),!,

compound\_name\_arity(S,SF,SA),

compound\_name\_arity(T,TF,TA), SF==TF, SA==TA.

### Clash

Pour la règle clash on doit vérifier si f(s1, ···, sn) et g(t1, ···, tn) sont deux fonctions de même symbole ou de même arité.

regle(F?=G,clash) :- **compound**(F), **compound**(G),!,

compound\_name\_arity(F,FF,FA),

compound\_name\_arity(G,GF,GA),

not( (FF == GF, FA == GA) ).

## Application des règles

Pour l’application d’une règle r on créer le predicat reduit(r,E,P,Q) qui transforme le système d’équations P en le système d’équations Q par application de la règle r de transformation sur l’équation E.

Par soucis de compréhension et pour une meilleur lisibilité du code nous avons décomposé l’application d’une règle en 2 prédicats tel que :

r(E,P,R) ← ….

reduit(r,E,P,Q) ← r(E,P,Q).

### Rename

La règle rename remplace x en t (x=t) et propage cette modification dans le reste du système d’équation.

rename(X?=T,[X?=T|L],Q) :- X=T,Q=L.

reduit(rename,E,P,Q) :- rename(E,P,Q).

### Simplify

La règle simplify remplace x en t (x=t) et propage cette modification dans le reste du système d’équation.

simplify(X?=T,[X?=T|L],Q) :- X=T,Q=L.

reduit(simplify,E,P,Q) :- simplify(E,P,Q).

### Expand

La règle expand remplace x en t (x=t) et propage cette modification dans le reste du système d’équation.

expand(X?=T,[X?=T|L],Q) :- X=T,Q=L.

reduit(expand,E,P,Q) :- expand (E,P,Q).

### Check

La règle check ne fait rien d’autre que d’échouer, en effet si elle est applicable alors l’unification n’est pas possible.

Puisque qu’elle échoue toujours on peut la remplacer par le prédicat fail.

### Orient

La règle orient remplace inverse t et x pour obtenir x?=t.

orient(T?=X,[T?=X|L],Q) :- **append**([X?=T],L,Q).

reduit(orient,E,P,Q) :- orient(E,P,Q).

### Decompose

La règle decompose remplace f(s1, ···, sn)?=f(t1, ···, tn) par s1?=t1, ···, sn?=tn dans le système d’équation.

decompose(X?=T,[X?=T|L],Q) :- term\_params(X, XL), term\_params(T, TL),

make\_list(XL, TL, LR), **append**(LR,L,Q).

reduit(decompose,E,P,Q) :- decompose(E,P,Q).

Le prédicat term\_param(T,L) est décrit dans la section « Prédicats annexes », il attribue à L les paramètres de T.

### Clash

La règle clash ne fait rien d’autre que d’échouer, en effet si elle est applicable alors l’unification n’est pas possible.

Puisque qu’elle échoue toujours on peut la remplacer par le prédicat fail.

# Unification

L’unification consiste à appliquer une règle si on le peut. On utilise donc le prédicat regle(E, r) pour tester si la règle r est applicable et si tel est le cas alors on utilise le prédicat reduit(r,E,P,Q) pour réduire le système d’équations.

Pour cela on créer le prédicat unifie(P) qui unifie le système d’équation P tel que :

unifie([X?=T|L]) ← regle(E,r), !, reduit(r,X?=T,[X?=T|L],Q), unifie(Q).

On teste si la règle est applicable, si on ne peut pas le prédicat échoue (cut), sinon on réduit le système d’équation jusqu’à avoir un système d’équation vide ou une unification impossible (aucune règle applicable).

Exemple pour la règle rename :

unifie([X?=T|L]) :- regle(X?=T,rename), reduit(rename,X?=T,[X?=T|L],Q),

unifie(Q), !.

unifie([]) :- echo('Fin de l\'unification\n').

Il y a cependant 2 exceptions, comme dit précédemment, les règles check et clash sont remplacées par le prédicat fail, elles sont donc faites de la façon suivante :

unifie([X?=T|L]) ← regle(E,r), !, fail.

Exemple pour la règle check :

unifie([X?=T|L]) :- regle(X?=T,rename), fail, !.

# Stratégies

En ce qui concerne les stratégies d’unification, on peut en utiliser plusieurs. Pour cela on peut créer une variante du prédicat unifie(P) qui est unifie(P,s) qui unifie P avec la stratégie s.

