Sémantique des langages de programmation
Mastère PLMT
solution de l'examen

Yves Bertot

Janvier 2007

1 Démonstration pour τ_b

Nous démontrons d'abord une propriété pour la fonction τ_b

$$forall \rho\ b\ v. \rho \vdash b \rightarrow v \Rightarrow \rho \vdash \tau_b(b) \rightarrow v.$$

Considérons une expression boolénne b fixée, cette expression est de la forme $e_1 < e_2$. Supposons que le jugement $\rho \vdash b \rightarrow v$ soit prouvable, alors l'une des deux règles d'évaluation pour les expressions booléennes a été utilisée. Dans le premier cas, il s'agit de la règle suivante:

$$\frac{\rho \vdash e_1 \rightarrow v_1 \quad \rho \vdash e_2 \rightarrow v_2 \quad v_1 < v_2}{\rho \vdash e_1 < e_2 \rightarrow \mathsf{true}}$$

et v vaut true. Si cette règle a été utilisée, alors $\rho \vdash e_1 \rightarrow v_1$ et $\rho \vdash e_2 \rightarrow v_2$ ont été prouvés. D'après la propriété de τ, on peut déduire les jugements $\rho \vdash \tau(e_1) \rightarrow v_1$ et $\rho \vdash \tau(e_2) \rightarrow v_2$. Donc la dérivation suivante est prouvable.

$$\frac{\rho \vdash \tau(e_1) \rightarrow v_1 \quad \rho \vdash \tau(e_2) \rightarrow v_2 \quad v_1 < v_2}{\rho \vdash \tau(e_1) < \tau(e_2) \rightarrow \mathsf{true}}$$

comme $\tau(e_1) < \tau(e_2) = \tau_b(e_1 < e_2) = \tau(b)$ et $v = \mathsf{true}$, on a bien construit une dérivation qui permet de prouver $\rho \vdash \tau_b(b) \rightarrow v$. Le deuxième cas se démontre exactement de la même manière. La propriété pour τ_b est acquise.

2 Démonstration pour τ_i

Montrons maintenant la propriété pour τ_i. Fixons ρ, i, et ρ' et supposons que le jugement $\rho \vdash i \rightsquigarrow \rho'$ est prouvable. Nous voulons prouver $\rho \vdash \tau_i(i) \rightsquigarrow \rho'$. Nous le démontrons par récurrence sur la dérivation δ qui permet de prouver $\rho \vdash i \rightsquigarrow \rho'$. Puisqu'il s'agit d'une preuve par récurrence, nous pourrons utiliser une hypothèse de récurrence chaque fois que l'occasion se présentera.

La dérivation δ qui prouve $\rho \vdash i\rho'$ utilise l'une des cinq règles de la spécification sémantique, nous avons donc 5 cas.

2.1 Règle de skip

Dans le premier cas, la règle utilisée est la règle suivante:

$$\frac{}{\rho \vdash skip \rightsquigarrow rho.}$$

dans ce cas, $rho' = \rho$ et $i = \mathsf{skip} = \tau_i(i)$. La même règle permet de démontrer $\rho \vdash \tau_i(i) \rightsquigarrow \rho'$. Ce cas est résolu.

2.2 Règle d'affectation

Dans le deuxième cas, la règle utilisée est la règle suivante:

$$\frac{\rho \vdash e \rightarrow v \quad \rho \vdash x, v \mapsto \rho'}{\rho \vdash x :=e \rightsquigarrow \rho'}.$$

Dans ce cas, $i = x{:=}e$ et $\tau_i(i) = x{:=}\tau(e)$ et le jugement $\rho \vdash e \rightarrow v$ D'après la propriété de τ, $\rho \vdash \tau(e) \rightarrow v$ est également prouvable et l'on peut donc construire une dérivation qui termine de la façon suivante:

$$\frac{\rho \vdash \tau(e) \rightarrow v \quad \rho \vdash x, v \mapsto \rho'}{\rho \vdash \tau_i(x{:=}e) \rightsquigarrow \rho'}.$$

Ce cas est également résolu.

