Algorithme de recherche - transformation de programmes

Gonzague YERNAUX

12 décembre 2016

Précédemment

- Transformations : unfolding, slicing, predicate pairing, ...
- Comparaison sémantique

Au programme

1. Algorithme naïf de comparaison
2. Framework de similarité entre algorithmes
3. Exemples d'unfoldings sur des listes
4. Conclusions

Algorithme naïf de comparaison

Algorithme naïf de comparaison

Soient A et B deux programmes à comparer, et T un ensemble de transformations. Soient $SA := \{A\}$ et $SB := \{B\}$

Tant que c'est intéressant de continuer et que $SA \cap SB = \emptyset$:
{
  Choisir un programme P dans SA, un programme P' dans SB, une transformation T dans T, une transformation T' dans T, ajouter PT := T(P) et PT' := T'(P') à SA et SB respectivement.
}

Si $SA \cap SB \neq \emptyset$ alors on a trouvé. Sinon les deux algorithmes sont différents.

Framework de similarité entre algorithmes

Le framework : transformations

Transformations préservant (dans une certaine mesure) le sens des programmes :

- Unfolding
- Slicing
- Folding
- Remplacement/suppression sémantique
- Spécialisation

Le framework : transformations

Transformations préservant (dans une certaine mesure) le sens des programmes :

- Unfolding
- Slicing
- Folding
- Remplacement/suppression sémantique
- Spécialisation

Mais aussi les transformations typiquement appliquées en dernier lieu :

- Renommage + permutation
- Simplification de contraintes

Exemple

```
p(A, B):- p2(1, A, B).
p2(I, A, B):- I < 10, A1 = A + I,
    p2(I + 1, A1, B).
p2(I, A, A):- I = 10.
```

```
q(X, Y):- Y1 = Y + 1, q2(X, Y1, 2).
q2(B, A, I):- I < 10, A1 = A + I,
    q2(A1, B, I + 1).
q2(A, A, I):- I = 10.
```

Le framework : paramétrage

Paramétrage

Idée : les transformations de l'ensemble T peuvent être paramétrées.

Exemples

- Unfolding[4] = unfolder jusque 4 fois
- Unfolding[*] = unfolder un nombre quelconque de fois
- Slicing[arguments inutiles]
- Slicing[tranche d'une variable]
- Simplification[strictement les mêmes contraintes]
- Simplification[contraintes équivalentes sémantiquement]

Une transformation particulière : la spécialisation

Spécialisation

La spécialisation est obtenue avec une seule modification : fixer une valeur dans une contrainte du programme. (Typiquement utilisé avec l'unfolding.)

Utilité : Spécialisation$[X, Y = N_1 \rightarrow N_2]$ permet de tester l'équivalence sur un jeu de valeurs d'input (allant de N_1 à N_2).

Exemples d'unfoldings sur des listes

Somme d'une liste d'entiers

Calcul de la somme d'une liste d'entiers : $sum(List, Sum)$. Deux versions :

- version récursive "directe" ;
- version avec accumulateur.

Résultat après unfolding : Les **mêmes calculs** sont "extraits" lors des dépliages même si les boucles respectives des algorithmes sont, elles, bien différentes.

Somme d'une liste d'entiers

```
sum1([], 0).
sum1([X|Xs], N):- N = X + N1, sum1(Xs, N1).

sum2(Xs, Sum):- sumAcc(Xs, 0, Sum).
sumAcc([], Acc, Acc).
sumAcc([X|Xs], Acc, Sum):- Acc1 = X + Acc,
    sumAcc(Xs, Acc1, Sum).
```

Somme d'une liste d'entiers

```
sum1([], 0).
sum1([X|Xs], N):- N = X + N1, sum1(Xs, N1).

sum2([], 0).
sum2([X|Xs], Sum):- sumAcc(Xs, X, Sum).
sumAcc([], Acc, Acc).
sumAcc([X|Xs], Acc, Sum):- Acc1 = X + Acc,
    sumAcc(Xs, Acc1, Sum).
```

