Sémantique des langages de programmation
Mastère PLMT
solutions

Yves Bertot

Décembre 2004

1 Définition de langage

Nous conviendrons que le résultat d'exécution d'un fragment de programme est soit un environnement usuel, soit la paire d'une étiquette de boucle et d'un environnement. L'exécution d'une affectation et d'une instruction conditionnelle sont inchangées par rapport à l'exécution donnée en cours, mais la séquence doit tenir compte des échappements de boucles:

$$\rho \vdash \texttt{skip} \rightsquigarrow \rho$$

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

$$\frac{\rho \vdash e \to \texttt{true} \quad \rho \vdash i_1 \to \rho'}{\rho \vdash \texttt{if}\ e\ \texttt{then}\ i_1\ \texttt{else}\ i_2 \to \rho'}$$

$$\frac{\rho \vdash e \to \texttt{false} \quad \rho \vdash i_2 \to \rho'}{\rho \vdash \texttt{if}\ e\ \texttt{then}\ i_1\ \texttt{else}\ i_2 \to \rho'}$$

$$\frac{\rho \vdash i_1 \to \langle l, \rho' \rangle}{\rho \vdash i_1; i_2 \to \langle l, \rho' \rangle}$$

$$\frac{\rho \vdash i_1 \to \rho' \quad \rho' \neq \langle l, r \rangle \quad \rho' \vdash i_2 \to \rho''}{\rho \vdash i_1; i_2 \to \rho''}$$

$$\frac{\rho \vdash i \to \langle l, \rho' \rangle}{\rho \vdash l\texttt{:}\ \texttt{loop}\ i\ \texttt{end} \to \rho'}$$

$$\frac{\rho \vdash i \to \langle l', \rho' \rangle \quad l' \neq l}{\rho \vdash l\texttt{:}\ \texttt{loop}\ i\ \texttt{end} \to \langle l', \rho' \rangle}$$

$$\frac{\rho \vdash i \to \rho' \quad \rho' \neq \langle l_0, \rho_0 \rangle \quad \rho' \vdash l\texttt{:}\ \texttt{loop}\ i\ \texttt{end} \to \rho''}{\rho \vdash l\texttt{:}\ \texttt{loop}\ i\ \texttt{end} \to \rho''}$$

Le programme suivant contient la traduction de ces règles en Prolog et la traduction de l'instruction donnée en exemple. Il suffit d'ajouter la description des prédicats `eval` et `update` pour avoir une spécification exécutable.

```
not_exit([]).   not_exit([_|_]).

exec(R,assign(X,E),R1) :-
  eval(R,E,V),update(R, X, V,R1).

exec(R, seq(I1,I2), state_exit(L,R1)) :-
 exec(R, I1, state_exit(L,R1)).

exec(R, seq(I1,I2), R2) :-
 exec(R, I1, R1), not_exit(R1),  exec(R1, I2, R2).

exec(R, loop(L,I),R1) :- exec(R,I,state_exit(L,R1)).

exec(R, loop(L,I),state_exit(L1,R1)) :-
  exec(R,I,state_exit(L1,R1)), L \= L1,write(exit_loop).

exec(R, loop(L,I), R2) :-
  exec(R, I, R1), not_exit(R1), exec(R1, loop(L,I),R2).

exec(R, exit(L), state_exit(L,R)).

exec(R,if(E,I1,I2),R1) :- eval(R,E,true), exec(R,I1,R1).

exec(R,if(E,I1,I2),R1) :- eval(R,E,false), exec(R,I2,R1).

exec(R, skip, R).

example(loop(main, seq(
          assign(x,plus(var(x),int(1))), seq(
          assign(z,int(0)),
          loop(inner, seq(
            assign(z,plus(var(z),int(1))), seq(
            assign(y,plus(var(y),var(x))),
            if(blt(int(10),var(y)),exit(main),
               if(blt(int(3),var(z)),exit(inner),
                  skip))))))))).

example_rho([{x,int(0)},{y,int(0)},{z,int(0)}]).
```

2 Démontrer des propriétés

Nous allons d'abord démontrer un énoncé plus général:

$$\forall e, p, p', p' \vdash e \rightsquigarrow p \Rightarrow \forall n, \vdash e \rightarrow n \Rightarrow \forall s, s', n \cdot s \vdash p' \mapsto^* s' \Rightarrow s \vdash p \mapsto^* s'$$

Nous effectuons cette preuve par récurrence sur la dérivation qui prouve $p' \vdash e \rightsquigarrow p$.

