

# Semantyka i weryfikacja programów

**Bartosz Klin**

(slajdy Andrzeja Tarleckiego)

Instytut Informatyki

Wydział Matematyki, Informatyki i Mechaniki

Uniwersytet Warszawski

<http://www.mimuw.edu.pl/~klin>

pok. 5680

[klin@mimuw.edu.pl](mailto:klin@mimuw.edu.pl)

Strona tego wykładu:

<http://www.mimuw.edu.pl/~klin/sem17-18.html>

# Program Semantics & Verification

**Bartosz Klin**

(slides courtesy of Andrzej Tarlecki)

Institute of Informatics  
Faculty of Mathematics, Informatics and Mechanics  
University of Warsaw

<http://www.mimuw.edu.pl/~klin>

office: 5680

[klin@mimuw.edu.pl](mailto:klin@mimuw.edu.pl)

This course:

<http://www.mimuw.edu.pl/~klin/sem17-18.html>

## Blocks & declarations

For now: let's look at declarations of variables:

TINY<sup>+</sup>

$$S \in \mathbf{Stmt} ::= \dots \mid \mathbf{begin} D_V S \mathbf{end}$$
$$D_V \in \mathbf{VDecl} ::= \mathbf{var} x; D_V \mid \varepsilon$$

## Locations

We have identified two roles of variables:

- identifiers, as used in programs
- names for memory cells, where values are stored

To separate them, the structure of the semantic domains has to be changed.

Splitting states into  
*environments and stores*

$$\mathbf{Int} = \{0, 1, -1, 2, -2, \dots\}_{\perp}$$

$$\mathbf{Bool} = \{\mathbf{tt}, \mathbf{ff}\}_{\perp}$$

$$\mathbf{VEnv} = \mathbf{Var} \rightarrow (\mathbf{Loc} + \{??\})$$

$$\mathbf{Store} = \mathbf{Loc} \rightarrow (\mathbf{Int} + \{??\})$$

$$\mathit{combine} : \mathbf{VEnv} \times \mathbf{Store} \rightarrow \mathbf{State}$$

$$\mathit{combine}(\rho_V, s) = \lambda x : \mathbf{Var}. s(\rho_V(x))$$

Inaccurate, but right...

## Semantic functions

$$\mathcal{N}: \mathbf{Num} \rightarrow \mathbf{Int}$$

$$\mathcal{E}: \mathbf{Exp} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{Store}}_{\mathbf{EXP}} \rightarrow (\mathbf{Int} + \{\?\})$$

$$\mathcal{B}: \mathbf{BExp} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{Store}}_{\mathbf{BEXP}} \rightarrow (\mathbf{Bool} + \{\?\})$$

and then for instance

$$\mathcal{E}[[x]] = \lambda\rho_V:\mathbf{VEnv}.\lambda s:\mathbf{Store}.\text{ifte}(\rho_V x = \?, \?, \text{ifte}(s(\rho_V x) = \?, \?, s(\rho_V x)))$$

$$\mathcal{E}[[e_1 + e_2]] = \lambda\rho_V:\mathbf{VEnv}.\lambda s:\mathbf{Store}.\text{ifte}(\mathcal{E}[[e_1]] \rho_V s = \?, \?, \\ \text{ifte}(\mathcal{E}[[e_2]] \rho_V s = \?, \?, \\ \mathcal{E}[[e_1]] \rho_V s + \mathcal{E}[[e_2]] \rho_V s))$$

Looks horrible!  
Reads even worse!

## Bits of notation

- (re)move lambda-abstraction
- use **where**-notation, **let**-notation, explicit **if-then**-notation, etc
- assume that errors ?? propagate

Then:

$$\mathcal{E}[[x]] \rho_V s = s l \text{ where } l = \rho_V x$$

$$\mathcal{E}[[e_1 + e_2]] \rho_V s = n_1 + n_2 \text{ where } n_1 = \mathcal{E}[[e_1]] \rho_V s, n_2 = \mathcal{E}[[e_2]] \rho_V s$$

*Relate this to the previous semantics using combine*

Write down all the other rules for  $\mathcal{E}$  and  $\mathcal{B}$ .  
Spell out their exact meaning  
expanding all the notations in use.

## Statements

One could work with “big states”

$$\mathcal{S}: \text{Stmt} \rightarrow \underbrace{\text{VEnv} \times \text{Store}}_{\text{STMT}} \rightarrow (\text{VEnv} \times \text{Store} + \{\?\})$$

BUT:

*Statements do not modify the environment!*

Hence:

$$\mathcal{S}: \text{Stmt} \rightarrow \underbrace{\text{VEnv} \rightarrow \text{Store}}_{\text{STMT}} \rightarrow (\text{Store} + \{\?\})$$

## Semantic clauses

$\mathcal{S}[[x := e]] \rho_V s = s[l \mapsto n]$  where  $l = \rho_V x, n = \mathcal{E}[[e]] \rho_V s$

$\mathcal{S}[[\text{skip}]] \rho_V s = s$

$\mathcal{S}[[S_1; S_2]] \rho_V s = \mathcal{S}[[S_2]] \rho_V s_1$  where  $s_1 = \mathcal{S}[[S_1]] \rho_V s$

$\mathcal{S}[[\text{if } b \text{ then } S_1 \text{ else } S_2]] \rho_V s = \text{let } v = \mathcal{B}[[b]] \rho_V s \text{ in}$   
    if  $v = \text{tt}$  then  $\mathcal{S}[[S_1]] \rho_V s$   
    if  $v = \text{ff}$  then  $\mathcal{S}[[S_2]] \rho_V s$

