Introduction aux propriétés inductives en Coq

Yves Bertot

Introduction

- Une nouvelle approche pour définir des propriétés
- Inspirée des types de données inductifs
- Une approche systématique pour faire des preuves

Définir une propriété inductive

- Une propriété inductive est habituellement un prédicat
- Une fonction dont le type d'arrivée est `Prop`
  - exemple : `le : nat -> nat -> Prop`
- Utiliser la même approche que pour définir un type inductif
- Les constructeurs sont des moyens de prouver des instances de la propriété
  - ```
    Inductive le (n:nat) : nat -> Prop :=
    | le_n : le n n
    | le_S : forall m, le n m -> le n (S m).
    ```
    ```

Signification des constructeurs

- Les constructeurs sont des théorèmes vrais sur la propriété
- Toute instance vraie de la propriété est une conséquence des constructeurs
  - Propriété de minimalité
  - Forte similarité avec la programmation logique (Prolog)
  - Un principe de récurrence exprime cette propriété de minimalité

Preuve par récurrence sur les propriétés

- L'énoncé de `H` est une propriété inductive
- `elim H` ou `induction H`
- Un but pour chaque constructeur
- Nouvelles hypothèses: les conditions de ce constructeur
  - Les prémisses du constructeur
- Ajout d'une hypothèse de récurrence pour chaque prémisse récursive
  - Prémisses qui utilisent la propriété inductive

Exemple

- ```
  Inductive ieven : nat -> Prop :=
  | iev0 : ieven 0
  | iev2 : forall n, ieven n -> ieven (S (S n)).
  ```
- ```
  Fixpoint even (n:nat) : bool :=
  match n with
    0 => true
  | S p => negb (even p) end.
  ```

Exemple de preuve par récurrence

- Prouvons l'équivalence entre `ieven even`

```
Lemma ieven_even :
    forall x, ieven x -> even x = true.
intros x H; induction H as [ | y evy IHy].
```

- Premier but

```
========================
even 0 = true
reflexivity.
```

Exemple de preuve par récurrence (suite)

- Deuxième but

```
y : nat
evy : ieven y
IHy : even y = true
========================
even (S (S y)) = true
```

- Correspondance avec

```
iev2 : forall n, ieven n -> ieven (S (S n))
```

- IHy est l'hypothèse de récurrence

Exemple de preuve par récurrence (suite)

- Pour terminer la preuve:

```
simpl.
IHy : even y = true
=======================
negb (negb (even y)) = true
```

- `rewrite IHy; reflexivity.`

Prouver une instance de propriété inductive

- Dans l'autre sens, on n'utilise pas le principe de récurrence mais les constructeurs.
- `Lemma even_ieven :`
  `tab forall n, even n = true -> ieven n.`
- Script de résolution:

```
assert (forall n, (even n = true -> ieven n) /\
    (even (S n) = true -> ieven (S n))).
induction n;[split; intros;
    [apply iev0 | discriminate] |].
split; [intuition | simpl; rewrite negb_involutive;
intros; apply iev2; intuition].
firstorder.
```

Approche inductive des connecteurs logiques en Coq

Yves Bertot

La conjonction en Coq

- `Inductive and (A B : Prop) : Prop :=`
  `conj : A -> B -> and A B.`
- Notation `A /\ B` pour `and A B`
- Preuve par récurrence sur cette proposition inductive
  - Un seul constructeur : un seul sous-but,
  - Deux hypothèses comme le constructeur
  - `A` et `B` séparément!
  - `destruct` : traitement comme si c'était un type de données polymorphe

La disjonction en Coq

- ```
  Inductive or (A B : Prop) : Prop :=
  | or_introl : A -> or A B
  | or_intror : B -> or A B.
  ```
- Notation : `A \/ B` pour `or A B`
- Preuve par récurrence
  - Deux constructeurs : deux buts
  - Chaque but a une hypothèse supplémentaire `A` ou bien `B`
  - `destruct` comme pour un type de données polymorphe
  ```

La proposition False

- `Inductive False : Prop := .`
- Preuve par récurrence
  - Pas de constructeur, pas de but!
  - Si vous avez `False` dans les hypothèses
    Une preuve par récurrence sur cette hypothèse finit la preuve

Quantification existentielle

- `Inductive ex (A : Type) (P : A -> Prop) : Prop := ex_intro : forall x : A, P x -> ex A p.`
- Notation `exists x:A, P x` pour `@ex A (fun x => P x)`
- Preuve par récurrence
  - un seul constructeur: un seul but
  - deux hypothèses `x : A`, `h : P x`
  - `destruct` presque comme pour un type de données

Egalité

- `Inductive eq (A : Type) (x : A) : A -> Type := refl_equal : eq A x x.`
- Notation `x = y` pour `@eq _ x y`
- Preuve par récurrence
  - hypothèse `x = y`, but à prouver `C y`
  - un seul constructeur un seul but
  - doit prouver `C x`
  - Correspond à une réécriture remplacement de `y` par `x`

Propriétés inductives sous condition

- Quand une hypothèse a la forme `A1 -> ...-> An -> B`
- Quand `B` est une propriété inductive
- La tactique `elim H` ajoute `A1 An` aux buts à prouver
- Exemple `H : ~A`
  `elim H`
- Le nouveau but est `A`

Principes de récurrences de propriétés en Coq

Yves Bertot

Introduction

- Pour chaque proposition inductive un théorème exprime la minimalité
- Quantification sur toutes les propositions
- Exprime la minimalité de la propriété inductive
- Obtenu par un traitement systématique à partir de la définition inductive