Cas de base. Si $e = n'$ alors $p = \texttt{imm}\ n' \cdot p'$, en observant la dérivation pour $e \rightarrow n$, on trouve que $n = n'$. Nous pouvons aussi supposer $n \cdot s \vdash p' \mapsto^* s'$ et nous devons

prouver $s \vdash p \mapsto^* s'$. En utilisant la première règle de $\mapsto$, nous obtenons une preuve de $s \vdash \texttt{imm}\ n \mapsto n \cdot s$ et avec l'hypothèse ci-dessus nous avons donc les deux prémisses de la deuxième règle pour $\mapsto^*$.

Cas de récurrence Si $e = e1 + e2$, alors il existe un programme p_1 et deux dérivations prouvant les énoncés $\texttt{add} \cdot p' \vdash e_1 \rightsquigarrow p_1$ et $\texttt{p}_1 \vdash e_2 \rightsquigarrow p$. Nous disposons également d'hypothèses de récurrences correspondant à ces deux dérivations:

$$\forall n_1, \vdash e_1 \to n_1 \Rightarrow \forall s, s', n_1 \cdot s \vdash \texttt{add} \cdot p' \mapsto^* s' \Rightarrow s \vdash p_1 \mapsto^* s'$$

et

$$\forall n_2, \vdash e_2 \to n_2 \Rightarrow \forall s, s', n_2 \cdot s \vdash p_1 \mapsto^* s' \Rightarrow s \vdash p \mapsto^* s'.$$

Nous pouvons aussi supposer les faits suivants:

$$\vdash e_1 + e2 \to n$$

$$n \cdot s_0 \vdash p' \mapsto^* s_1$$

Nous devons démontrer $s_0 \vdash p \mapsto^* s_1$. En analysant la dérivation pour $e_1 + e_2 \to n$, nous trouvons qu'il existe nécessairement deux nombres n_1 et n_2 tels que $e_1 \to n_1$, $e_2 \to n_2$, et $n_1 + n_2 = n$. En utilisant la deuxième hypothèse de récurrence pour $s = s_0$, $s' = s_1$, nous trouvons

$$n_2 \cdot s_0 \vdash p_1 \mapsto^* s_1 \Rightarrow s_0 \vdash p \mapsto^* s_1$$

En utilisant la première hypothèse de récurrence pour $s = n_2 \cdot s_0$, $s' = s_1$, nous trouvons

$$n_1 \cdot n_2 \cdot s_0 \vdash \texttt{add} \cdot p' \mapsto^* s' \Rightarrow n_2 \cdot s_0 \vdash p_1 \mapsto^* s_1$$

En combinant les deux résultats, nous avons

$$n_1 \cdot n_2 \cdot s_0 \vdash \texttt{add} \cdot p \mapsto^* s_1 \Rightarrow s_0 \vdash p \mapsto^* s_1$$

En utilisant la seconde règle de $\mapsto$ et la seconde règle de $\mapsto^*$, nous arrivons à construire la dérivation suivante, qui utilise aussi le fait $n \cdot s_0 \vdash p \mapsto^* s_1$ que nous avons supposé:

$$\frac{\overline{n1 \cdot n2 \cdot s_0 \vdash \texttt{add} \mapsto n \cdot s_0} \quad n \cdot s_0 \vdash p \mapsto^* s_1}{n_1 \cdot n_2 \cdot s_0 \vdash \texttt{add} \cdot p \vdash s_1}$$

Nous pouvons déduire $s_0 \vdash p \mapsto^* s_1$, que nous cherchions.

Nous avons donc fini la preuve de l'énoncé général. Pour démontrer l'énoncé requis, il nous suffit de remarquer que $n \cdot s \vdash \emptyset \mapsto^* n \cdot s$ est donné par la première règle de $\mapsto^*$ et que l'énoncé requis est l'instanciation de l'énoncé général pour $p' = \emptyset$ et $s' = n \cdot s$.

