IF107 - Preuve de programmes récursifs

Frédéric Herbreteau
frederic.herbreteau@bordeaux-inp.fr

31 décembre 2024

1. Programmes récursifs & spécification

Preuve de programmes transformationnels, récursifs

```
function factorial (n : int) returns (f : int)
    if n == 0:
        f := 1
    else:
        g := factorial(n-1)
        f := n * g
```

- On souhaite que la fonction `factorial` :
  - **termine** pour toute entrée `n`
  - retourne **correctement** la valeur `n`!
- Objectif : **techniques de preuve** pour la **terminaison** et la **correction** pour les **programmes récursifs**
- Langage de programmation : variables, affectation, **if**, **function**, **appels récursifs**, etc.

Spécification d'un programme récursif

```
function factorial (n : int) returns (f : int)
    if n == 0:
        f := 1
    else:
        g := factorial(n-1)
        f := n * g
```

- Spécification :

  PRECOND : $n \in \mathbb{N}$

  POSTCOND : $f = n!$

Exemple

La fonction `factorial` :

- termine-t-elle pour toute entrée $\mathtt{n} \in \mathbb{N}$?
- retourne-t-elle $\mathtt{f} = \mathtt{n}!$ pour toute entrée $\mathtt{n} \in \mathbb{N}$?

2. Preuve de terminaison

Exécutions d'un programme récursif

```
1  function factorial (n : int) returns (f : int)
2      if n == 0:
3          f := 1
4      else:
5          g := factorial(n-1)
6          f := n * g
```

Exemple

- Exécution pour $n = 2$
- Exécution pour $n = -1$

Rappel : terminaison de programme

Définition

Un programme **termine** si chacune de ses exécutions est finie.

Exemple

La fonction `factorial` ne termine pas (PRECOND : *true*)

Théorème

*Le problème qui consiste à décider si un programme récursif P termine est **indécidable**.*

Méthodologie : preuve de terminaison

On veut montrer que **toute exécution d'un programme récursif est finie (ou termine)** :

1. trouver un **variant**, c'est à dire une expression formée à partir des variables du programme
2. montrer que :
   2.1 le variant est **strictement décroissant par appel récursif**
   2.2 le variant est **à valeur dans un ensemble bien ordonné**

   $\rightarrow$ les valeurs du variant forment donc une **suite strictement décroissante dans un ensemble bien ordonné** le long de **toute exécution du programme**

   $\rightarrow$ cette suite est donc nécessairement **finie** (théorème des ensembles bien ordonnés)
3. en conclure que **toute exécution du programme est finie**

Exemple : terminaison de `factorial`

```
1  function factorial (n : int) returns (f : int)
2      if n == 0:
3          f := 1
4      else:
5          g := factorial(n-1)
6          f := n * g
```

PRECOND : $n \in \mathbb{N}$ POSTCOND : $f = n!$

Exemple

- Variant :
- Strictement décroissant par appel récursif :
- À valeurs dans un ensemble bien ordonné :

3. Preuve de correction

Rappel : correction des programmes

Définition

Un programme est **correct** pour une spécification (PRE, POST) si **pour chaque exécution finie** $s_0 s_1 \cdots s_n$, si PRE est vraie en s_0, alors POST est vraie en s_n.

Exemple

```
function factorial (n : int) returns (f : int)
    if n == 0:
        f := 1
    else:
        g := factorial(n-1)
        f := n * g
```

PRECOND : $n \in \mathbb{N}$ POSTCOND : $f = n!$

Théorème

Le problème qui consiste à décider si un programme P est correct pour spécification (PRECOND, POSTCOND) est ***indécidable****.*

Preuve par induction sur les appels récursifs

Théorème

Soit P un programme récursif, et $(PRE, POST)$ une spécification.
Si :

1. *PRE est vrai* ***avant chaque appel récursif***
2. *et, en* ***supposant*** *$POST$* ***vrai après chaque appel récursif****, alors $POST$ est vrai* ***à la fin de chaque exécution***

Alors, P est correct pour la spécification $(PRE, POST)$.

Démonstration.

