Semantics and Verification of Software

Summer Semester 2015

Lecture 13: Extension by Blocks and Procedures I (Operational Semantics)

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

http://moves.rwth-aachen.de/teaching/ss-15/sv-sw/

Outline of Lecture 13

Extension by Blocks and Procedures

Extending the Syntax

New Semantic Domains

Execution Relation

Blocks and Procedures

- Extension of WHILE by blocks with (local) variables and (recursive) procedures

Blocks and Procedures

- Extension of WHILE by blocks with (local) variables and (recursive) procedures
- Simple memory model ($\Sigma := \{\sigma \mid \sigma : Var \to \mathbb{Z}\}$) not sufficient anymore as variables can occur in several instances

$\Rightarrow$ Involves new semantic concepts:
  - variable und procedure environments
  - locations (memory addresses) and stores (memory states)

Blocks and Procedures

- Extension of WHILE by blocks with (local) variables and (recursive) procedures
- Simple memory model ($\Sigma := \{\sigma \mid \sigma : Var \to \mathbb{Z}\}$) not sufficient anymore as variables can occur in several instances

$\Rightarrow$ Involves new semantic concepts:

  - variable und procedure environments
  - locations (memory addresses) and stores (memory states)

- Important: scope of variable and procedure identifiers

  static scoping: scope of identifier = declaration environment
  (also: “lexical” scoping; here)

  dynamic scoping: scope of identifier = calling environment
  (old Algol/Lisp dialects)

Static and Dynamic Scoping

Example 13.1

```
begin
  var x; var y;
  proc P is y := x end;
  x := 1;
  begin
    var x;
    x := 2;
    call P
  end
end
```

Static and Dynamic Scoping

Example 13.1

```
begin
  var x; var y;
  proc P is y := x end;
  x := 1;
  begin
    var x;
    x := 2;
    call P
  end
end
```

static scoping $\Rightarrow$ y = 1

Static and Dynamic Scoping

Example 13.1

```
begin
  var x; var y;
  proc P is y := x end;
  x := 1;
  begin
    var x;
    x := 2;
    call P
  end
end
```

static scoping $\Rightarrow$ y = 1

dynamic scoping $\Rightarrow$ y = 2

Outline of Lecture 13

Extending the Syntax

Syntactic categories:

Category	Domain	Meta variable
Procedure identifiers	$\mathit{PVar} = \{\mathtt{P}, \mathtt{Q}, \ldots\}$	P
Procedure declarations	PDec	p
Variable declarations	VDec	v
Commands (statements)	Cmd	c

Extending the Syntax

Syntactic categories:

Category	Domain	Meta variable
Procedure identifiers	$PVar = \{\mathtt{P}, \mathtt{Q}, \ldots\}$	P
Procedure declarations	$PDec$	p
Variable declarations	$VDec$	v
Commands (statements)	Cmd	c

Context-free grammar:

$$\begin{aligned}
p &::= \mathtt{proc}\ P\ \mathtt{is}\ c\ \mathtt{end;}\,p \mid \varepsilon \in PDec \\
v &::= \mathtt{var}\ x\,\mathtt{;}\,v \mid \varepsilon \in VDec \\
c &::= \mathtt{skip} \mid x \mathrel{\mathtt{:=}} a \mid c_1\,\mathtt{;}\,c_2 \mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \mathtt{end} \mid \mathtt{while}\ b\ \mathtt{do}\ c\ \mathtt{end} \mid \\
&\quad\ \mathtt{call}\ P \mid \mathtt{begin}\ v\ p\ c\ \mathtt{end} \in Cmd
\end{aligned}$$

Extending the Syntax

Syntactic categories:

Category	Domain	Meta variable
Procedure identifiers	$PVar = \{\mathtt{P}, \mathtt{Q}, \ldots\}$	P
Procedure declarations	$PDec$	p
Variable declarations	$VDec$	v
Commands (statements)	Cmd	c

Context-free grammar:

$$
\begin{aligned}
p &::= \mathtt{proc}\ P\ \mathtt{is}\ c\ \mathtt{end;}\, p \mid \varepsilon \in PDec \\
v &::= \mathtt{var}\ x\,\mathtt{;}\, v \mid \varepsilon \in VDec \\
c &::= \mathtt{skip} \mid x\ \mathtt{:=}\ a \mid c_1\,\mathtt{;}\,c_2 \mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \mathtt{end} \mid \mathtt{while}\ b\ \mathtt{do}\ c\ \mathtt{end} \mid \\
&\ \mathtt{call}\ P \mid \mathtt{begin}\ v\ p\ c\ \mathtt{end} \in Cmd
\end{aligned}
$$

