

Rapport

Projet L.M.C.

HAJEK Simon

COURTIL Antoine

Question 1 :

rule(X ?= Y, rename) :

Prédicat de règle qui retourne vrai si la règle "rename" est applicable sur l'équation. C'est-à-dire si Y est une variable dans X.

rule(X ?= Y, simplify) :

Prédicat de règle qui retourne vrai si la règle "simplify" est applicable sur l'équation. C'est-à-dire si Y est une constante dans X.

rule(X ?= Y, expand) :

Prédicat de règle qui retourne vrai si la règle "expand" est applicable sur l'équation. C'est-à-dire si Y est une fonction et que X n'apparaît pas dans ses arguments.

rule(X ?= Y, orient) :

Prédicat de règle qui retourne vrai si la règle "orient" est applicable sur l'équation. C'est-à-dire si X n'est pas une variable.

rule(X ?= Y, decompose) :

Prédicat de règle qui retourne vrai si la règle "decompose" est applicable sur l'équation. C'est-à-dire si l'équation peut se décomposer en deux fonction X et Y.

rule(X ?= Y, clash) :

Prédicat de règle qui retourne vrai si la règle "clash" est applicable sur l'équation. C'est-à-dire si X et Y sont des fonctions et que leurs noms sont différents.

rule(X ?= Y, occur_check) :

Prédicat de règle qui retourne vrai si la règle "occur_check" est applicable sur l'équation. C'est-à-dire si X est différent de Y et que X apparaît dans Y.

rule(X ?= Y, clean) :

Prédicat de règle qui retourne vrai si la règle "clean" est applicable sur l'équation.

occur_check(V,T) :

Prédicat qui permet de tester si la variable V apparait dans le terme T.

var_into_arg(V,T) :

Prédicat qui permet de tester si $T = V$, alors la variable apparait dans le terme T

Si T composé de plusieurs arguments, on vérifie si V apparait dans un de ces arguments

var_into_term(V, T, A) :

Prédicat qui parcourt les arguments A de T pour vérifier si V apparait dans T.

reduce(rule, X \neq Y, P, Q) :

Prédicat qui transforme le système d'équations P en le système d'équations Q par application de la règle de transformation R à l'équation E.

decomposition(X, Y, N, Q) :

Décomposition des arguments d'une fonction en une liste d'équations

unifie([X|T], Strategie) :

Predicat où [X|T] est un système d'équations à résoudre représenté sous la forme d'une liste [S1 \neq T1,...,SN \neq TN].

Question 2 :

choix_premier([X|T]) :

Unifie avec une stratégie de base : prendre les équations dans l'ordre de lecture de gauche à droite.

choix_premier([]) :

Cas d'arrêt.

weight(rule,5) :

Définition des différents poids matérialisant les priorités entre les différentes opérations

choix_pondere(X) :

Unifie avec une stratégie de préférence d'équations en fonction de leur opération.

choix_pondere([]) :

Cas d'arrêt.

maxWeight([X,Y|P], R, E) :

Cherche à récupérer celle qui a le poids le plus fort.

extract([T|R],X,Res) :

Récupère la bonne équation à traiter.

Question 3 :

`trace(SystEq,Strategie,Trace) :`

Traitement choix user trace.

`trace_unif(P,Strategie) :`

Activation de la trace avec `set_echo`.

`unif(P,Strategie) :`

Désactivation de la trace avec `set_echo`.

Gestion des interactions utilisateurs :

Afin d'avoir un déroulement complet du programme, veuillez entrer juste la commande **run**.

Exemple de vérification de la stratégie :

```
readStrategie(SystEq,Strategie,Trace) :-  
    repeat,  
    write("\n\nQuelle stratégie voulez-vous utiliser ? ( \'premier.\' OU \'pondere.\')\n"),  
    write('>> Stratégie : '),  
    read(Strategie),  
    (Strategie == premier ; Strategie == pondere),  
    write(Strategie).
```

Afin de vérifier que l'utilisateur réponde correctement à quelle stratégie utiliser pour l'unification, on utilise la propriété de **repeat** qui permet de vérifier une variable d'entrée, ici *Strategie*, avec comme seule possibilité de valeur premier ou pondéré grâce à la ligne en **surlignage**. Tant que *Strategie* ne vaut pas l'une des deux valeurs, il est demandé à l'utilisateur d'entrer une stratégie.

