Operational semantics of programs

Giuseppe De Giacomo

Programs

We will consider a very simple programming language:

a	atomic action
$skip$	empty action
$\delta_1 ; \delta_2$	sequence
if ϕ **then** δ_1**else** δ_2	if-then-else
while ϕ **do** δ	while-loop

As atomic action we will typically consider assignments:

$$x := v$$

As test any boolean condition on the current state of the memory.

Notice that our consideration extend to full-fledged programming language (as Java).

Program semantics

Programs are syntactic objects.

How do we assign a formal semantics to them?

Any idea of what the semantics should talk about?

Evaluation semantics

Idea: describe the overall result of the evaluation of the program.

Given a program δ *and a memory state* s **compute the memory state** s' **obtained by executing** δ **in** s.

More formally: Define the **relation**:

$$(\delta, s) \longrightarrow s'$$

where δ is a program, s is the memory state in which the program is evaluated, and s' is the memory state obtained by the evaluation.

Such a relation can be defined inductively in a standard way using the so called **evaluation (structural) rules**

Evaluation semantics: references

The general approach we follows is is the *structural operational semantics* approach[Plotkin81, Nielson&Nielson99].

This whole-computation semantics is often call: *evaluation semantics* or *natural semantics* or *computation semantic*.

Evaluation rules for our programming constructs

Act : $\dfrac{(a, s) \longrightarrow s'}{true}$ if $s \models Pre(a)$ and $s' = Post(a, s)$

special case: assignment $\dfrac{(x := v, s) \longrightarrow s'}{true}$ if $s' = s[x = v]$

$Skip$: $\dfrac{(skip, s) \longrightarrow s}{true}$

Seq : $\dfrac{(\delta_1; \delta_2,\ s) \longrightarrow s'}{(\delta_1, s) \longrightarrow s'' \ \wedge\ (\delta_2, s'') \longrightarrow s'}$

if : $\dfrac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s) \longrightarrow s'}{(\delta_1, s) \longrightarrow s'}$ if $s \models \phi$ $\dfrac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s) \longrightarrow s'}{(\delta_2, s) \longrightarrow s'}$ if $s \models \neg\phi$

$while$: $\dfrac{(\textbf{while } \phi \textbf{ do } \delta, s) \longrightarrow s}{true}$ if $s \models \neg\phi$ $\dfrac{(\textbf{while } \phi \textbf{ do } \delta, s) \longrightarrow s'}{(\delta, s) \longrightarrow s'' \ \wedge\ (\textbf{while } \phi \textbf{ do } \delta, s'') \longrightarrow s'}$ if $s \models \phi$

Structural rules

The structural rules have the following schema:

$$\frac{\text{CONSEQUENT}}{\text{ANTECEDENT}} \text{ if SIDE-CONDITION}$$

which is to be interpreted logically as:

$$\forall(\text{ANTECEDENT} \wedge \text{SIDE-CONDITION} \supset \text{CONSEQUENT})$$

where $\forall Q$ stands for the universal closure of all free variables occurring in Q, and, typically, ANTECEDENT, SIDE-CONDITION and CONSEQUENT share free variables.

The structural rules define inductively a relation, namely: **the smallest relation satisfying the rules**.

Examples

Compute s_f in the following cases, assuming that in the memory state S_0 we have $x = 10$ and $y = 0$:

- $(x := x + 1; x := x * 2, S_0) \longrightarrow s_f$

- $(x := x + 1;$
  if $(x < 10)$ **then** $x := 0$ **else** $x := 1;$
  $x := x + 1,$
  $S_0) \longrightarrow s_f$

- $(y := 0;$ **while** $(y < 4)$ **do** $\{x := x * 2; y := y + 1\}, S_0) \longrightarrow s_f$

Transition semantics

Idea: describe the result of executing a **single step** of the program.

- *Given a program δ and a memory state s* **compute the memory state s' and the program δ' that remains to be executed obtained by executing a single step of δ in s**.

- *Assert when a program δ can be considered* **successfully terminated** *in a memory state s*.

Transition semantics (cont.)

