LIFLC – Logique classique

CM7 – Preuves de programme

Licence informatique UCBL – Automne 2017–2018

https://liris.cnrs.fr/ecoquery/dokuwiki/doku.php?id=enseignement:logique:start

Prouver un programme

Objectif : assurer qu'il fait ce que l'on veut

- Bien **spécifier** → logique
- Connaître le langage → sémantique
- Prouver → systèmes de déduction syntaxique

Prouver un programme

Objectif : assurer qu'il fait ce que l'on veut

- Bien **spécifier** $\rightarrow$ logique
- Connaître le langage $\rightarrow$ sémantique
- Prouver $\rightarrow$ systèmes de déduction syntaxique

Software Foundations

Livre en ligne sur les langages de programmation

Écrit en Coq + commentaires ← livre “exécutable”

Sert de base à ce cours

https://softwarefoundations.cis.upenn.edu/

Un langage impératif : Imp

Mini langage de programmation

- Impératif
- Expression arithmétiques et booléenne

Exemple

```
Z ::= X;;
Y ::= 1;;
WHILE not (Z = 0) DO
  Y ::= Y * Z;;
  Z ::= Z - 1
END
```

Syntaxe abstraite : expressions

Signature : expressions arithmétiques (aexp) :

$\mathcal{C}_{\texttt{aexp}}$: entiers naturels représentés en base 10

$\mathcal{F}_{\texttt{aexp}}$: $\{\texttt{APlus}/2, \texttt{AMinus}/2, \texttt{AMult}/2\}$

Signature : expressions booléennes (bexp) :

$\mathcal{C}_{\texttt{bexp}}$: $\{\texttt{BTrue}, \texttt{BFalse}\}$

$\mathcal{F}_{\texttt{bexp}}$: $\{\texttt{BEq}/2, \texttt{BLe}/2, \texttt{BNot}/1, \texttt{BAnd}/2\}$

⚠ $\{\texttt{BEq}/2, \texttt{BLe}/2$: termes particuliers : aruments construits sur aexp

→ En Coq

Sémantique des expressions arithmétiques

Fonction d'évaluation =

- Évaluation (standard) de termes
- à interprétation I fixée

I pour `aexp` :

- Constantes : $I(n) = n$
- $I(\texttt{APlus}) = n_1, n_2 \mapsto n_1 + n_2$
- $I(\texttt{AMinus}) = n_1, n_2 \mapsto n_1 - n_2$
- $I(\texttt{AMult}) = n_1, n_2 \mapsto n_1 \times n_2$

I fixée : $\texttt{aeval}(\zeta)(e) = \mathit{eval}(I, \zeta)(e)$

→ En Coq

Sémantique des expressions booléennes

I pour `bexp` :

- Constantes : $I(\texttt{BTrue}) = \mathit{true}$
  $I(\texttt{BFalse}) = \mathit{false}$
- $I(\texttt{BEq}) : (n_1, n_2) \mapsto \begin{cases} \mathit{true} \text{ si } n_1 = n_2 \\ \mathit{false} \text{ sinon} \end{cases}$
- $I(\texttt{BLe}) : (n_1, n_2) \mapsto \begin{cases} \mathit{true} \text{ si } n_1 \leq n_2 \\ \mathit{false} \text{ sinon} \end{cases}$
- $I(\texttt{BNot}) : b \mapsto \begin{cases} \mathit{true} \text{ si } b = \mathit{false} \\ \mathit{false} \text{ sinon} \end{cases}$
- $I(\texttt{BLe}) : (b_1, b_2) \mapsto \begin{cases} \mathit{true} \text{ si } b_1 = \mathit{true} \text{ et } b_2 = \mathit{true} \\ \mathit{false} \text{ sinon} \end{cases}$

I fixée : $\texttt{beval}(\zeta)(e) = \mathit{eval}(I, \zeta)(e)$

→ En Coq

Remarques

- Mélange d'expressions booléennes et arithmétiques
  - `APlus(BTrue, x)` ? ? ? ← interdit par le typage Coq
- Pas de variables booléennes

Optimisation : Supprimer les additions de 0

Idée : récrire l'arbre de syntaxe pour supprimer les additions à 0

Fonction `optimize_0plus` :

- $n \mapsto n$ si $n \in \mathcal{N}$
- $x \mapsto x$ si $x \in \mathcal{V}$
- $\texttt{APlus}(0, e_2) \mapsto \texttt{optimize\_0plus}(e_2)$
- $\texttt{APlus}(e_1, e_2) \mapsto$
  $\texttt{APlus}(\texttt{optimize\_0plus}(e_1), \texttt{optimize\_0plus}(e_2))$
- $\texttt{AMinus}(e_1, e_2) \mapsto$
  $\texttt{AMinus}(\texttt{optimize\_0plus}(e_1), \texttt{optimize\_0plus}(e_2))$
- $\texttt{AMult}(e_1, e_2) \mapsto$
  $\texttt{AMult}(\texttt{optimize\_0plus}(e_1), \texttt{optimize\_0plus}(e_2))$

→ En Coq

Consistence de l'optimisation

A-t-on le droit de faire cette optimisation ?
Est-ce que cela change le résultat ?