On créer aussi une variante du prédicat unifie pour choisir une règle et une stratégie:

unifie([X?=T|L], regle CHOIX) ← regle(E,r), !, reduit(r,X?=T,[X?=T|L],Q),

unifie(Q, CHOIX).

## Choix de la première équation

La stratégie utilisé jusqu’à maintenant choisissait la première équation du système d’équation et la première règle qui était applicable, on peut l’appeler choix\_premier.

unifie([X?=T|L],choix\_premier) :- unifie([X?=T|L]).

## Choix pondéré

La stratégie pondérée (choix\_pondere) consiste à favoriser des règles plutôt que des autres, on prendra la pondération suivante :

1. clash, check
2. rename, simplify
3. orient
4. decompsoe
5. expand

Pour le choix pondéré nous avons procédé de la façon suivant :

Si la règle est applicable  
 on applique la règle  
Sinon  
 on regarde pour les autres règles dans l’ordre de la pondération

unifie([X?=T|L],choix\_pondere):-

( (regle(X?=T,clash)) ->

unifie([X?=T|L],clash,choix\_pondere) ;

( (regle(X?=T,check)) ->

unifie([X?=T|L],check,choix\_pondere) ;

( (regle(X?=T,rename)) ->

unifie([X?=T|L],rename,choix\_pondere) ;

( (regle(X?=T,simplify)) ->

unifie([X?=T|L],simplify,choix\_pondere) ;

( (regle(X?=T,orient)) ->

unifie([X?=T|L],orient,choix\_pondere) ;

( (regle(X?=T,decompose)) ->

unifie([X?=T|L],decompose,choix\_pondere) ;

( (regle(X?=T,expand)) ->

unifie([X?=T|L],expand,choix\_pondere) ))))))).

## Choix aléatoire

La stratégie aléatoire (choix\_aleatoire) consiste à choisir une règle au hasard jusqu’à arriver à une règle qui est applicable.

unifie([X?=T|L],choix\_aleatoire) :-

random\_member(F,[rename,simplify,expand,check,orient,decompose,clash]),

(unifie([X?=T|L],F,choix\_aleatoire))

->

(**true**)

;

(unifie([X?=T|L],choix\_aleatoire)) .

# Prédicats annexes

Pour simplifier le programme et/ou pour factoriser, nous avons créé des prédicats annexes.

## Teste d’occurrence

Le prédicat occur\_check(V,T) réussi si V est dans T

occur\_check(V,T) :- contains\_var(V,T).

## Création de liste d’équations

Le prédicat make\_liste(A,B,L) créer une liste L du type [A1 ?=B1, …, An ?=Bn] à partir de deux listes A et B respectivement de la forme [A1, …, An] et [B1, …, Bn] où les Ai et Bi sont des termes.

make\_list([X|L1],[T|L2],L) :- make\_list(L1,L2,Z), append([X?=T],Z,LR), L=LR.

% Termine si la liste est vide

make\_list([],[],L) :- L=[].

## Liste des paramètres d’un terme

Le prédicat term\_params(X,L) créer une liste L qui contient les paramètres du terme X.

% Cas où on met directement le terme en paramètre du prédicat

% (on transforme le terme en liste [symbole, param1, param2, ...])

term\_params(X,L) :- \+is\_list(X), !, X=..XP, term\_params(XP,L).

% Cas où on met une liste de paramètres

% (on ne garde que la queue de la liste car le premier item est le symbole du terme)

term\_params([\_|P],L) :- L=P.

% Cas où la liste est vide

term\_params([],L) :- L=L.

# Programme

Le programme suivant a été effectué à l’aide du langage Prolog et d’une de ses implémentations SWI-Prolog (<http://www.swi-prolog.org/>).

% 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(\_).

% Regles

% Conditions d'utilisation des regles

% Renomage

% Vrai si les X et T sont des variables

regle(X?=T,rename) :- var(X), var(T).

% Simplification

% Vrai si X est une variable et T une constante

regle(X?=T,simplify) :- var(X), atomic(T).

% Developpement

% Vrai si X est une variable, si T est compose et si X n'apparait pas dans T

regle(X?=T,expand) :- var(X), compound(T), \+occur\_check(X,T).

% Teste d'ocurrence

% Faux si X =/= T et si X apparait dans T

regle(X?=T,check) :- X \== T, var(X), !, occur\_check(X,T).