More formally:

- Define the **relation**, named $Trans$ and denoted by "$\longrightarrow$"):

$$(\delta, s) \longrightarrow (\delta', s')$$

  where δ is a program, s is the memory state in which the program is executed, and s' is the memory state obtained by executing a single step of δ and δ' is what remains to be executed of δ after such a single step.

- Define a **predicate**. named $Final$ and denoted by " $\surd$":

$$(\delta, s)^{\surd}$$

  where δ is a program that can be considered (successfully) terminated in the memory state s.

Such a relation and predicate can be defined inductively in a standard way, using the so called **transition (structural) rules**

Transition semantics: references

The general approach we follows is is the *structural operational semantics* approach[Plotkin81, Nielson&Nielson99].

This single-step semantics is often call: *transition semantics* or *computation semantics*.

Transition rules for our programming constructs

$Act:\quad \dfrac{(a,s)\longrightarrow(\epsilon,s')}{true}$ if $s \models Pre(a)$ and $s' = Post(a,s)$

special case: assignment $\dfrac{(x:=v,s)\longrightarrow(\epsilon,s')}{true}$ if $s' = s[x=v]$

$Skip:\quad \dfrac{(skip,s)\longrightarrow(\epsilon,s)}{true}$

$Seq:\quad \dfrac{(\delta_1;\delta_2,\ s)\longrightarrow(\delta_1';\delta_2,s')}{(\delta_1,s)\longrightarrow(\delta_1',s')}$ $\quad \dfrac{(\delta_1;\delta_2,\ s)\longrightarrow(\delta_2',s')}{(\delta_2,s)\longrightarrow(\delta_2',s')}$ if $(\delta_1,s)\surd$

$if:\quad \dfrac{(\textbf{if}\ \phi\ \textbf{then}\ \delta_1 \textbf{else}\ \delta_2,s)\longrightarrow(\delta_1',s')}{(\delta_1,s)\longrightarrow(\delta_1',s')}$ if $s\models\phi$ $\quad \dfrac{(\textbf{if}\ \phi\ \textbf{then}\ \delta_1 \textbf{else}\ \delta_2,s)\longrightarrow(\delta_2',s')}{(\delta_2,s)\longrightarrow(\delta_2',s')}$ if $s\models\neg\phi$

$while:\quad \dfrac{(\textbf{while}\ \phi\ \textbf{do}\ \delta,s)\longrightarrow(\delta';\textbf{while}\ \phi\ \textbf{do}\ \delta,s)}{(\delta,s)\longrightarrow(\delta',s')}$ if $s\models\phi$

ϵ is the empty program.

Termination rules for our programming constructs

$$\epsilon: \quad \frac{(\epsilon, s)^{\surd}}{true}$$

$$Seq: \quad \frac{(\delta_1; \delta_2,\ s)^{\surd}}{(\delta_1, s)^{\surd}\ \wedge\ (\delta_2; s)^{\surd}}$$

$$if: \quad \frac{(\textbf{if}\ \phi\ \textbf{then}\ \delta_1 \textbf{else}\ \delta_2, s)^{\surd}}{(\delta_1, s)^{\surd}} \quad \text{if } s \models \phi \qquad \frac{(\textbf{if}\ \phi\ \textbf{then}\ \delta_1 \textbf{else}\ \delta_2, s)^{\surd}}{(\delta_2, s)^{\surd}} \quad \text{if } s \models \neg\phi$$

$$while: \quad \frac{(\textbf{while}\ \phi\ \textbf{do}\ \delta, s)^{\surd}}{true} \quad \text{if } s \models \neg\phi \qquad \frac{(\textbf{while}\ \phi\ \textbf{do}\ \delta, s)^{\surd}}{(\delta, s)^{\surd}} \quad \text{if } s \models \phi$$

Structural rules

The structural rules have the following schema:

$$\frac{\text{CONSEQUENT}}{\text{ANTECEDENT}} \text{ if SIDE-CONDITION}$$

which is to be interpreted logically as:

$$\forall(\text{ANTECEDENT} \wedge \text{SIDE-CONDITION} \supset \text{CONSEQUENT})$$

where $\forall Q$ stands for the universal closure of all free variables occurring in Q, and, typically, ANTECEDENT, SIDE-CONDITION and CONSEQUENT share free variables.

