Semantics and Verification of Software

Lecture 15: Extension by Blocks and Procedures II (Denotational Semantics)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH AACHEN UNIVERSITY

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/svsw13/

Summer Semester 2013

Outline

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:

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

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 $\mathit{VEnv} := \{\rho \mid \rho : \mathit{Var} \dashrightarrow \mathit{Loc}\}$
  - 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:

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

  denotes the set of procedure environments

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

$$\begin{aligned} \mathrm{upd}_v[\![.]\!] &: VDec \times VEnv \times Sto \to VEnv \times Sto \\ \mathrm{upd}_v[\![\texttt{var } x\texttt{; } v]\!](\rho,\sigma) &:= \mathrm{upd}_v[\![v]\!](\rho[x \mapsto l_x], \sigma[l_x \mapsto 0]) \\ \mathrm{upd}_v[\![\varepsilon]\!](\rho,\sigma) &:= (\rho,\sigma) \\ \mathrm{upd}_p[\![.]\!] &: PDec \times VEnv \times PEnv \to PEnv \\ \mathrm{upd}_p[\![\texttt{proc } P \texttt{ is } c\texttt{; } p]\!](\rho,\pi) &:= \mathrm{upd}_p[\![p]\!](\rho, \pi[P \mapsto (c,\rho,\pi)]) \\ \mathrm{upd}_p[\![\varepsilon]\!](\rho,\pi) &:= \pi \end{aligned}$$

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

Execution Relation I

Definition (Execution relation)

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

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

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

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

$$\text{(if-t)}\ \frac{\langle b, \sigma \circ \rho \rangle \to \mathsf{true} \qquad (\rho, \pi) \vdash \langle c_1, \sigma \rangle \to \sigma'}{(\rho, \pi) \vdash \langle \texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma \rangle \to \sigma'}$$

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

Execution Relation II

Definition (Execution relation; continued)

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

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

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

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

Outline

Handling Variable Declarations

Exactly as in operational semantics:

- Variable environments keep location information:

$$VEnv := \{\rho \mid \rho : Var \dashrightarrow Loc\}$$

with $Loc := \mathbb{N}$

Handling Variable Declarations

Exactly as in operational semantics:

- Variable environments keep location information:

$$\mathit{VEnv} := \{\rho \mid \rho : \mathit{Var} \dashrightarrow \mathit{Loc}\}$$

  with $\mathit{Loc} := \mathbb{N}$

- Effect of variable declaration: update of environment and store

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

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

Statement Semantics Using Locations

- **First step:** reformulation of Definition 5.1 using locations
- **So far:** $\mathfrak{C}[\![.]\!] : \mathit{Cmd} \to (\Sigma \dashrightarrow \Sigma)$

Statement Semantics Using Locations

- **First step:** reformulation of Definition 5.1 using locations
- **So far:** $\mathfrak{C}[\![.]\!] : \mathit{Cmd} \to (\Sigma \dashrightarrow \Sigma)$

Definition 15.1 (Denotational semantics using locations)

The (denotational) semantic functional for statements,

$$\mathfrak{C}'[\![.]\!] : \mathit{Cmd} \to \mathit{VEnv} \to (\mathit{Sto} \dashrightarrow \mathit{Sto}),$$

is given by:

$$\begin{aligned}
\mathfrak{C}'[\![\texttt{skip}]\!]\rho &:= \mathrm{id}_{\mathit{Sto}} \\
\mathfrak{C}'[\![x \texttt{ := } a]\!]\rho\,\sigma &:= \sigma[\rho(x) \mapsto \mathfrak{A}[\![a]\!](\mathrm{lookup}\ \rho\ \sigma)] \\
\mathfrak{C}'[\![c_1 \texttt{;} c_2]\!]\rho &:= (\mathfrak{C}'[\![c_2]\!]\rho) \circ (\mathfrak{C}'[\![c_1]\!]\rho) \\
\mathfrak{C}'[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!]\rho &:= \mathrm{cond}(\mathfrak{B}[\![b]\!] \circ (\mathrm{lookup}\ \rho), \mathfrak{C}'[\![c_1]\!]\rho, \mathfrak{C}'[\![c_2]\!]\rho) \\
\mathfrak{C}'[\![\texttt{while } b \texttt{ do } c]\!]\rho &:= \mathrm{fix}(\Phi)
\end{aligned}$$

where $\mathrm{lookup} : \mathit{VEnv} \to \mathit{Sto} \to \Sigma$ with $\mathrm{lookup}\ \rho\ \sigma := \sigma \circ \rho$ and

$$\begin{aligned}
\Phi : (\mathit{Sto} \dashrightarrow \mathit{Sto}) &\to (\mathit{Sto} \dashrightarrow \mathit{Sto}) : \\
f &\mapsto \mathrm{cond}(\mathfrak{B}[\![b]\!] \circ (\mathrm{lookup}\ \rho), f \circ \mathfrak{C}'[\![c]\!]\rho, \mathrm{id}_{\mathit{Sto}})
\end{aligned}$$