Le script suivant contient les définitions et démonstrations en Coq.

```
Require Export List. Require Export ZArith.
Open Scope Z_scope.

Inductive expr : Set :=
```

```
   e_int : Z -> expr | e_plus : expr -> expr -> expr.

Inductive e_eval : expr -> Z -> Prop :=
  ev_int : forall n, e_eval (e_int n) n
| ev_plus :
   forall e1 e2 n1 n2, e_eval e1 n1 -> e_eval e2 n2 ->
   e_eval (e_plus e1 e2) (n1+n2).

Inductive instr : Set := imm : Z -> instr | add : instr.

Inductive i_exec : list Z -> instr -> list Z -> Prop :=
  ie_imm : forall n s, i_exec s (imm n) (n::s)
| ie_add :
     forall n1 n2 s, i_exec (n1::n2::s) add ((n1+n2)::s).

Inductive i_star : list Z -> list instr -> list Z -> Prop :=
  is_empty : forall s, i_star s nil s
| is_step : forall s s1 s2 i p, i_exec s i s1 ->
   i_star s1 p s2 -> i_star s (i::p) s2.

Inductive comp : list instr -> expr -> list instr -> Prop :=
  c_plus : forall e1 e2 p p' p'',
    comp (add::p) e1 p' -> comp p' e2 p'' ->
    comp p (e_plus e1 e2) p''
| c_int : forall p n, comp p (e_int n) (imm n::p).

Theorem comp_correct_aux :
 forall e p p', comp p' e p -> forall n, e_eval e n ->
 forall s s', i_star (n::s) p' s' -> i_star s p s'.
intros e p p' H; elim H; clear e p p' H.
(* We start with the induction case. *)
intros e1 e2 p p1 p2 Hc1 Hr1 Hc2 Hr2 n Hev s0 s1 Hs.
inversion Hev; subst n.
apply Hr2 with (n:=n2);try assumption.
apply Hr1 with (n:=n1);try assumption.
apply is_step with ((n1+n2)::s0).
apply ie_add.
exact Hs.
(* We now have the base case. *)
intros p n' n Hev s s' Hs.
inversion Hev; subst n.
apply is_step with (n'::s).
apply ie_imm.
exact Hs.
Qed.
```

3 Vérification de preuves sur machine

```
intros var val e_1 e_2 H; elim H.
 (* premier but
forall var0 val0 r, a_eval r (avar var0) val0 ->
forall n, a_eval r (avar var0) n ->
     a_eval r (aint val0) n. *)
intros var0 val0 r Hev1 n Hev2.
assert (Heq : val0 = n).
apply eval_det with (r:= r) (e:=(avar var0));auto.
rewrite <- Heq.
apply ae_int.
 (* deuxième but.
forall var0 val0 x r,
  var0 <> x -> a_eval r (avar var0) val0 ->
  forall n, a_eval r (avar x) n ->
        a_eval r (avar x) n. *)
intros  var0 val0 x r Hneq Hev1 n Hev2; auto.
 (* troisième but
forall var0 val0 m r,
  a_eval r (avar var0) val0 ->
  forall n, a_eval r (aint m) n ->
        a_eval r (aint m) n. *)
intros var0 val0 m r Hev1 n Hev2; auto.
 (* Quatrième but
 forall var0 val0 e1 e2 e3 e4,
   subst var0 val0 e1 e3 ->
   (forall r, a_eval r (avar var0) val0 ->
      forall n, a_eval r e1 n -> a_eval r e3 n) ->
   subst var0 val0 e2 e4 ->
   (forall r, a_eval r (avar var0) val0 ->
      forall n, a_eval r e2 n -> a_eval r e4 n) ->
   forall r, a_eval r (avar var0) val0 ->
   forall n, a_eval r (aplus e1 e2) n ->
   a_eval r (aplus e3 e4) n *)
intros var0 val0 e1 e2 e3 e4 Hs1 Hr1 Hs2 Hr2 r
   Hev1 n Hev2.
inversion Hev2.
 (* nouveau but : a_eval (aplus e3 e4) (v1+v2) *)
apply ae_plus.
apply Hr1; auto.
apply Hr2; auto.
Qed.
```