

Formal Syntax and Semantics of Programming Languages

Mooly Sagiv

Reference: Semantics with Applications

Chapter 2

H. Nielson and F. Nielson

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

Natural Semantics for While

$$[\text{ass}_{\text{ns}}] \langle x := a, s \rangle \rightarrow s[x \mapsto \mathbf{A}[[a]]s]$$

axioms

$$[\text{skip}_{\text{ns}}] \langle \mathbf{skip}, s \rangle \rightarrow s$$

$$[\text{comp}_{\text{ns}}] \frac{\langle S_1, s \rangle \rightarrow s', \langle S_2, s' \rangle \rightarrow s''}{\langle S_1; S_2, s \rangle \rightarrow s''}$$

rules

$$\langle S_1; S_2, s \rangle \rightarrow s''$$

$$[\text{if}^{\text{tt}}_{\text{ns}}] \langle S_1, s \rangle \rightarrow s'$$

$$\frac{\langle S_1, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'}$$

if $\mathbf{B}[[b]]s = \text{tt}$

$$[\text{if}^{\text{ff}}_{\text{ns}}] \langle S_2, s \rangle \rightarrow s'$$

$$\frac{\langle S_2, s \rangle \rightarrow s'}{\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'}$$

if $\mathbf{B}[[b]]s = \text{ff}$

Natural Semantics for While (More rules)

$$\frac{[\text{while}_{\text{ns}}^{\text{ff}}] \quad \langle \text{while } b \text{ do } S, s \rangle \rightarrow s}{\text{if } \mathbf{B}[[b]]s = \text{ff}}$$

$$\frac{[\text{while}_{\text{ns}}^{\text{tt}}] \quad \langle S, s \rangle \rightarrow s', \langle \text{while } b \text{ do } S, s' \rangle \rightarrow s''}{\langle \text{while } b \text{ do } S, s \rangle \rightarrow s''} \quad \text{if } \mathbf{B}[[b]]s = \text{tt}$$

An Example Derivation Tree

$\langle (x := x+1; y := x+1); z := y, s_0 \rangle \rightarrow s_0[x \mapsto 1][y \mapsto 2][z \mapsto 2]$

comp_{ns}

$\langle x := x+1; y := x+1, s_0 \rangle \rightarrow s_0[x \mapsto 1][y \mapsto 2]$

$\langle z := y, s_0[x \mapsto 1][y \mapsto 2] \rangle \rightarrow s_0[x \mapsto 1][y \mapsto 2][z \mapsto 2]$

comp_{ns}

ass_{ns}

$\langle x := x+1; s_0 \rangle \rightarrow s_0[x \mapsto 1]$

$\langle y := x+1, s_0[x \mapsto 1] \rangle \rightarrow s_0[x \mapsto 1][y \mapsto 2]$

ass_{ns}

ass_{ns}

Top Down Evaluation of Derivation Trees

- Given a program S and an input state s
- Find an output state s' such that
$$\langle S, s \rangle \rightarrow s'$$
- Start with the root and repeatedly apply rules until the axioms are reached
- Inspect different alternatives in order
- In While s' and the derivation tree is unique

Semantic Equivalence

- S_1 and S_2 are **semantically equivalent** if for all s and s'
 $\langle S_1, s \rangle \rightarrow s'$ if and only if $\langle S_2, s \rangle \rightarrow s'$
- Simple example
“while b do S ”
is semantically equivalent to:
“if b then (S ; while b do S) else skip”

Deterministic Semantics for While

- If $\langle S, s \rangle \rightarrow s_1$ and $\langle S, s \rangle \rightarrow s_2$ then $s_1 = s_2$
- The proof uses induction on the shape of derivation trees
 - Prove that the property holds for all simple derivation trees by showing it holds for axioms
 - Prove that the property holds for all composite trees:
 - For each rule assume that the property holds for its premises (induction hypothesis) and prove it holds for the conclusion of the rule