Somme d'une liste d'entiers

```
sum1([], 0).
sum1([X|Xs], X):- Xs = [A].
sum1([X|Xs], N):- N = X + N1, Xs = [A|Xss], N1 = A + N2,
    sum1(Xss, N2).

sum2([], 0).
sum2([X|Xs], X):- Xs = [A].
sum2([X|Xs], Sum):- Xs = [A|Xss], Acc1 = X + A,
    sumAcc(Xss, Acc1, Sum).
sumAcc([], Acc, Acc).
sumAcc([X|Xs], Acc, Sum):- Acc1 = X + Acc,
    sumAcc(Xs, Acc1, Sum).
```

Somme d'une liste d'entiers

```
sum1([], 0).
sum1([X|Xs], X):- Xs = [A].
sum1([X|Xs], N):- Xs = [A, B], N = A + B.
sum1([X|Xs], N):- N = X + N1, Xs = [A|Xss], N1 = A + N2,
    Xss = [B|Xsss], N2 = B + N3, sum1(Xsss, N3).

sum2([], 0).
sum2([X|Xs], X):- Xs = [A].
sum2([X|Xs], Sum):- Xs = [A, B], Sum = A + B.
sum2([X|Xs], Sum):- Xs = [A|Xss], Xss = [B|Xsss],
    Acc1 = X + A + B, sumAcc(Xsss, Acc1, Sum).
sumAcc([], Acc, Acc).
sumAcc([X|Xs], Acc, Sum):- Acc1 = X + Acc,
    sumAcc(Xs, Acc1, Sum).
```

Somme d'une liste d'entiers

Pourrait-on spécialiser sur un autre paramètre d'induction ?

Suite de Fibonacci

Suite de Fibonacci : $fibo(List)$. Deux versions :

- vérification élément par élément ;
- construction par génération d'une liste de même longueur.

Résultat après unfolding : Les clauses générées ne sont pas rigoureusement les mêmes mais elles sont **très similaires**.

Algorithmes de tri

Algorithmes de tri : $sort(List, SortedList)$. Deux versions :

- tri fusion ;
- tri à bulles.

Résultat après unfolding : Mêmes résultats que précédemment.
Mais l'unfolding du tri à bulles est bien plus gourmand en temps.

Factorielle

Factorielle : *facto*(*N*, *Value*). Trois versions :

- boucle ascendante ;
- boucle descendante ;
- récursivité.

Résultat après unfolding pour N = 5 :

```
facto1(A):- { A = 1* (1+1)* (1+1+1)*
    (1+1+1+1)* (1+1+1+1+1) }.
facto2(A):- { A = 5* (5-1)* (5-1-1)*
    (5-1-1-1)* (5-1-1-1-1) }.
facto3(A) :- { A = (5-1-1-1-1)* (5-1-1-1)*
    (5-1-1)* (5-1)*5 }.
```

Conclusions

Conclusions

- La notion de similarité entre algorithmes n'est pas gravée dans le marbre : *a priori*, il semblerait qu'une seule définition de similarité n'existe pas.

Conclusions

- La notion de similarité entre algorithmes n'est pas gravée dans le marbre : *a priori*, il semblerait qu'une seule définition de similarité n'existe pas.
- D'où l'utilité d'un framework paramétrable ayant plus ou moins de souplesse en fonction des transformations qu'il admet.

Conclusions

- La notion de similarité entre algorithmes n'est pas gravée dans le marbre : *a priori*, il semblerait qu'une seule définition de similarité n'existe pas.
- D'où l'utilité d'un framework paramétrable ayant plus ou moins de souplesse en fonction des transformations qu'il admet.
- Le tout en utilisant des transformations génériques elles-mêmes paramétrables.

Conclusions

- La notion de similarité entre algorithmes n'est pas gravée dans le marbre : *a priori*, il semblerait qu'une seule définition de similarité n'existe pas.
- D'où l'utilité d'un framework paramétrable ayant plus ou moins de souplesse en fonction des transformations qu'il admet.
- Le tout en utilisant des transformations génériques elles-mêmes paramétrables.
- En ne prenant aucune transformation comme acquise, même si certaines seront très probablement acceptées.

Merci pour votre attention !