Tests

Test pour un unifie true.

```
?- unifie([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(Y)]).  
  
system : [f(_G983,_G984)?=f(g(_G986),h(a)),_G986?=f(_G984)]  
decompose : f(_G983,_G984)?=f(g(_G986),h(a))  
system : [_G984?=h(a),_G983?=g(_G986),_G986?=f(_G984)]  
expand : _G984?=h(a)  
system : [_G983?=g(_G986),_G986?=f(h(a))]  
expand : _G983?=g(_G986)  
system : [_G986?=f(h(a))]  
expand : _G986?=f(h(a))  
Système d'equation unifiable.  
true .
```

Test pour une unifie false.

```
?- unifie([f(X,Y) ?= f(g(Z),h(a)), Z ?= f(X)]).  
  
system : [f(_G983,_G984)?=f(g(_G986),h(a)),_G986?=f(_G983)]  
decompose : f(_G983,_G984)?=f(g(_G986),h(a))  
system : [_G984?=h(a),_G983?=g(_G986),_G986?=f(_G983)]  
expand : _G984?=h(a)  
system : [_G983?=g(_G986),_G986?=f(_G983)]  
expand : _G983?=g(_G986)  
system : [_G986?=f(g(_G986))]  
occur check : _G986?=f(g(_G986))  
false.
```

Comparaison entre premier et pondéré

Syst $\models f(g(Z), h(a)), Z \models f(Y), f(X, Y, Z) \models g(a, b, c)$.

Premier : Ici on est obligé d'aller à la dernière équation au clash pour se rendre compte qu'elle est fausse.

```
system :
[f(_G1031,_G1032)?=f(g(_G1034),h(a)),_G1034?=f(_G1032),f(_G1031,_G1032,_G1034)?=g(a,b,c)]
decompose : f(_G1031,_G1032)?=f(g(_G1034),h(a))
system :
[_G1032?=h(a),_G1031?=g(_G1034),_G1034?=f(_G1032),f(_G1031,_G1032,_G1034)?=g(a,b,c)]
expand : _G1032?=h(a)
system : [_G1031?=g(_G1034),_G1034?=f(h(a)),f(_G1031,h(a),_G1034)?=g(a,b,c)]
expand : _G1031?=g(_G1034)
system : [_G1034?=f(h(a)),f(g(_G1034),h(a),_G1034)?=g(a,b,c)]
expand : _G1034?=f(h(a))
system : [f(g(f(h(a))),h(a),f(h(a)))?=g(a,b,c)]
clash : f(g(f(h(a))),h(a),f(h(a)))?=g(a,b,c)
false.
```

Pondéré : Ici il évalue les priorités et fait directement le clash pour se rendre compte que c'est non unifiable.

```
system :
[f(_G1031,_G1032)?=f(g(_G1034),h(a)),_G1034?=f(_G1032),f(_G1031,_G1032,_G1034)?=g(a,b,c)]
clash : f(_G1031,_G1032,_G1034)?=g(a,b,c)
clash : f(_G1031,_G1032,_G1034)?=g(a,b,c)
clash : f(_G1031,_G1032,_G1034)?=g(a,b,c)
false.
```

Test orient :	<code>unifie([a ?= X])</code> renvoie bien $X = a$
Test expand :	<code>unifie([X ?= f(a)])</code> renvoie bien $X = f(a)$
Test rename :	<code>unifie([X ?= Y])</code> renvoie bien $X = Y$
Test simplify :	<code>unifie([X ?= a])</code> renvoie bien $X = a$
Test check :	<code>unifie([X ?= f(X)])</code> renvoie bien false et <code>unifie([X ?= f(Y)])</code> renvoie bien $X = f(Y)$
Test decompose :	<code>unifie([f(X, Y, Z) ?= f(a, b, c)])</code> renvoie bien $X = a, Y = b, Z = c$
Test clash :	<code>unifie([f(X, Y, Z) ?= g(a, b, c)])</code> renvoie bien false

ANNEXE

Totalité du code pour le projet.