Outline

Procedure Environments

- Procedure environments now store semantic information:
  - So far: $PEnv := \{\pi \mid \pi : PVar \dashrightarrow Cmd \times VEnv \times PEnv\}$
  - Now: $PEnv := \{\pi \mid \pi : PVar \dashrightarrow (Sto \dashrightarrow Sto)\}$, to be used in $\mathfrak{C}''[\![.]\!] : Cmd \to VEnv \to PEnv \to (Sto \dashrightarrow Sto)$

Procedure Environments

- Procedure environments now store semantic information:
  - So far: $PEnv := \{\pi \mid \pi : PVar \dashrightarrow Cmd \times VEnv \times PEnv\}$
  - Now: $PEnv := \{\pi \mid \pi : PVar \dashrightarrow (Sto \dashrightarrow Sto)\}$, to be used in $\mathfrak{C}''[\![.]\!] : Cmd \to VEnv \to PEnv \to (Sto \dashrightarrow Sto)$
- Procedure declarations ("`proc` P `is` c") update procedure environment:

$$\mathrm{upd}_p[\![.]\!] : PDec \times VEnv \times PEnv \to PEnv$$

  - non-recursive case: P not (indirectly) called within c
    $\Rightarrow \pi(P)$ immediately given by $\mathfrak{C}''[\![c]\!]\rho\,\pi$

$$\mathrm{upd}_p[\![\texttt{proc } P \texttt{ is } c\texttt{;} p]\!](\rho,\pi) := \mathrm{upd}_p[\![p]\!](\rho, \pi[P \mapsto \mathfrak{C}''[\![c]\!]\rho\,\pi])$$

  - recursive case: $\pi(P)$ must be a solution of equation
    $f = \mathfrak{C}''[\![c]\!]\rho\,\pi[P \mapsto f]$
    (cf. fixpoint semantics of `while` loop – Slide 5.15)

$$\mathrm{upd}_p[\![\texttt{proc } P \texttt{ is } c\texttt{;} p]\!](\rho,\pi) := \mathrm{upd}_p[\![p]\!](\rho, \pi[P \mapsto \mathrm{fix}(\Psi)])$$

    where $\Psi : (Sto \dashrightarrow Sto) \to (Sto \dashrightarrow Sto) : f \mapsto \mathfrak{C}''[\![c]\!]\rho\,\pi[P \mapsto f]$
  - $\mathrm{upd}_p[\![\varepsilon]\!](\rho,\pi) := \pi$
  - **Remark:** non-recursive is special case of recursive situation

Statement Semantics Including Procedures

So far: $\mathfrak{C}'[\![.]\!] : \mathit{Cmd} \to \mathit{VEnv} \to (\mathit{Sto} \dashrightarrow \mathit{Sto})$

Statement Semantics Including Procedures

So far: $\mathfrak{C}'[\![.]\!] : \mathit{Cmd} \to \mathit{VEnv} \to (\mathit{Sto} \dashrightarrow \mathit{Sto})$

Definition 15.2 (Denotational semantics with procedures)

$$\mathfrak{C}''[\![.]\!] : \mathit{Cmd} \to \mathit{VEnv} \to \mathit{PEnv} \to (\mathit{Sto} \dashrightarrow \mathit{Sto})$$

is given by:

