Structural Operational Semantics

Mooly Sagiv

Reference: Semantics with Applications

Chapter 2

H. Nielson and F. Nielson

http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html

Motivation

- The natural semantics may be too abstract
- Does not capture individual steps
- Hard to express certain properties

Structural Operational Semantics

- Emphasizes the individual steps
- Usually more suitable for analysis
- For every statement S, write meaning rules <S, *i*> $\Rightarrow \gamma$
  "If the **first** step of executing the statement S on an input state *i* leads to γ"
- Two possibilities for γ
  - γ = <S', s'> The execution of S is not completed, S' is the remaining computation which need to be performed on s'
  - γ = o The execution of S has terminated with a final state o
  - γ is a stuck configuration when there are no transitions
- The meaning of a program P on an input state s is the set of final states that can be executed in arbitrary finite steps

Structural Semantics for While

axioms

$[ass_{sos}] \langle x := a, s\rangle \Rightarrow s[x \mapsto \mathbf{A}[\![a]\!]s]$

$[skip_{sos}] \langle \mathbf{skip}, s\rangle \Rightarrow s$

rules

$$[comp^1_{sos}] \frac{\langle S_1, s\rangle \Rightarrow \langle S'_1, s'\rangle}{\langle S_1; S_2, s\rangle \Rightarrow \langle S'_1; S_2, s'\rangle}$$

$$[comp^2_{sos}] \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1; S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

Structural Semantics for While if construct

$[if^{tt}_{sos}]$ $\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$ if $\mathbf{B}[\![b]\!]s=tt$

$[if^{ff}_{os}]$ $\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$ if $\mathbf{B}[\![b]\!]s=ff$

Structural Semantics for While while construct

$[\text{while}_{sos}]$ $\langle \text{while b do S}, s \rangle \Rightarrow$
$\langle \text{if b then (S; while b do S) else skip}, s \rangle$

Derivation Sequences

- A finite derivation sequence starting at <S, s>
  $\gamma_0, \gamma_1, \gamma_2 \ldots, \gamma_k$ such that
  - γ_0=<S, s>
  - $\gamma_i \Rightarrow \gamma_{i+1}$
  - γ_k is either stuck configuration or a final state
- An infinite derivation sequence starting at <S, s>
  $\gamma_0, \gamma_1, \gamma_2 \ldots$ such that
  - γ_0=<S, s>
  - $\gamma_i \Rightarrow \gamma_{i+1}$
- $\gamma_0 \Rightarrow^i \gamma_i$ in i steps
- $\gamma_0 \Rightarrow^* \gamma_i$ in finite number of steps
- For each step there is a derivation tree

Example

- Let s_0 such that
  $s_0\, x = 5$
  and
  $s_0\, y = 7$
- S = (z:=x; x := y); y := z

Factorial Program

- Input state s such that s x = 3

y := 1; while ¬(x=1) do (y := y * x; x := x - 1)

$\langle y := 1 ; W, s\rangle$

$\Rightarrow \langle W, s[y \mapsto 1]\rangle$

$\Rightarrow \langle$if ¬ ¬ (x =1) then skip else ((y := y * x ; x := x – 1); W), $s[y \mapsto 1]\rangle$

$\Rightarrow \langle$ ((y := y * x ; x := x – 1); W), $s[y \mapsto 1]\rangle$

$\Rightarrow \langle$(x := x – 1 ; W), $s[y \mapsto 3]\rangle$

$\Rightarrow \langle$ W , $s[y \mapsto 3][x \mapsto 2]\rangle$

$\Rightarrow \langle$if ¬ ¬ (x =1) then skip else ((y := y * x ; x := x – 1); W), $s[y \mapsto 3][x \mapsto 2]\rangle$

$\Rightarrow \langle$ ((y := y * x ; x := x – 1); W), $s[y \mapsto 3]\ [x \mapsto 2]\ \rangle$

$\Rightarrow \langle$(x := x – 1 ; W) , $s[y \mapsto 6]\ [x \mapsto 2]\ \rangle$

$\Rightarrow \langle$ W, $s[y \mapsto 6][x \mapsto 1]\rangle$

$\Rightarrow \langle$if ¬ ¬ (x =1) then skip else ((y := y * x ; x := x – 1); W), $s[y \mapsto 6][x \mapsto 1]\rangle$