- All used variable/procedure identifiers have to be declared
- Identifiers declared within a block must be distinct

Outline of Lecture 13

Extension by Blocks and Procedures

Extending the Syntax

New Semantic Domains

Execution Relation

New Semantic Domains

Locations and Stores

- So far: states $\Sigma = \{\sigma \mid \sigma : \mathit{Var} \to \mathbb{Z}\}$

New Semantic Domains

Locations and Stores

- So far: states $\Sigma = \{\sigma \mid \sigma : \mathit{Var} \to \mathbb{Z}\}$
- Now: explicit control over all (nested) instances of a variable:
  - variable environments $\mathit{VEnv} := \{\rho \mid \rho : \mathit{Var} \dashrightarrow \mathit{Loc}\}$
    (partial function to maintain declaredness information)
  - locations $\mathit{Loc} := \mathbb{N}$
  - stores $\mathit{Sto} := \{\sigma \mid \sigma : \mathit{Loc} \dashrightarrow \mathbb{Z}\}$
    (partial function to maintain allocation information)

Locations and Stores

- So far: states $\Sigma = \{\sigma \mid \sigma : \mathit{Var} \to \mathbb{Z}\}$
- Now: explicit control over all (nested) instances of a variable:
  - variable environments $\mathit{VEnv} := \{\rho \mid \rho : \mathit{Var} \dashrightarrow \mathit{Loc}\}$
    (partial function to maintain declaredness information)
  - locations $\mathit{Loc} := \mathbb{N}$
  - stores $\mathit{Sto} := \{\sigma \mid \sigma : \mathit{Loc} \dashrightarrow \mathbb{Z}\}$
    (partial function to maintain allocation information)

$\Rightarrow$ Two-level access to a variable $x \in \mathit{Var}$:

1. determine current memory location of x:
$$l := \rho(x)$$
2. reading/writing access to σ at position l

- Thus: previous state information represented as $\sigma \circ \rho$

Procedure Environments and Declarations

- Effect of procedure call determined by its body and variable and procedure environment of its declaration:

$$\mathit{PEnv} := \{\pi \mid \pi : \mathit{PVar} \dashrightarrow \mathit{Cmd} \times \mathit{VEnv} \times \mathit{PEnv}\}$$

denotes the set of procedure environments

Procedure Environments and Declarations

- Effect of procedure call determined by its body and variable and procedure environment of its declaration:

$$PEnv := \{\pi \mid \pi : PVar \dashrightarrow Cmd \times VEnv \times PEnv\}$$

denotes the set of procedure environments

- Effect of declaration: update of environment (and store)

$$\begin{aligned}
\mathrm{upd}_v[\![.]\!] &: VDec \times VEnv \times Sto \to VEnv \times Sto\\
\mathrm{upd}_v[\![\mathtt{var}\ x\,;v]\!](\rho,\sigma) &:= \mathrm{upd}_v[\![v]\!](\rho[x \mapsto l_x], \sigma[l_x \mapsto 0])\\
\mathrm{upd}_v[\![\varepsilon]\!](\rho,\sigma) &:= (\rho,\sigma)
\end{aligned}$$

$$\begin{aligned}
\mathrm{upd}_p[\![.]\!] &: PDec \times VEnv \times PEnv \to PEnv\\
\mathrm{upd}_p[\![\mathtt{proc}\ P\ \mathtt{is}\ c\ \mathtt{end}\,;p]\!](\rho,\pi) &:= \mathrm{upd}_p[\![p]\!](\rho, \pi[P \mapsto (c,\rho,\pi)])\\
\mathrm{upd}_p[\![\varepsilon]\!](\rho,\pi) &:= \pi
\end{aligned}$$

where $l_x := \min\{l \in Loc \mid \sigma(l) = \bot\}$

Outline of Lecture 13

Execution Relation I

Definition 13.2 (Execution relation)

