Utilisation de Coq pour l'étude d'algorithmes

Yves Bertot

Introduction

- Structures de données
- Programmation de l'algorithme
- Description de propriétés des données
- Démonstration de cohérence

Partie 1: Structures de données

- Décrire les cas possibles dans les données
- Utiliser une approche structurée
  - Chaque cas a plusieurs composantes
  - Certaines composantes sont de même nature que le tout
- Diviser pour régner

Description des données par cas

```
Require Import List ZArith.
Open Scope Z_scope.

Print list.
Inductive list (A : Type) : Type :=
    nil : list A | cons : A -> list A -> list A
```

- Notation plus intuitive : `a::l` pour la séquence qui commence par a et continue avec `l`

Partie 2: Programmation par cas

```
Fixpoint insert (x : Z) (l : list Z) : list Z :=
 match l with
   nil => x::nil
 | a::l => if Zle_bool x a then x::a::l else a::insert x l
 end.

Fixpoint sort (l : list Z) : list Z :=
  match l with
    nil => nil
  | x::l => insert x (sort l)
  end.
```

Partie 3: Décrire les propriétés des listes

- Définir une fonction qui teste si une propriété est satisfaite
- Definir un prédicat minimal garantissant les propriétés recherchées
- Utiliser le langage d'expression logique

Exemple des listes triées

```
Inductive sorted (l : list Z) : Prop :=
  s0 : sorted nil
| s1 : forall x, sorted (x::nil)
| s2 : forall x y l, x < y -> sorted (y::l) ->
       sorted (x::y::l).
```

- Le prédicat qui satisfait ces trois propriétés ...
- et seulement ces propriétés

Partie 4: Démonstrations

- Propriétés intrinsèques des relations et prédicats
- Propriétés de l'algorithme défini vis-à-vis des relations

Exemple pour la permutation

```
Lemma trans_perm : forall l1 l2 l3,
  permutation l1 l2 -> permutation l2 l3 ->
  permutation l1 l3.
Proof.
intros l1 l2 l3 H H' x; rewrite (H x); apply H'.
Qed.
```

Propriétés de l'algorithme

```
Lemma insert_perm :
   forall x l, permutation (insert x l) (x::l).
Proof.
intros x; induction l.
  intros y; reflexivity.
simpl; case (Zle_bool x a).
  intros y; reflexivity.
intros y; simpl; case (Zeq_bool a y);
  rewrite (IHl y); simpl count; ring.
Qed.
```

Caractéristiques principales

- Le choix des structures de données guide la programmation
- Les algorithmes sont décrits en couvrant tous les cas
- Les propriétés donnent un point de vue différent sur l'algorithme
- La vérification des propriétés suit une approche similaire au test unitaire
- Les vérifications sont exhaustives, vis-à-vis des propriétés énoncées

Utilisation de Coq pour l'étude de langages de programmation

Yves Bertot

Introduction

- Description de langages de programmation
  - Syntaxe abstraite
- Description du comportement des programmes
  - Relations entre programmes, entrées et sorties,
  - Fonction exécutant les programmes
- Développements d'outils d'analyse ou de compilation
  - Vérification de propriétés
  - Traduction vers un autre langage
- Démonstration de propriétés

Partie 1: Syntaxe abstraite

- Les programmes sont composés d'instructions
- Les instructions sont composées d'instructions plus petites
- Les instructions de base sont composées d'expressions
- La forme importe peu, la structure importe beaucoup
- Décrire la structure avant toute chose

Un tout petit langage

```
Require Import ZArith String.
Open Scope Z_scope.

Inductive aexpr :=
  var (x : string) | add (e1 e2 : aexpr) | num (n : Z).

Inductive bexpr :=
  lt (e1 e2 : aexpr) | and (b1 b2 : bexpr) | true | false.

Inductive instr :=
  i_if (b : bexpr) (i1 i2 : instr)
| i_seq (i1 i2 : instr)
| while (b : bexpr) (i : instr)
| assign (x : string) (e : aexpr)
| skip.
```

Partie 2: Le comportement des programmes

- Sémantique opérationnelle
- Décrire un état ou environnement
  - Comment chaque forme d'instruction opère
- Relation entre des programmes et des environnements
- Plusieurs approches
  - Petits pas: chaque exécution élémentaire laisse un programme à continuer
  - Grands pas: décrire l'exécution jusqu'à la fin