$\Rightarrow \langle$skip, $s[y \mapsto 6][x \mapsto 1]\rangle \Rightarrow s[y \mapsto 6][x \mapsto 1]$

Prolog Implementation

```
sos(skip, Env, Env).

sos(ass(LHS, RHS), Env, Envp) :-
                    eval_exp(RHS, Env, Val),
                    upd(Env, LHS, Val, Envp).

sos(seq(S1, S2), Env, S2, Envp) :-
                  sos(S1, Env, Envp).

sos(seq(S1, S2), Env, seq(S1p, S2), Envp) :-
                sos(S1, Env, S1p, Envp).

sos(if(B, S1, _), Env, S1, Env) :-
              eval_boolean_exp(B, Env).

sos(if(B, _, S2), Env, S2, Env) :- \+ eval_boolean_exp(B, Env).

sos(while(B,S), Env, if(B, seq(S, while(B,S)), skip), Env).
```

Reflexive Transitive Closure

- Let $R \subseteq V \times V$ be a binary relation
- Then R* is the reflexive transitive closure of R
  - $R \subseteq R^*$
  - R* is reflexive, i.e., for all $v \in V$: $\langle v, v \rangle \in R^*$
  - R* is transitive, i.e., for all $u, v, w \in V$ such that $\langle u, v \rangle, \langle v, w \rangle \in R$, it must be that $\langle u, w \rangle \in R^*$
  - R* is minimal

```
rstar(U, V) :- r(U, V).
rstar(U, U).
rstar(U, W) :- r(U, V), rstar(V, W).
r(a, b).
r(b, c).
r(c, d).
```

Prolog Implementation

```
rtc_sos(S, Env, S, Env).
rtc_sos(S, Env, Sp, Envp) :- sos(S, Env, Spp, Envpp),
                             rtc_sos(Spp, Envpp, Sp, Envp).
rtc_sos(S, Env, Envp) :- rtc_sos(S, Env, Sp, Envpp),
                         sos(Sp, Envpp, Envp).
```

rts_sos(y:=1, [], E).

rts_sos(y:=5; while y != 0 do y := y-1, [], E).

rts_sos(y:=5; while y != 0 do y := y+1, [], E).

Program Termination

- Given a statement S and input s
  - S terminates on s if there exists a finite derivation sequence starting at <S, s>
  - S terminates successfully on s if there exists a finite derivation sequence starting at <S, s> leading to a final state
  - S loops on s if there exists an infinite derivation sequence starting at <S, s>

Properties of the Semantics

- S_1 and S_2 are semantically equivalent if:
  - for all s and γ which is either final or stuck $\langle S_1, s\rangle \Rightarrow^* \gamma$ if and only if $\langle S_2, s\rangle \Rightarrow^* \gamma$
  - there is an infinite derivation sequence starting at $\langle S_1, s\rangle$ if and only if there is an infinite derivation sequence starting at $\langle S_2, s\rangle$
- Deterministic
  - If $\langle S, s\rangle \Rightarrow^* s_1$ and $\langle S, s\rangle \Rightarrow^* s_2$ then $s_1 = s_2$
- The execution of $S_1; S_2$ on an input can be split into two parts:
  - execute S_1 on s yielding a state s'
  - execute S_2 on s'

Sequential Composition

- If $\langle S_1; S_2, s\rangle \Rightarrow^k s''$ then there exists a state s' and numbers k_1 and k_2 such that
  - $\langle S_1, s\rangle \Rightarrow^{k1} s'$
  - $\langle S_2, s'\rangle \Rightarrow^{k2} s''$
  - and $k = k_1 + k_2$
- The proof uses induction on the length of derivation sequences
  - Prove that the property holds for all derivation sequences of length 0
  - Prove that the property holds for all other derivation sequences:
    - Show that the property holds for sequences of length k+1 using the fact it holds on all sequences of length k (induction hypothesis)

The Semantic Function S_{sos}

- The meaning of a statement S is defined as a partial function from **State** to **State**
- S_{sos}: **Stm** $\rightarrow$ **(State** $\hookrightarrow$ **State)**
- $S_{sos}[\![S]\!]s = s'$ if $\langle S, s\rangle \Rightarrow^* s'$ and otherwise $S_{sos}[\![S]\!]s$ is undefined