% Orientation

% Vrai si T n'est pas une variable et si X en est une variable

regle(T?=X,orient) :- var(X), nonvar(T).

% Decomposition

% Vrai si S et T sont deux fonctions de meme symbole et de meme arite

regle(S?=T,decompose) :- compound(S), compound(T),!,

compound\_name\_arity(S,SF,SA),

compound\_name\_arity(T,TF,TA), SF==TF, SA==TA.

% Conflit

% Faux si F et G sont deux fonctions de differents symboles ou de differentes arites

regle(F?=G,clash) :- compound(F), compound(G),! ,

compound\_name\_arity(F,FF,FA),

compound\_name\_arity(G,GF,GA), **not**( (FF == GF, FA == GA) ).

% Application des regles

% Renomage

rename(X?=T,[X?=T|L],Q) :- X=T,Q=L.

% Simplification

simplify(X?=T,[X?=T|L],Q) :- X=T,Q=L.

% Developpement

expand(X?=T,[X?=T|L],Q) :- X=T,Q=L.

% Orientation

orient(T?=X,[T?=X|L],Q) :- append([X?=T],L,Q).

% Decomposition

decompose(X?=T,[X?=T|L],Q) :- term\_params(X, XL), term\_params(T, TL),

make\_list(XL, TL, LR), append(LR,L,Q).

% Reduction : transforme le systeme d equations P en le systeme dequations Q par application de la regle de transformation sur lequation E

% Renomage

reduit(rename,E,P,Q) :- rename(E,P,Q).

% Simplification

reduit(simplify,E,P,Q) :- simplify(E,P,Q).

% Developpement

reduit(expand,E,P,Q) :- expand(E,P,Q).

% Orientation

reduit(orient,E,P,Q) :- orient(E,P,Q).

% Decomposition

reduit(decompose,E,P,Q) :- decompose(E,P,Q).

% Unification

unifie([]) :- echo('\n\n'), echo('Fin de l\'unification\n').

unifie([],\_) :- echo('\n\n'), echo('Fin de l\'unification\n').

% Renomage

unifie([X?=T|L]) :- regle(X?=T,rename), !, echo('\nsystem : '), echo([X ?=T |L]),

echo('\n'), echo('\trename : '), echo(X?=T), echo('\n'),

reduit(rename,X?=T,[X?=T|L],Q), unifie(Q), !.

% Simplification

unifie([X?=T|L]) :- regle(X?=T,simplify), !, echo('\nsystem : '), echo([X?=T|L]),

echo('\n'), echo('\tsimplify : '), echo(X?=T), echo('\n'),

reduit(simplify,X?=T,[X?=T|L],Q), unifie(Q), !.

% Developpement

unifie([X?=T|L]) :- regle(X?=T,expand), !, echo('\nsystem : '), echo([X?=T|L]),

echo('\n'), echo('\texpand : '), echo(X?=T), echo('\n'),

reduit(expand,X?=T,[X?=T|L],Q), unifie(Q), !.

% Teste d'ocurrence

unifie([X?=T|L]) :- regle(X?=T,check), !, echo('\nsystem : '), echo([X?=T|L]),

echo('\n'), echo('\tcheck : '), echo(X?=T), echo('\n'), **fail**, !.

% Orientation

unifie([X?=T|L]) :- regle(X?=T,orient), !, echo('\nsystem : '), echo([X?=T|L]),

echo('\n'), echo('\torient : '), echo(X?=T), echo('\n'),

reduit(orient,X?=T,[X?=T|L],Q), unifie(Q), !.

% Decomposition

unifie([X?=T|L]) :- regle(X?=T,decompose), !, echo('\nsystem : '), echo([X ?=T |L]),

echo('\n'), echo('\tdecompose : '), echo(X?=T), echo('\n'),

reduit(decompose,X?=T,[X?=T|L],Q), unifie(Q), !.

% Conflit

unifie([X?=T|L]) :- regle(X?=T,clash), !, echo('\nsystem : '), echo([X?=T|L]),

echo('\n'), echo('\tclash : '), echo(X?=T), echo('\n'), **fail**, !.

# Webographie

* SWI-Prolog   
  <http://www.swi-prolog.org/>
* Unification — Wikipédia  
  <https://fr.wikipedia.org/wiki/Unification>