□

Méthodologie : preuve de correction

On veut montrer que **toute exécution d'un programme récursif est correcte** pour une spécification ($PRECOND, POSTCOND$) :

1. Montrer que :
   - 1.1 Toute exécution **sans appel récursif** est correcte
   - 1.2 $PRECOND$ est **vraie avant tout appel récursif**
   - 1.3 Si $POSTCOND$ est vrai après chaque appel récursif, alors $POSTCOND$ **est vrai à la fin de chaque exécution**
2. En conclure que **toute exécution du programme est correcte** pour ($PRECOND, POSTCOND$)

Exemple : terminaison de `factorial`

```
1  function factorial (n : int) returns (f : int)
2      if n == 0:
3          f := 1
4      else:
5          g := factorial(n-1)
6          f := n * g
```

PRECOND : $n \in \mathbb{N}$ POSTCOND : $f = n!$

Exemple

Montrer que :

1. toute exécution **sans appel récursif** est correcte :
2. *PRECOND* est **vraie avant tout appel récursif** :
3. si *POSTCOND* est vrai après chaque appel récursif, alors *POSTCOND* **est vrai à la fin de chaque exécution** :

4. Exemple : arbres binaires de recherche

Arbre binaire de recherche

Type inductif "arbre binaire" Tree :

- Nil est un arbre binaire
- $\text{Node}(n, l, r)$ est un arbre binaire pour tout $n \in \mathbb{N}$ et $l, r \in \text{Tree}$

$$\mathit{value}(\text{Node}(n, l, r)) = n \qquad \mathit{left}(\text{Node}(n, l, r)) = l \qquad \mathit{right}(\text{Node}(n, l, r)) = r$$

Fonction "appartenance" :

- $\mathit{in}(x, \text{Nil}) \triangleq \mathit{false}$
- $\mathit{in}(x, \text{Node}(n, l, r)) \triangleq (x = n) \vee \mathit{in}(x, l) \vee \mathit{in}(x, r)$

Prédicat "Arbre binaire de recherche" :

- $\mathit{abr}(\text{Nil}) \triangleq \mathit{true}$
- $\mathit{abr}(\text{Node}(n, l, r)) \triangleq \forall x.(\mathit{in}(x, l) \implies x < n) \wedge \forall y.(\mathit{in}(x, r) \implies n < y) \wedge \mathit{abr}(l) \wedge \mathit{abr}(r)$

Recherche récursive : terminaison

```
1  function search (x : int, t : Tree) returns (found : bool)
2      if t = Nil:
3          found := false
4      elif value(t) = x:
5          found := true
6      elif x < value(t):
7          found := search(x, left(t))
8      else:
9          found := search(x, right(t))
```

PRECOND : $abr(t)$ POSTCOND : $found = true \iff in(x, t)$

Exemple

- variant :
- strictement décroissant :
- dans un ensemble bien ordonné :

Recherche récursive : correction

```
1  function search (x : int, t : Tree) returns (found : bool)
2      if t = Nil:
3          found := false
4      elif value(t) = x:
5          found := true
6      elif x < value(t):
7          found := search(x, left(t))
8      else:
9          found := search(x, right(t))
```

PRECOND : $abr(t)$ POSTCOND : $found = true \iff in(x, t)$

Exemple

- POSTCOND vrai sans appel récursif :
- PRECOND vrai avant appels récursifs :
- POSTCOND vrai avec appel récursif :

Recherche itérative : terminaison

```
1  function search (x : int, t : Tree) returns (found : bool)
2      t2 := t; found := false;
3      while found = false and t2 ≠ Nil:
4          if value(t2) = x:
5              found := true
6          elif x < value(t2):
7              t2 := left(t2)
8          else:
9              t2 := right(t2)
```

PRECOND : $abr(t)$ POSTCOND : $found = true \iff in(x, t)$

Exemple

- variant :
- strictement décroissant :
- dans un ensemble bien ordonné :

Recherche itérative : correction

```
1  function search (x : int, t : Tree) returns (found : bool)
2      t2 := t; found := false;
3      while found = false and t2 ≠ Nil:
4          if value(t2) = x:
5              found := true
6          elif x < value(t2):
7              t2 := left(t2)
8          else:
9              t2 := right(t2)
```

PRECOND : $abr(t)$ POSTCOND : $found = true \iff in(x, t)$

Exemple

- invariant :
- avant la boucle :
- préservation :
- POSTCOND est vrai :

5. Conclusion

La preuve pour programmer

Cadres de raisonnement :

- Logique
- Induction
- Ensembles bien fondés
- Preuve de programmes

Penser “preuve” à la conception :

- Quel **objet** mon programme manipule-t-il ? (type inductif, etc)
- Quel **variant** ? (entier naturel, type inductif, ensemble bien fondé, etc.)
- Quel **invariant de cet objet** mon programme doit-il préserver ?