2.3 Règle de séquence

Dans le troisième cas, la règle utilisée est la règle suivante:

$$\frac{\rho \vdash i_1 \rightsquigarrow \rho_1 \quad \rho \vdash i_2 \rho'}{\rho \vdash i_1{;}i_2 \rightarrow rho'}$$

Dans ce cas, $i = i_1{;}i_2$ et $\tau_i(i) = \tau_i(i_1){;}\tau_i(i_2)$. Les dérivations $\rho \vdash i_1 \rightsquigarrow \rho_1$ et $\rho \vdash i_2 \rightsquigarrow \rho_2$ sont plus petites que la dérivation complète et l'on dispose donc pour chacune d'entre elle d'une hypothèse de récurrence. Ces deux hypothèses de récurrences assurent que les jugements suivants sont prouvables: $\rho \vdash \tau_i(i_1) \rightsquigarrow \rho_1$ et $\rho \vdash \tau_i(i_2) \rightsquigarrow \rho_2$. On peut alors utiliser les deux hypothèses de récurrence et la règle pour la séquence pour fabriquer une dérivation de la forme suivante:

$$\frac{\rho \vdash \tau_i(i_1) \rightsquigarrow \rho_1 \quad \rho \vdash \tau_i(i_2) \rho'}{\rho \vdash \tau_i(i_1){;}\tau_i(i_2) \rightarrow rho'}$$

Comme $\tau_(i) = \tau_i(i_1){;}\tau_i(i_2)$, ceci suffit à résoudre ce cas.

2.4 Règle de boucle (cas true)

Dans le quatrième cas, la règle utilisée est la suivante:

$$\frac{\rho \vdash b \rightarrow \mathsf{true} \quad \rho \vdash i_i \rightsquigarrow \rho_1 \quad \rho_1 \vdash \mathsf{while}\ b\ \mathsf{do}\ i_1\ \mathsf{done} \rightsquigarrow \rho'}{\rho \vdash \mathsf{while}\ b\ \mathsf{do}\ i_1\ \mathsf{done} \rightsquigarrow \rho'}$$

Dans ce cas $i = \mathsf{while}\ b\ \mathsf{do}\ i_1\ \mathsf{done}$ et $\tau_i(i) = \mathsf{while}\ \tau_b(b)\ \mathsf{do}\ \tau_i(i_1)\ \mathsf{done}$ et nous disposons de deux hypothèses de récurrences pour les deux sous dérivations correspondant à la deuxième et à la troisième prémisse. Ces hypothèses de récurrence assurent que les deux jugements suivants sont prouvables:

$$\rho \vdash \tau_i(i) \rightsquigarrow \rho_1 \qquad \rho_1 \vdash \tau_i(\mathsf{while}\ b\ \mathsf{do}\ i_1\ \mathsf{done} \rightsquigarrow \rho'$$

D'après la définition de tau_i, le deuxième de ces jugements nous donne aussi:

$$\rho_1 \vdash \mathsf{while}\ tau_b(b)\ \mathsf{do}\ \tau_i(i_1)\ \mathsf{done} \rightsquigarrow \rho'$$

Enfin, la propriété que nous avons prouvée pour tau_b assure que nous avons également:

$$\rho \vdash\ \tau_b(b) \rightarrow \mathsf{true}.$$

Avec tous ces jugements et la règle de sémantique pour la boucle, nous pouvons donc reconstruire la dérivation suivante:

$$\frac{\rho \vdash \tau_b(b) \rightarrow \mathsf{true} \quad \rho \vdash \tau_i(i_i) \rightsquigarrow \rho_1 \quad \rho_1 \vdash \mathsf{while}\ \tau_b(b)\ \mathsf{do}\ \tau_i(i_1)\ \mathsf{done} \rightsquigarrow \rho'}{\rho \vdash \mathsf{while}\ \tau_b(b)\ \mathsf{do}\ \tau_i(i_1)\ \mathsf{done} \rightsquigarrow \rho'}$$

Comme $\tau_i(i) = \mathsf{while}\ \tau_b(b)\ \mathsf{do}\ \tau_i(i_1)\ \mathsf{done}$, ceci suffit pour conclure ce cas.