$$\begin{aligned}
\mathfrak{C}''[\![\texttt{skip}]\!]\rho\,\pi &:= \mathrm{id}_{Sto} \\
\mathfrak{C}''[\![x \texttt{ := } a]\!]\rho\,\pi\,\sigma &:= \sigma[\rho(x) \mapsto \mathfrak{A}[\![a]\!](\mathrm{lookup}\ \rho\ \sigma)] \\
\mathfrak{C}''[\![c_1 \texttt{;} c_2]\!]\rho\,\pi &:= (\mathfrak{C}''[\![c_2]\!]\rho\,\pi) \circ (\mathfrak{C}''[\![c_1]\!]\rho\,\pi) \\
\mathfrak{C}''[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!]\rho\,\pi &:= \mathrm{cond}(\mathfrak{B}[\![b]\!] \circ (\mathrm{lookup}\ \rho), \\
&\qquad \mathfrak{C}''[\![c_1]\!]\rho\,\pi, \mathfrak{C}''[\![c_2]\!]\rho\,\pi) \\
\mathfrak{C}''[\![\texttt{while } b \texttt{ do } c]\!]\rho\,\pi &:= \mathrm{fix}(\Phi) \\
\mathfrak{C}''[\![\texttt{call } P]\!]\rho\,\pi &:= \pi(P) \\
\mathfrak{C}''[\![\texttt{begin } v\ p\ c\ \texttt{end}]\!]\rho\,\pi\,\sigma &:= \mathfrak{C}''[\![c]\!]\rho'\,\pi'\,\sigma'
\end{aligned}$$

where
$$\begin{aligned}
\mathrm{upd}_v[\![v]\!](\rho,\sigma) &= (\rho',\sigma') \\
\mathrm{upd}_p[\![p]\!](\rho',\pi) &= \pi' \\
\mathrm{lookup}\ \rho\ \sigma &:= \sigma \circ \rho \\
\Phi(f) &:= \mathrm{cond}(\mathfrak{B}[\![b]\!] \circ (\mathrm{lookup}\ \rho), f \circ \mathfrak{C}''[\![c]\!]\rho\,\pi, \mathrm{id}_{Sto})
\end{aligned}$$

Outline

Example: Non-Recursive Case

Example 15.3 (Non-recursive procedure call)

(also demonstrates static scoping principle)

$c =$
```
begin
  var x;
  proc P is x := x - 1;
  x := 2;
  begin
    var x;
    x := 3;
    call P;
  end;
end
```
c_1 = `x := 2;`, c_2 = `begin var x; x := 3; call P; end;`

- Initial environments/store: $\rho_\emptyset \in \mathit{VEnv}$, $\pi_\emptyset \in \mathit{PEnv}$, $\sigma_\emptyset \in \mathit{Sto}$
- Computation of $\mathfrak{C}''[\![c]\!]\rho_\emptyset\ \pi_\emptyset\ \sigma_\emptyset$: on the board

Example: Recursive Case

Example 15.4 (Recursive procedure call)

$c =$
```
begin
    proc F is
      if x = 1 then
        skip;
      else
        y := x * y;
        x := x - 1;
        call F;
    y := 1;
    call F;
end
```
(c_1 = the procedure body, p = the whole procedure declaration, c_2 = `y := 1; call F;`)

- Initial environments/store: $\rho_1 := \rho_\emptyset[x \mapsto 0, y \mapsto 1] \in \mathit{VEnv}$, $\pi_\emptyset \in \mathit{PEnv}$, $\sigma \in \mathit{Sto}$ (with $\sigma(0) \neq \bot$)
- Computation of $\mathfrak{C}''[\![c]\!]\rho_1\ \pi_\emptyset\ \sigma$: on the board

Outline

Justification of Fixpoint Semantics

Lemma 15.5

1. *(cf. Lemma 6.9)*
   $(Sto \dashrightarrow Sto, \sqsubseteq)$ *is a CCPO where* $f \sqsubseteq g$ *iff for all* $\sigma, \sigma' \in \Sigma$*:*
   $f(\sigma) = \sigma' \Rightarrow g(\sigma) = \sigma'$

Justification of Fixpoint Semantics

Lemma 15.5

1. *(cf. Lemma 6.9)*
   $(Sto \dashrightarrow Sto, \sqsubseteq)$ *is a CCPO where* $f \sqsubseteq g$ *iff for all* $\sigma, \sigma' \in \Sigma$*:*
   $f(\sigma) = \sigma' \Rightarrow g(\sigma) = \sigma'$
2. *(cf. Lemmata 6.13 and 6.16)*
   Let $b \in BExp$, $c \in Cmd$, $\rho \in VEnv$, $\pi \in PEnv$, *and*
   $\Phi : (Sto \dashrightarrow Sto) \to (Sto \dashrightarrow Sto)$ *with*
   $\Phi(f) := \mathrm{cond}(\mathfrak{B}[\![b]\!] \circ (\mathrm{lookup}\ \rho), f \circ \mathfrak{C}''[\![c]\!]\rho\ \pi, \mathrm{id}_{Sto})$. *Then* Φ *is monotonic and continuous w.r.t.* $(Sto \dashrightarrow Sto, \sqsubseteq)$.

