Récursion générale

Yves Bertot

Introduction

- Récursion structurelle limitée
- Obligation de garantir que les calculs terminent
- Notion générale de relation bien fondée

Le théorème de récursion bien fondée

- `Fix :`
  ```
  forall (A : Type) (R : A -> A -> Prop),
  well_founded R ->
  forall P : A -> Type,
  (forall x : A, (forall y, R y x -> P y) -> P x) ->
  forall x : A, P x
  ```
- La partie en rouge décrit l'algorithme
- L'argument `x` est l'argument initial
- L'argument de type `forall y :A, R y x -> P y` sert pour les appels récursifs
  - `R y x` restreint les appels récursifs
- Les arguments `A` et `R` sont implicites

Exemple d'utilisation: la division des entiers naturels

- ```
  Inductive div_data (n m:nat) : Type :=
    cdd : forall q r, n = q * m + r -> r < m ->
          div_data n m.
  ```
- Algorithme par soustraction successives.
- Supposons l'existence des théorèmes suivants:
  ```
  th1 : forall n m, m <> 0 -> m <= n -> n - m < n.
  th2 : forall n m q r,
      m <= n -> n - m = q * m + r -> n = S q * m + r
  le_plus_minus :
      forall n m, n <= m -> m = n + (m - n)
  ```
  ```

La fonction de division

```
Definition div_nat : forall n m, m <> 0 -> div_data n m :=
  Fix lt_wf (fun n, forall m, m <> 0 -> div_data n m)
   (fun n f m h =>
      match le_lt_dec m n with
        left mlen =>
          let (q, r, qp, rp) :=
              f (n - m) (th1 _ _ h mlen) m h in
            cdd n m (S q) r (th2 _ _ _ _ mlen qp) rp
      | right nltm =>
          cdd n m 0 n (refl_equal _) nltm
      end).
```

Relations bien fondées

Yves Bertot

Introduction

- Pour le théorème de récursion `Fix`, il faut des relations bien fondées
- Notion définie par un prédicat inductif
- Une librairie prédéfinie pour prouver de nouvelles relations

Relations bien fondées

- Relation sans chaîne infinie décroissante
- Propriété capturée par le prédicat `well_founded`
- Définie à l'aide d'une propriété inductive
- Associée à un principe de récurrence
  - Fournit également des outils pour la logique

La notion d'accessibilité

- *Un point est accessible pour une relation R si tous ses prédécesseurs le sont.*
-
```
Inductive Acc
    (A : Type) (R : A -> A -> Prop) : A -> Prop :=
 Acc_intro:
    forall x, (forall y, R y x -> Acc A R y) ->
    Acc A R x.
```
- Par exemple, les éléments sans prédécesseurs sont accessibles
-
```
Definition well_founded
        (A : Type) (R : A -> A -> Prop) :=
  forall x : A, Acc A R x
```

Prouver de nouvelles relations bien fondées

- Preuve directe en passant par l'accessibilité
  - Par exemple, la preuve de `lt_wf`
- Utilisation de théorèmes d'héritage
  - si `R x y` $\equiv$ `R' (f x) (f y)` et `R'` est bien fondée alors `R` est bien fondée
    - En particulier si `R'` est `lt`, utiliser le théorème `ltof`
  - si `R` est un ordre lexicographique construit au dessus de relations bien fondées, alors `R` est bien fondée
- Plusieurs outils dans le package `Wellfounded`, disponibles après la commande
  `Require Import Wellfounded.`

La commande `Function`

Yves Bertot

Introduction

- Autoriser une programmation naturelle
- Satisfaire les contraintes a posteriori
- Eviter l'utilisation de types dépendants
- Laisser l'ordinateur trouver les preuves à faire
- Utiliser un principe de récurrence adapté

La commande `Function`: exemple de la factorielle

```
Require Import ZArith Recdef Zwf.
Open Scope Z_scope.

Function fact (x : Z) {wf (Zwf 0)} : Z :=
  if Zle_bool x 0 then 1 else x * fact (x - 1).
```

- Syntaxe proche de `Fixpoint`
- Indiquer une méthode (ici `wf`)
- Programmer sans faire référence à la terminaison

Obligations de preuve

- Pour la méthode `wf`, il faut montrer
  - que l'argument d'appel récursif est un prédécesseur
  - que la relation est bien-fondée
- Pour la méthode `measure`, il faut montrer
  - que la fonction de mesure décroit sur l'argument de l'appel récursif

Obligations pour fact

```
2 subgoals
  ============================
   forall x : Z, Zle_bool x 0 = false ->
     Zwf 0 (x - 1) x

subgoal 2 is:
 well_founded (Zwf 0)
  intros x h; generalize (Zle_cases x 0); rewrite h.
  unfold Zwf; omega.
apply Zwf_well_founded.
```

- Le but montre le contexte de l'appel récursif
  - Information importante pour montrer la décroissance

Principe de récurrence spécialisé

- Raisonnement par cas sur le calcul de la fonction
- Hypothèses de récurrence pour les appels récursifs

Exemple sur fact

```
Lemma fact_pos : forall x, 0 < fact x.
Proof.
intros x.
functional induction fact x.
2 subgoals
  e : Zle_bool x 0 = true
  ============================
   0 < 1

subgoal 2 is:
 0 < x * fact (x - 1)
```

Exemple sur fact (2)

```
omega.
  e : Zle_bool x 0 = false
  IHz : 0 < fact (x - 1)
  ============================
   0 < x * fact (x - 1)
assert (e' : x > 0).
  generalize (Zle_cases x 0); rewrite e; auto.
apply Zmult_lt_0_compat; omega.
```

La notion d'accessibilité

Yves Bertot

Introduction

- Capturer la notion “pas de chaîne infinie décroissante”
- Utilisation d’un type inductif: présentation constructive
- Intégrer la négation dans l’énoncé
- x n’appartient pas à une chaîne infinie décroissante si aucun de ces prédécesseur n’appartient
- Proposé par P. Aczel puis B. Nordström

Le prédicat inductif d'accessibilité

```
Inductive Acc
   (A : Type) (R : A -> A -> Prop) : A -> Prop :=
  Acc_intro : forall x,
      (forall y, R y x -> Acc A R y) -> Acc A R x.
```

- Un seul constructeur
- Deux composantes, dont la deuxième est une fonction
- Branchement potentiellement infini, limité par `R`

Remonter l'accessibilité

```
Definition Acc_inv A R x (h : Acc A R x) :
   forall y, R y x -> Acc A R y :=
  match h in Acc _ _ x
    return forall y, R y x -> Acc A R y with
    Acc_intro x' f => fun y hy => f y hy
  end.
```

- La preuve pour `Acc A R y` est obtenue par `f`
- Un sous-terme acceptable pour la récursion

Récursion sur l'accessibilité

```
Fixpoint Fix_F A (R : A -> A -> Prop)
  (P : A -> Type)
  (F : forall x, (forall y, R y x -> P y) -> P x)
  (x : A) (hx : Acc A R x) : P x :=
  F x
    (fun y  (hy : R y x) =>
         Fix_F A R P F y (Acc_inv A R x hx y hy)).
```

Typage pour la récursion

- La fonction
  `F : forall x, (forall y, R y x -> P y) -> P x`
  décrit un procédé de calcul
- Le premier argument décrit l'argument initial
- Le deuxième argument est une fonction avec une contrainte d'utilisation
- Le typage exprime les contraintes pour la récursion
- La fonction `Fix_F` est à la base de la fonction `Fix`