Exemples de propriétés inductives

- ```
  Inductive ieven : nat -> Prop :=
  | iev0 : ieven 0
  | iev2 : forall n, ieven n -> ieven (S (S n)).
  ```
- ```
  Inductive le (n : nat) : nat -> Prop :=
  | le_n : le n n | le_S : forall m, le n m -> le n
  (S m).
  ```

En-tête du principe de récurrence

- Quantification universelle sur une propriété
- Même type que la propriété inductive considérée
- Dérivé de l'entête de la définition inductive
  - `Inductive ieven : nat -> Prop :=`
    `ieven_ind : forall P : nat -> Prop ...`
- Prise en compte des paramètres stables
  - `Inductive le (n : nat) : nat -> Prop :=`
    `le_ind : forall n : nat, forall P : nat -> Prop`
    `...`

Epilogue du principe de récurrence

- Exprime que sous les bonnes conditions, la propriété quantifiée universellement est une conséquence de la propriété inductive
- L'ensemble des valeurs satisfaisant la propriété inductive est plus petit
  - forme `forall` x_1 `...`x_n`, A` x_1 `...`x_n `-> P` x_1 `...`x_n
  - Quantification sur les arguments non paramètres
  - exemple pour `ieven_ind`
    `forall (P : nat -> Prop), ...-> forall n, ieven n -> P n`
  - exemple pour `le_ind`
    `forall (n:  nat)(P : nat -> Prop), ...-> forall m, le n m -> P m`

Prémisses par constructeurs

- Exprimer que la propriété doit être satisfaite lorsque les conditions sont réunies pour que ce constructeur s'applique
- Pour chaque quantification universelle dans le type du constructeur
  ajouter une quantification universelle dans la prémisse
- Pour chaque implication `B -> ...`
  ajouter une implication `B -> ...`
- Pour chaque implication `A` e_1 `...`e_k `-> ...`
  deux implications `A` e_1 `...`e_k `-> P` e_{j+1} `...`e_k `-> ...`
  - les j premiers arguments sont les paramètres
- La formule finale du constructeur est de la forme `A` e_1 `...`e_k
  La formule finale de la prémisse est `P` e_{j+1} `...`e_k

Exemples

- Constructeurs de `ieven`
  - `iev0 : ieven 0`
    `P 0`
  - `iev2 : forall n, ieven n -> ieven (S (S n))`
    `(forall n, ieven n -> P n -> P (S (S n)))`
- Constructeur de `le`
  - `le_S : forall m, le n m -> le n (S m)`
    `(forall m, le n m -> P m -> P (S m))`

Exemple complet

- ```
  Inductive ieven : nat -> Prop :=
  | iev0 : ieven 0
  | iev2 : forall n, ieven n -> ieven (S (S n)).
  ```
- ```
  ieven_ind : forall P : nat -> Prop :=
  P 0 ->
  (forall n, ieven n -> P n -> P (S (S n))) ->
  forall n, ieven n -> P n
  ```

Inversion des propriétés inductives

Yves Bertot

Traitement par cas sur les prédicats inductifs

- Comparer les arguments d'un prédicat inductif en hypothèse avec les constructeurs
- comprendre les constructeurs qui ont dû être utilisés
- Tirer des conclusions sur les prémisses des constructeurs
- Lire les implications *à l'envers*

Fonctionnement des preuves par récurrence

- Forme générale des théorèmes de récurrence

```
forall P : A -> Prop,
...
forall x, pred x -> P x
```

- Utilisé sur une hypothèse `H : pred e`
  - Cherche toutes les instances de `e` dans le but,
  - Abstrait pour obtenir `P`,
  - Ne regarde pas la forme de `e`,
  - Oubli d'information

Exemple de mauvaise preuve par récurrence

- ```
  Inductive ieven : nat -> Prop :=
  | iev0 : ieven 0
  | iev2 : forall n, ieven n -> ieven (S (S n)).
  ```
- ```
  Lemma iev1 : ieven 1 -> False.
  intros H; induction H.
  ```
- Propriété P pour le principe de récurrence

  ```
  fun x => False
  ```
- Deux buts engendrés le premier est `P 0`
  - C'est à dire `False` sans hypothèse supplémentaire
- `1` n'apparaît pas dans le but avant la récurrence
- Oubli de l'information que c'est `1`

Conclusion intermédiaire

- Pour une preuve par récurrence sur un prédicat inductif, il faut que les arguments soient des variables
- Dans l'exemple l'hypothèse `H` a l'énoncé `ieven 1`
  - 1 n'est pas une variable

La tactique `inversion`

- Observe la forme des arguments
- Rejette les cas incompatibles : égalités discriminables
- Ajoute les prémisses correspondantes au constructeur appliqué
- Effet d'inversion
  - Parce que la conclusion est satisfaite, la prémisse doit l'être
  - Illusion : seulement lorsque les constructeurs sont exclusifs

La tactique `inversion` sur un exemple

- `ieven 1 -> False`
  - `intros H; inversion H`
  - Reconnaît que `0 = 1` est contradictoire
  - Même chose pour un `x` arbitraire et `S (S x) = 1`
  - Donc aucun constructeur ne s'applique: la preuve est directe.
- `forall x, ieven (S (S x)) -> ieven x`
  - `intros x H; inversion H.`
  - Reconnaît que le premier constructeur ne s'applique pas
  - Ajoute `ieven x` dans les hypothèses, parce que seul le deuxième constructeur s'applique
  - La preuve peut être conclue par `assumption`