Justification of Fixpoint Semantics

Lemma 15.5

1. *(cf. Lemma 6.9)*
   $(Sto \dashrightarrow Sto, \sqsubseteq)$ *is a CCPO where* $f \sqsubseteq g$ *iff for all* $\sigma, \sigma' \in \Sigma$*:*
   $f(\sigma) = \sigma' \Rightarrow g(\sigma) = \sigma'$
2. *(cf. Lemmata 6.13 and 6.16)*
   Let $b \in BExp$, $c \in Cmd$, $\rho \in VEnv$, $\pi \in PEnv$, *and*
   $\Phi : (Sto \dashrightarrow Sto) \to (Sto \dashrightarrow Sto)$ *with*
   $\Phi(f) := \mathrm{cond}(\mathfrak{B}[\![b]\!] \circ (\mathrm{lookup}\ \rho), f \circ \mathfrak{C}''[\![c]\!]\rho\,\pi, \mathrm{id}_{Sto})$. *Then* Φ *is monotonic and continuous w.r.t.* $(Sto \dashrightarrow Sto, \sqsubseteq)$.
3. *Let* `proc` P `is` $c \in PDec$, $\rho \in VEnv$, $\pi \in PEnv$, *and*
   $\Psi : (Sto \dashrightarrow Sto) \to (Sto \dashrightarrow Sto)$ *with* $\Psi(f) := \mathfrak{C}''[\![c]\!]\rho\,\pi[P \mapsto f]$.
   Then Ψ *is monotonic and continuous w.r.t.* $(Sto \dashrightarrow Sto, \sqsubseteq)$.

Proof.

omitted □

Outline

Summary: Blocks and Procedures

- Blocks allow to declare local variables and recursive procedures

Summary: Blocks and Procedures

- Blocks allow to declare local variables and recursive procedures
- Requires concept of locations to support instantiation of variables

Summary: Blocks and Procedures

- Blocks allow to declare local variables and recursive procedures
- Requires concept of locations to support instantiation of variables
- Static scoping: meaning of identifier determined by declaration context (rather than calling context)

Summary: Blocks and Procedures

- Blocks allow to declare local variables and recursive procedures
- Requires concept of locations to support instantiation of variables
- Static scoping: meaning of identifier determined by declaration context (rather than calling context)
- Meaning of variable declaration: storage allocation

Summary: Blocks and Procedures

- Blocks allow to declare local variables and recursive procedures
- Requires concept of locations to support instantiation of variables
- Static scoping: meaning of identifier determined by declaration context (rather than calling context)
- Meaning of variable declaration: storage allocation
- Meaning of procedure call:
  - operationally: execution of procedure body

  $\Rightarrow$ procedure environment records statement ("symbol table")
  - denotationally: application of procedure meaning

  $\Rightarrow$ procedure environment records (partial) store transformation
  - recursive behavior again handled by fixpoint approach

Summary: Blocks and Procedures

- Blocks allow to declare local variables and recursive procedures
- Requires concept of locations to support instantiation of variables
- Static scoping: meaning of identifier determined by declaration context (rather than calling context)
- Meaning of variable declaration: storage allocation
- Meaning of procedure call:
  - operationally: execution of procedure body

  $\Rightarrow$ procedure environment records statement ("symbol table")
  - denotationally: application of procedure meaning

  $\Rightarrow$ procedure environment records (partial) store transformation
  - recursive behavior again handled by fixpoint approach
- Further extensions:
  - axiomatic semantics (for $\texttt{proc}\ P\ \texttt{is}\ c \in PDec$)
    - non-recursive: $\text{(call)}\ \dfrac{\{A\}\, c\, \{B\}}{\{A\}\, \texttt{call}\ P\, \{B\}}$
    - recursive: $\text{(call)}\ \dfrac{\{A\}\, \texttt{call}\ P\, \{B\} \vdash \{A\}\, c\, \{B\}}{\{A\}\, \texttt{call}\ P\, \{B\}}$
  - procedure parameters
  - higher-order procedures