Théorème

Pour toute expression e dans `aexp` *:*

$$\texttt{aeval}(e) = \texttt{aeval}(\texttt{optimize\_0plus}(e))$$

→ En Coq

Instructions

Ensemble inductif des programmes `prog`

- `CSkip`
- `CAss`(x, e) si $x \in \mathcal{V}$ et $e \in$ `aexp`
- `CSeq`(p_1, p_2) si p_1 et p_2 sont des programmes
- `CIf`(b, p_1, p_2) si $b \in$ `bexp` et si p_1 et p_2 sont des programmes
- `CWhile`(b, p) si $b \in$ `bexp` et si p est un programme.

Grammaire (*c.f.* LIFLF) :

$C \to$ `SKIP | x ::= A | C ;; C | IF B THEN C ELSE C FI`
`| WHILE B DO c END`
$A \to \ldots$ (expressions arithmétiques)
$B \to \ldots$ (expressions booléennes)

Exemple

```
Z ::= X;;
Y ::= 1;;
WHILE not (Z = 0) DO
  Y ::= Y * Z;;
  Z ::= Z - 1
END
```

$\rightsquigarrow$

```
CSeq(CSeq(CAss(Z, X),
          CAss(Y, 1)),
     CWhile(BNot(BEq(Z, 0)),
            CSeq(CAss(Y, AMult(Y, Z)),
                 CAss(Z, AMinus(Z, 1)))
            )
     )
```

Sémantique opérationnelle

⚠ Boucle `WHILE` $\rightsquigarrow$ non-terminaison ⚠

Pas de fonction d'évaluation
(non définie sur les programmes qui ne terminent pas)

Sémantique *opérationnelle* (structurée à petits pas) :

- système de déduction syntaxique
- Jugement : permet de dire si un programme P transforme
  - un état ζ (*i.e.* une valuation) de départ
  - en un état ζ' d'arrivée
- Notation : $P : \zeta \rightsquigarrow \zeta'$

Règles de la sémantique opérationnelle - SKIP, : :=, ; ;

$$\frac{}{\texttt{CSkip} : \zeta \rightsquigarrow \zeta} \ (Imp_{Skip})$$

$$\frac{}{\texttt{CAss}(x, a) : \zeta \rightsquigarrow \zeta[x := n]} \ (Imp_{Ass}) \quad \text{si } \texttt{aeval}(\zeta)(a) = n$$

$$\frac{P_1 : \zeta \rightsquigarrow \zeta' \quad P_2 : \zeta' \rightsquigarrow \zeta''}{\texttt{CSeq}(P_1, P_2) : \zeta \rightsquigarrow \zeta''} \ (Imp_{Seq})$$

Règles de la sémantique opérationnelle - IF

$$\frac{P_1 : \zeta \rightsquigarrow \zeta'}{\texttt{CIf}(b, P_1, P_2) : \zeta \rightsquigarrow \zeta'}\ (Imp_{IfTrue}) \qquad \text{si } \texttt{beval}(\zeta)(b) = \mathit{true}$$

$$\frac{P_2 : \zeta \rightsquigarrow \zeta'}{\texttt{CIf}(b, P_1, P_2) : \zeta \rightsquigarrow \zeta'}\ (Imp_{IfFalse}) \qquad \text{si } \texttt{beval}(\zeta)(b) = \mathit{false}$$

Règles de la sémantique opérationnelle - WHILE

$$\frac{}{\texttt{CWhile}(b, P) : \zeta \rightsquigarrow \zeta} \ (Imp_{WhileFalse}) \quad \text{si } \texttt{beval}(\zeta)(b) = \mathit{false}$$

$$\frac{P : \zeta \rightsquigarrow \zeta' \quad CWhile(b, P) : \zeta' \rightsquigarrow \zeta''}{\texttt{CWhile}(b, P) : \zeta \rightsquigarrow \zeta''} \ (Imp_{WhileTrue})$$

$$\text{si } \texttt{beval}(\zeta)(b) = \mathit{true}$$

En Coq

$P : \zeta \rightsquigarrow \zeta'$ noté `P / st \\st'`

Règle de déduction :

$$\frac{\mathit{premisse}_1 \quad \mathit{premisse}_2}{\mathit{conclusion}}$$

notée :

```
premisse1 ->
premisse2 ->
conclusion
```