2.5 Règle de boucle (cas false)

Dans le cinquième cas, la dérivation utilise la règle suivante:

$$\frac{\rho \vdash b \rightarrow \mathsf{false}}{\rho \vdash \mathsf{while}\ b\ \mathsf{do}\ i_1\ \mathsf{done} \rightsquigarrow \rho}$$

Dans ce cas, $i = \mathsf{while}\ b\ \mathsf{do}\ i_1\ \mathsf{done}$, $\tau_i(i) = \mathsf{while}\ \tau_b(b)\ \mathsf{do}\ \tau_i(i_1)\ \mathsf{done}$ et $\rho' = \rho$ et le jugement $\rho \vdash b \rightarrow \mathsf{false}$ est forcément prouvable. D'après la propriété que nous avons établie pour τ_b, nous avons également $\rho \vdash \tau_b(b) \rightarrow \mathsf{false}$ et nous pouvons reconstruire une dérivation ayant la forme suivante:

$$\frac{\rho \vdash \tau_b(b) \rightarrow \mathsf{false}}{\rho \vdash \mathsf{while}\ \tau_b(b)\ \mathsf{do}\ \tau_i(i_1)\ \mathsf{done} \rightsquigarrow \rho}$$

Comme $tau_i(i) = \mathsf{while}\ \tau_b(b)\ \mathsf{do}\ \tau_i(i_1)\ \mathsf{done}$, ceci suffit pour conclure ce cas.

Nous avons donc démontré $\rho \vdash \tau_i(i) \rightsquigarrow \rho'$ dans les cinq cas possibles. CQFD

3 Preuves en Coq

```
Require Import string ZArith.

Section ex_tau.
(*First let's summarize the semantic description of the language,
or at least, the needed part. *)
```

```
Variable aexpr : Type.
Variable aeval : list(string*Z)->aexpr->Z->Prop.

Inductive bexpr : Type :=
  blt(e1 e2:aexpr).

Inductive beval : list(string*Z)->bexpr->bool->Prop:=
  blt1 : forall r e1 e2 v1 v2,
    aeval r e1 v1 -> aeval r e2 v2 -> v1 < v2 ->
    beval r (blt e1 e2) true
| blt2 : forall r e1 e2 v1 v2,
    aeval r e1 v1 -> aeval r e2 v2 -> v2 <= v1 ->
    beval r (blt e1 e2) false.

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

Variable update :
  list(string*Z)->string->Z->list(string*Z)->Prop.

Inductive exec :
  list(string*Z)->instr->list(string*Z)->Prop :=
  sn1 : forall r, exec r skip r
| sn2 : forall r e v r',
        aeval r e v -> update r x v r' ->
        exec r (assign x e) r'
| sn3 : forall r i1 r1 i2 r',
        exec r i1 r1 -> exec r1 i2 r' ->
        exec r (sequence i1 i2) r'
| sn4 : forall b r i1 r1 r',
        beval r b true -> exec r i1 r1 ->
        exec r1 (while b i1) r' ->
        exec r (while b i1) r'
| sn5 : forall b r i1,
        beval r b false ->
        exec r (while b i) r'.

(* End of summary of the semantic description. *)

Variable tau : aexpr -> aexpr.

Hypothesis tau_prop :
  forall r e v, aeval r e v -> aeval r (tau e) v.

Definition tau_b (b:bexpr) : bexpr :=
```

```
  match b with blt e1 e2 => blt (tau e1) (tau e2) end.

Fixpoint tau_i (i:instr) : instr :=
match i with
  skip => skip
| assign x e => assign x (tau e)
| sequence i1 i2 => sequence (tau i1)(tau i2)
| while b i1 => while (tau_b b)(tau_i i1)
end.

Lemma tau_b_prop :
  forall r e v, beval r e v -> beval r (tau_b b) v.
Proof.
intros r e v H; inversion H.
simpl.
eapply blt1; auto.
eapply blt2; auto.
Qed.

Lemma tau_i_prop :
  forall r i r', exec r i r' -> exec r (tau_i i) r'.
induction 1.
simpl; apply sn1.
simpl; apply sn2; auto.
simpl; apply sn3; auto.
simpl in * |- *; apply sn4; auto.
apply tau_b_prop; auto.
simpl; apply sn5.
apply tau_b_prop; auto.
Qed.
```