The Semantic Function S_{ns}

- The meaning of a statement S is defined as a partial function from **State** to **State**
- $S_{ns}: \mathbf{Stm} \rightarrow (\mathbf{State} \hookrightarrow \mathbf{State})$
- $S_{ns} \llbracket S \rrbracket s = s'$ if $\langle S, s \rangle \rightarrow s'$ and otherwise $S_{ns} \llbracket S \rrbracket s$ is undefined
- Examples
 - $S_{ns} \llbracket \text{skip} \rrbracket s = s$
 - $S_{ns} \llbracket x := 1 \rrbracket s = s [x \mapsto 1]$
 - $S_{ns} \llbracket \text{while true do skip} \rrbracket s = \text{undefined}$

Structural Operational Semantics

- Emphasizes the individual execution steps
- $\langle S, i \rangle \Rightarrow \gamma$
 - If the “**first**” step of executing the statement S on an input state i leads to γ
- Two possibilities for γ
 - $\gamma = \langle S', s' \rangle$
 - The execution of S is not completed, S' is the remaining computation which need to be performed on s'
 - $\gamma = o$
 - 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

$$[\text{ass}_{\text{sos}}] \langle x := a, s \rangle \Rightarrow s[x \mapsto \mathbf{A}[[a]]s]$$

axioms

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

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

rules

$$\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle$$

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

$$\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle$$

Structural Semantics for While if construct

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

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

Structural Semantics for While while construct

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

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 $s.x = 3$

$y := 1; \text{ while } \neg(x=1) \text{ do } (y := y * x; x := x - 1)$

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

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

$\Rightarrow \langle \text{if } \neg \neg(x=1) \text{ then skip else } ((y := y * x; x := x - 1); W), s[y \mapsto 1] \rangle$

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

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

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

$\Rightarrow \langle \text{if } \neg \neg(x=1) \text{ then skip else } ((y := y * x; x := x - 1); W), s[y \mapsto 3][x \mapsto 2] \rangle$

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

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

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

$\Rightarrow \langle \text{if } \neg \neg(x=1) \text{ then skip else } ((y := y * x; x := x - 1); W), s[y \mapsto 6][x \mapsto 1] \rangle$

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

Finite Derivation Sequences

- finite derivation sequence starting at $\langle S, i \rangle$

$\gamma_0, \gamma_1, \gamma_2 \dots, \gamma_k$ such that

- $\gamma_0 = \langle S, i \rangle$

- $\gamma_i \Rightarrow \gamma_{i+1}$

- γ_k is either stuck configuration or a final state

- For each step there is a derivation tree
- $\gamma_0 \Rightarrow^k \gamma_k$ in k steps
- $\gamma_0 \Rightarrow^* \gamma$ in finite number of steps

Infinite Derivation Sequences

- An infinite derivation sequence starting at $\langle S, i \rangle$

$\gamma_0, \gamma_1, \gamma_2 \dots$ such that

– $\gamma_0 = \langle S, i \rangle$

– $\gamma_i \Rightarrow \gamma_{i+1}$

- Example

– $S = \text{while true do skip}$

– $s_0 \ x = 0$

Program Termination

- Given a statement S and input s
 - S **terminates** on s if there exists a finite derivation sequence starting at $\langle S, s \rangle$
 - S **terminates successfully** on s if there exists a finite derivation sequence starting at $\langle S, s \rangle$ leading to a final state
 - S **loops** on s if there exists an infinite derivation sequence starting at $\langle S, s \rangle$

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
 - $\langle S_1, s \rangle \Rightarrow^{k_1} s'$
 - $\langle S_2, s' \rangle \Rightarrow^{k_2} s''$
 - 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**
- $S_{\text{sos}}: \mathbf{Stm} \rightarrow (\mathbf{State} \hookrightarrow \mathbf{State})$
- $S_{\text{sos}}[[S]]s = s'$ if $\langle S, s \rangle \Rightarrow^* s'$ and otherwise $S_{\text{sos}}[[S]]s$ is undefined

An Equivalence Result

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

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
$$S ::= x := a \mid \mathbf{skip} \mid S_1 ; S_2 \mid \mathbf{if} \ b \ \mathbf{then} \ S_1 \ \mathbf{else} \ S_2 \mid$$
$$\mathbf{while} \ b \ \mathbf{do} \ S \mid \mathbf{abort}$$
- 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