For $c \in Cmd$, $\sigma, \sigma' \in Sto$, $\rho \in VEnv$, and $\pi \in PEnv$, the execution relation $(\rho, \pi) \vdash \langle c, \sigma \rangle \rightarrow \sigma'$ ("in environment (ρ, π), statement c transforms store σ into σ'") is defined by the following rules:

$$\text{(skip)}\ \frac{}{(\rho, \pi) \vdash \langle \mathtt{skip}, \sigma \rangle \rightarrow \sigma}$$

$$\text{(asgn)}\ \frac{\langle a, \sigma \circ \rho \rangle \rightarrow z}{(\rho, \pi) \vdash \langle x := a, \sigma \rangle \rightarrow \sigma[\rho(x) \mapsto z]}$$

$$\text{(seq)}\ \frac{(\rho, \pi) \vdash \langle c_1, \sigma \rangle \rightarrow \sigma' \quad (\rho, \pi) \vdash \langle c_2, \sigma' \rangle \rightarrow \sigma''}{(\rho, \pi) \vdash \langle c_1 ; c_2, \sigma \rangle \rightarrow \sigma''}$$

$$\text{(if-t)}\ \frac{\langle b, \sigma \circ \rho \rangle \rightarrow \mathsf{true} \quad (\rho, \pi) \vdash \langle c_1, \sigma \rangle \rightarrow \sigma'}{(\rho, \pi) \vdash \langle \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \mathtt{end}, \sigma \rangle \rightarrow \sigma'}$$

Execution Relation II

Definition 13.2 (Execution relation; continued)

$$\text{(if-f)}\ \frac{\langle b, \sigma \circ \rho\rangle \to \mathsf{false} \qquad (\rho, \pi) \vdash \langle c_2, \sigma\rangle \to \sigma'}{(\rho, \pi) \vdash \langle \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \texttt{ end}, \sigma\rangle \to \sigma'}$$

$$\text{(wh-f)}\ \frac{\langle b, \sigma \circ \rho\rangle \to \mathsf{false}}{(\rho, \pi) \vdash \langle \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma\rangle \to \sigma}$$

$$\text{(wh-t)}\ \frac{\langle b, \sigma \circ \rho\rangle \to \mathsf{true} \quad (\rho, \pi) \vdash \langle c, \sigma\rangle \to \sigma' \quad (\rho, \pi) \vdash \langle \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma'\rangle \to \sigma''}{(\rho, \pi) \vdash \langle \texttt{while } b \texttt{ do } c \texttt{ end}, \sigma\rangle \to \sigma''}$$

$$\text{(call)}\ \frac{(\rho', \pi'[P \mapsto (c, \rho', \pi')]) \vdash \langle c, \sigma\rangle \to \sigma'}{(\rho, \pi) \vdash \langle \texttt{call } P, \sigma\rangle \to \sigma'} \qquad \text{if } \pi(P) = (c, \rho', \pi')$$

$$\text{(block)}\ \frac{\mathsf{upd}_v[\![v]\!](\rho, \sigma) = (\rho', \sigma') \qquad (\rho', \mathsf{upd}_p[\![p]\!](\rho', \pi)) \vdash \langle c, \sigma'\rangle \to \sigma''}{(\rho, \pi) \vdash \langle \texttt{begin } v\ p\ c \texttt{ end}, \sigma\rangle \to \sigma''}$$

Execution Relation III

Remarks about rules (call) and (block):

- Static scoping is modelled in (call) by using the environments ρ' and π' (as determined in (block)) from the declaration site of procedure P (and not ρ and π from the calling site)
- In (call), the procedure environment associated with procedure P is extended by a P-entry to handle recursive calls of P:

$$\pi'[P \mapsto (c, \rho', \pi')]$$

Execution Relation IV

Example 13.3

```
begin
  var x; var y;  } v
  proc F is
    begin
      var z;
      z:=x;
      if z=1 then skip                                   c1   cF   p   c
              else x:=x-1; call F; y:=z*y end  } c2
    end
  end;
  x:=2; y:=1; call F  } c0
end
```

Let $\sigma_\emptyset(l) = \rho_\emptyset(x) = \pi_\emptyset(P) = \bot$ for all $l \in Loc, x \in Var, P \in PVar$

Notation: $\sigma_{ijkl} \Leftrightarrow \sigma(0) = i, \sigma(1) = j, \sigma(2) = k, \sigma(3) = l$

Derivation tree for $(\rho_\emptyset, \pi_\emptyset) \vdash \langle c, \sigma_\emptyset \rangle \to \sigma_{1221}$: on the board