Sémantique opérationnelle (petits pas)

```
Inductive sms : (instr * env) -> (instr * env) -> Prop :=
| sms_w : forall e b i,
  eval e b true ->
  sms (while b i, e) (i_seq i (while b i), e)
| sms_s1 : forall i1 i1' i2 e e',
  sms (i1, e) (i1', e') ->
  sms (i_seq i1 i2, i_seq i1' i2, e')
  ...
```

- Couvrir tous les cas possibles d'instructions et de comportement des sous-instructions
- Chaque règle logique reste très simple

Partie 3: Calculs sur des programmes annotés

- Calculs sur des assertions dans les programmes
  - Produire des formules logiques qu'il faut prouver (en Coq ou non)
- Analyse automatique de programme
  - Produire des assertions garanties

Exemple de calcul sur les assertions

```
Fixpoint pc (i:instr) (g : assertion) :=
  match i with
    assign x e => subst x e g
  | i_seq i1 i2 => pc i1 (pc i2 g)
  ...
  end.
```

- Calculer ce qu'il faut assurer avant pour garantir `g` après l'exécution
- Exemple sur l'affectation `x := x + 1`
  - Supposons que l'on veuille garantir `x < 3` après l'affectation
  - Donc il suffit que `x + 1 < 3` avant l'affectation
  - Remplacer la variable `x` par l'expression `x + 1`

Construction de formules logiques

```
Fixpoint vc (i : instr) (g : assertion) :=
  match i with
    while b annot i =>
      (not b & annot --> g)::(b & annot --> pc i annot)::
      vc i annot
    ...
   end.
```

- Les annotations dans les boucles doivent être satisfaite au début et à la fin de chaque exécution
- Peuvent être utilisée pour garantir la condition finale
- Il faut vérifier qu'elles sont *invariantes*

Exemple d'analyse statique

- Un intervalle pour chaque variable et chaque instruction
- Permet de prédire des dépassements de bornes
- Calcul symbolique.
- `x := x + 1`
  - Si `x` $\in$ `[0,10]` avant, alors `x` $\in$ `[1,11]` après
- `while (x < 10)` *fragment*
  - si `x` $\in$ `[0,100]` avant `while`,
    alors `x` $\in$ `[0,9]` avant *fragment*

Partie 4: Prouver qu'un analyseur est correct

- Un analyseur statique produit un programme annoté
- Le programme annoté peut ensuite être traité par un vérificateur d'annotations
- Il s'agit de démontrer que les formules logiques produites par ce dernier sont toujours valides

Preuve de compilateur

- La sémantique opérationnelle d'un langage fournit un moyen d'exécuter les programmes
- La compilation fournit un deuxième moyen
  - Repose sur la sémantique opérationnelle d'un second langage
- Preuve de correction du compilateur: montrer que les deux moyens d'exécuter sont équivalents

Lien avec le cours

- La syntaxe abstraite: les types inductifs de Coq
- Le comportement: les prédicats inductifs
- Les outils d'analyse: des fonctions récursives
- Les preuves de correction: des preuves dirigées par les buts

Utilisation de Coq pour les mathématiques

Yves Bertot

Introduction

- Une interaction avec les mathématiques en plein essor
- Quelques domaines clefs: cryptographie, arithmétique, algèbre
- Un souci de rigueur renforcé par l'ordinateur
- Vers l'utilisation de l'ordinateur pour la découverte de nouveaux résultats mathématiques

Quels objets mathématiques en Coq?

- Les nombres: entiers, rationnels
- Les objets algébriques: polynômes, vecteurs, matrices
- Les structures: groupes, anneaux
- Les objets axiomatiques: nombres réels, etc.

Quels résultats mathématiques?

- Théorie des nombres: critères de reconnaissances de grands nombres entiers, conjecture de Bertrand
- Graphes: théorème des 4 couleurs
- Géométrie: à partir des axiomes utilisés à l'école
  - Ou bien à l'aide d'outils algébriques
- Calcul de fonctions mathématiques: racine carrée, trigonométrie
- Topologie

Quelles difficultés?

- Changement de point vue très rapide dans l'esprit humain
  - Très difficile avec l'ordinateur
- Effort de rigueur parfois limitant dans la création scientifique
  - Rigueur parfois recherchée
- Faible quantité de résultats présents dans les bibliothèques

Quelles perspectives?

- Plus de mathématiques pour dire plus de choses pertinentes sur le logiciel
- Plus de logiciel mathématique pour faire plus de mathématiques
- Vérifier formellement de nouvelles catégories de logiciel
- Aborder de nouveaux domaines des mathématiques