$S ::= x := a \mid \mathbf{skip} \mid S_1 ; S_2 \mid \mathbf{if} \ b \ \mathbf{then} \ S_1 \ \mathbf{else} \ S_2 \mid$
 $\mathbf{while} \ b \ \mathbf{do} \ S \mid S_1 \ \mathbf{or} \ S_2$

- Either S_1 or S_2 is executed
- Example
 - $x := 1 \ \mathbf{or} \ (x := 2 ; x := x+2)$

The While Programming Language with Non-Determinism Natural Semantics

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

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

The While Programming Language with Non-Determinism Structural Semantics

The While Programming Language with Non-Determinism

Examples

- $x := 1$ or $(x := 2 ; x := x+2)$
- $(\text{while true do skip})$ 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

$S ::= x := a \mid \mathbf{skip} \mid S_1 ; S_2 \mid \mathbf{if} \ b \ \mathbf{then} \ S_1 \ \mathbf{else} \ S_2 \mid$
 $\mathbf{while} \ b \ \mathbf{do} \ S \mid S_1 \ \mathbf{par} \ S_2$

- All the interleaving of S_1 or S_2 are executed
- Example
 - $x := 1 \ \mathbf{par} \ (x := 2 ; x := x+2)$

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

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

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

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

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

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

- Abstract syntax

$S ::= x := a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid$
 $\quad \text{while } b \text{ do } S \mid \text{begin } D_v S \text{ end}$

$D_v ::= \text{var } x := a ; D_v \mid \varepsilon$

Simple Example

```
begin
  var y := 1;
  (var x := 2 ;
    begin
      var x := 3 ;
      y := x + y;
    end
    x := y + x ;
  )
end
```

Another Example

```
while (y > 0) (  
  begin  
    var x := y ;  
    y := x + y;  
    y := y - 1 ;  
  end  
  x := y + x ;  
)  
end
```

Natural Semantics

$$DV : D_v \rightarrow 2^{\text{Var}}$$

$$DV(\varepsilon) = \emptyset$$

$$DV(\mathbf{var} \ x := a ; D_v) = \{x\} \cup DV(D_v)$$

$$s[X \mapsto s'] = \begin{cases} s \ x & \text{if } x \in X \\ s' \ x & \text{if } x \notin X \end{cases}$$

$$[\text{none}_{\text{ns}}] \langle \varepsilon, s \rangle \rightarrow s$$

$$[\text{var}_{\text{ns}}] \frac{\langle D_v, s[x \mapsto \mathbf{A}[[a]]s] \rangle \rightarrow s'}{\langle \mathbf{var} \ x := a ; D_v, s \rangle \rightarrow s'}$$

$$[\text{block}_{\text{ns}}] \frac{\langle D_v, s \rangle \rightarrow s', \langle S, s' \rangle \rightarrow s''}{\langle \mathbf{begin} \ D_v \ S \ \mathbf{end}, s \rangle \rightarrow s'' [DV(D_v) \mapsto s]}$$

Structural Semantics

$$\frac{?}{[\text{block}_{\text{sos}}]} \langle \text{begin } D_v \text{ S end, } s \rangle \Rightarrow s'$$

Conclusions Local Variables

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

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

- Abstract syntax

$S ::= x := a \mid \text{skip} \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \mid$
 $\text{while } b \text{ do } S \mid$

$\text{begin } D_v D_p S \text{ end} \mid \text{call } p$

$D_v ::= \text{var } x := a ; D_v \mid \varepsilon$

$D_p ::= \text{proc } p \text{ is } S ; D_p \mid \varepsilon$

Summary

- SOS is powerful enough to describe imperative programs
 - Can define the set of traces
 - Can represent program counter implicitly
 - Handle gotos
- Natural operational semantics is an abstraction
- Different semantics may be used to justify different behaviors
- Thinking in concrete semantics is essential for language designer/compiler writer/...