An Equivalence Result

- For every statement S of the While language
  - $S_{nat}[\![S]\!] = S_{sos}[\![S]\!]$

Extensions to While

- Abort statement (like C exit w/o return value)
- Non determinism
- Parallelism
- Local Variables
- Procedures
  - Static Scope
  - Dynamic scope

The **While** Programming Language with Abort

- Abstract syntax
  S::= x := a | **skip** | S_1 ; S_2 | **if** b **then** S_1 **else** S_2 |
  while b do S| **abort**
- Abort terminates the execution
- No new rules are needed in natural and structural operational semantics
- Statements
  - if x = 0 then abort else y := y / x
  - skip
  - abort
  - while true do skip

Conclusion

- The natural semantics cannot distinguish between looping and abnormal termination (unless the states are modified)
- In the structural operational semantics looping is reflected by infinite derivations and abnormal termination is reflected by stuck configuration

The **While** Programming Language with Non-Determinism

- Abstract syntax
  S::= x := a | **skip** | S_1 ; S_2 | **if** b **then** S_1 **else** S_2 |
  while b do S| S_1 or S_2
- Either S_1 or S_2 is executed
- Example
  - x := 1 or (x :=2 ; x := x+2)

The While Programming Language with Non-Determinism Natural Semantics

$$[or^1_{ns}]\ \frac{\langle S_1, s\rangle \rightarrow s'}{\langle S_1 \text{ or } S_2, s\rangle \rightarrow s'}$$

$$[or^2_{ns}]\ \frac{\langle S_2, s\rangle \rightarrow s'}{\langle S_1 \text{ or } S_2, s\rangle \rightarrow s'}$$

The While Programming Language with Non-Determinism Structural Semantics

The While Programming Language with Non-Determinism Examples

- x := 1 or (x :=2 ; x := x+2)
- (while true do skip) or (x :=2 ; x := x+2)

Conclusion

- In the natural semantics non-determinism will suppress looping if possible (mnemonic)
- In the structural operational semantics non-determinism does suppress not termination configuration

The **While** Programming Language with Parallel Constructs

- Abstract syntax
  S::= x := a | **skip** | S_1 ; S_2 | **if** b **then** S_1 **else** S_2 |
  while b do S| S_1 par S_2
- All the interleaving of S_1 or S_2 are executed
- Example
  - x := 1 par (x :=2 ; x := x+2)

The **While** Programming Language with Parallel Constructs Structural Semantics

$$[par^1_{sos}]\ \frac{\langle S_1, s\rangle \Rightarrow \langle S'_1, s'\rangle}{\langle S_1 \text{ par } S_2, s\rangle \Rightarrow \langle S'_1 \text{ par } S_2, s'\rangle}$$

$$[par^2_{sos}]\ \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 \text{ par } S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$

$$[par^3_{sos}]\ \frac{\langle S_2, s\rangle \Rightarrow \langle S'_2, s'\rangle}{\langle S_1 \text{ par } S_2, s\rangle \Rightarrow \langle S_1 \text{ par } S'_2, s'\rangle}$$

$$[par^4_{sos}]\ \frac{\langle S_2, s\rangle \Rightarrow s'}{\langle S_1 \text{ par } S_2, s\rangle \Rightarrow \langle S_1, s'\rangle}$$

The **While** Programming Language with Parallel Constructs Natural Semantics

Conclusion

- In the natural semantics immediate constituent is an atomic entity so we cannot express interleaving of computations
- In the structural operational semantics we concentrate on small steps so interleaving of computations can be easily expressed

The **While** Programming Language with local variables and procedures

- Abstract syntax

  S::= x := a | **skip** | S_1 ; S_2 | **if** b **then** S_1 **else** S_2 |
  while b do S|
  begin D_v D_p S **end | call p**

  D_v ::= **var** x := a ; D_v | ε

  D_p ::= **proc** p **is** S ; D_p | ε

Conclusions Local Variables

- The natural semantics can “remember” local states
- Need to introduce stack or heap into state of the structural semantics

Some Applications

- An Operational Semantics for JavaScript: Ankur Taly: Stanford & Google 2008
  - Used for security
- Operational semantics for Java
  - Used for proving type safety
  - Used to understand concurrency

Summary

- SOS is powerful enough to describe imperative programs
- Natural operational semantics is an abstraction
- Different semantics may be used to justify different behaviors