The structural rules define inductively a relation, namely: **the smallest relation satisfying the rules**.

Examples

Compute δ', s' in the following cases, assuming that in the memory state S_0 we have $x = 10$ and $y = 0$:

- $(x := x + 1; x := x * 2, S_0) \longrightarrow (\delta', s')$

- (**if** $(x < 10)$ **then** $\{x := 0; y := 50\}$ **else** $\{x := 1; y := 100\}$;
  $x := x + 1$,
  $S_0) \longrightarrow (\delta', s')$

- (**while** $(y < 4)$ **do** $\{x := x * 2; y := y + 1\}, S_0) \longrightarrow (\delta', s')$

Evaluation vs. transition semantics

How do we characterize a whole computation using single steps?

First we define the relation, named $Trans^*$, denoted by $\longrightarrow^*$ by the following rules:

0 $step$: $$\frac{(\delta, s)\longrightarrow^*(\delta, s)}{true}$$

$n\ step$: $$\frac{(\delta, s)\longrightarrow^*(\delta'', s'')}{(\delta, s)\longrightarrow(\delta', s') \ \wedge\ (\delta', s')\longrightarrow^*(\delta'', s'')} \quad \textit{(for some } \delta', s'\textit{)}$$

Notice that such relation is the **reflexive-transitive closure** of (single step) $\longrightarrow$.

Then it can be shown that:

$$(\delta, s_0) \longrightarrow s_f \equiv$$
$$(\delta, s_0)\longrightarrow^*(\delta_f, s_f) \ \wedge\ (\delta_f, s_f)^{\surd} \quad \text{for some } \delta_f$$

Examples

Compute s_f, using the definition based on $\longrightarrow^*$, in the following cases, assuming that in the memory state S_0 we have $x = 10$ and $y = 0$:

- $(x := x + 1; x := x * 2, S_0) \longrightarrow s_f$

- $(x := x + 1;$
  $\textbf{if } (x < 10) \textbf{ then } \{x := 0; y := 50\} \textbf{ else } \{x := 1; y := 100\};$
  $x := x + 1,$
  $S_0) \longrightarrow s_f$

- $(y := 0; \textbf{while } (y < 4) \textbf{ do } \{x := x * 2; y := y + 1\}, S_0) \longrightarrow s_f$

Concurrency

The transition semantics extends immediately to constructs for concurrency: The evaluation semantics can still be defined but only in terms of the transition semantics (as above).

We model concurrent processes by **interleaving**: *A concurrent execution of two processes is one where the primitive actions in both processes occur, interleaved in some fashion.*

It is OK for a process to remain **blocked** for a while, the other processes will continue and eventually unblock it.

Constructs for concurrency

if ϕ **then** δ_1 **else** δ_2,	synchronized conditional
while ϕ **do** δ,	synchronized loop
$(\delta_1 \parallel \delta_2)$,	concurrent execution

The constructs **if** ϕ **then** δ_1 **else** δ_2 and **while** ϕ **do** δ are the synchronized: *testing the condition* ϕ *does not involve a transition per se, the evaluation of the condition and the first action of the branch chosen are executed as an atomic unit.*

Similar to test-and-set atomic instructions used to build semaphores in concurrent programming.

Transition and termination rules for concurrency

$$transition: \quad \frac{(\delta_1 \parallel \delta_2,\ s) \longrightarrow (\delta_1' \parallel \delta_2, s')}{(\delta_1, s) \longrightarrow (\delta_1', s')} \qquad \frac{(\delta_1 \parallel \delta_2,\ s) \longrightarrow (\delta_1 \parallel \delta_2', s')}{(\delta_2, s) \longrightarrow (\delta_2', s')}$$

$$termination: \quad \frac{(\delta_1 \parallel \delta_2,\ s)^{\surd}}{(\delta_1, s)^{\surd} \ \wedge \ (\delta_2, s)^{\surd}}$$