$\mathcal{S}[[\text{while } b \text{ do } S]] \rho_V s = \text{let } v = \mathcal{B}[[b]] \rho_V s \text{ in}$   
    if  $v = \text{ff}$  then  $s$   
    if  $v = \text{tt}$  then  $\mathcal{S}[[\text{while } b \text{ do } S]] \rho_V s'$   
        where  $s' = \mathcal{S}[[S]] \rho_V s$



## More compact version

Relying on propagation of errors  $??$  to be also built into composition of functions from **Store** to **Store** +  $\{??\}$ :

$$\mathcal{S}[[x := e]] \rho_V s = s[l \mapsto n] \text{ where } l = \rho_V x, n = \mathcal{E}[[e]] \rho_V s$$

$$\mathcal{S}[[\text{skip}]] \rho_V = id_{\text{Store}}$$

$$\mathcal{S}[[S_1; S_2]] \rho_V = \mathcal{S}[[S_1]] \rho_V; \mathcal{S}[[S_2]] \rho_V$$

$$\mathcal{S}[[\text{if } b \text{ then } S_1 \text{ else } S_2]] \rho_V = \text{cond}(\mathcal{B}[[b]] \rho_V, \mathcal{S}[[S_1]] \rho_V, \mathcal{S}[[S_2]] \rho_V)$$

$$\mathcal{S}[[\text{while } b \text{ do } S]] \rho_V = \text{cond}(\mathcal{B}[[b]] \rho_V, \mathcal{S}[[S]] \rho_V; \mathcal{S}[[\text{while } b \text{ do } S]] \rho_V, id_{\text{Store}})$$

The missing clause for blocks in a moment

## Declarations modify environments

$$\mathcal{D}_V : \mathbf{VDecl} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{Store} \rightarrow (\mathbf{VEnv} \times \mathbf{Store} + \{??\})}_{\mathbf{VDECL}}$$

$$\mathcal{D}_V[\varepsilon] \rho_V s = \langle \rho_V, s \rangle$$

$$\mathcal{D}_V[\mathbf{var} \ x; D_V] \rho_V s = \mathcal{D}_V[D_V] \rho'_V s' \\ \text{where } l = \mathit{newloc}(s), \rho'_V = \rho_V[x \mapsto l], s' = s[l \mapsto ??]$$

**Trouble:** We want  $\mathit{newloc} : \mathbf{Store} \rightarrow \mathbf{Loc}$  to yield a new, unused location. This cannot be defined under the definitions given so far. Solution: more information in stores is needed to determine used and unused locations.

## Simple solution

Take:

$$\mathbf{Loc} = \{0, 1, 2, \dots\}$$

Add to each store a pointer to the next unused location:

$$\mathbf{Store} = (\mathbf{Loc} + \{next\}) \rightarrow (\mathbf{Int} + \{??\})$$

Semantic clauses then:

$$\mathcal{D}_V[\varepsilon] \rho_V s = \langle \rho_V, s \rangle$$

$$\mathcal{D}_V[\mathbf{var} \ x; D_V] \rho_V s =$$

$$\mathcal{D}_V[D_V] \rho'_V s' \text{ where } l = s \ next, \rho'_V = \rho_V[x \mapsto l], s' = s[l \mapsto ??, next \mapsto l + 1]$$

## Semantics of blocks

$$\mathcal{S}[\mathbf{begin } D_V \ S \ \mathbf{end}] \rho_V \ s = \mathcal{S}[S] \rho'_V \ s' \text{ where } \langle \rho'_V, s' \rangle = \mathcal{D}_V[D_V] \rho_V \ s$$

*The scope of a declaration is the block it occurs in  
with holes resulting from redeclarations of the same variable within it*

For instance

**begin var  $y$ ; var  $x$   $x := 1$ ; begin var  $x$   $y := 2$ ;  $x := 5$  end;  $y := x$  end**

may be marked as follows to indicate the relevant declarations:

**begin var  $y$ ; var  $x$   $x$   $:= 1$ ; **begin** var  $x$   $y$   $:= 2$ ;  $x$   $:= 5$  **end**;  $y$   $:=$   $x$  **end****

# Procedures

TINY++

$$S \in \mathbf{Stmt} ::= \dots \mid \mathbf{begin} \ D_V \ D_P \ S \ \mathbf{end} \mid \mathbf{call} \ p$$
$$D_V \in \mathbf{VDecl} ::= \mathbf{var} \ x; D_V \mid \varepsilon$$
$$D_P \in \mathbf{PDecl} ::= \mathbf{proc} \ p \ \mathbf{is} \ (S); D_P \mid \varepsilon$$

- binding of global variables
- recursion

# Binding of global variables

## Static binding

```
begin var y;  
  var x;  
  proc p is (x := 1);  
  begin var x;  
    x := 3;  
    call p;  
    y := x  
  end  
end
```

## Dynamic binding

```
begin var y;  
  var x;  
  proc p is (x := 1);  
  begin var x;  
    x := 3;  
    call p; %%% with x  
    y := x  
  end  
end
```

# Semantic domains and functions

## Dynamic binding

$$\mathbf{PEnv} = \mathbf{IDE} \rightarrow (\mathbf{PROC}_0 + \{??\})$$

$$\mathbf{PROC}_0 = \mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow \mathbf{Store} \rightarrow (\mathbf{Store} + \{??\})$$

$$\mathcal{S}: \mathbf{Stmt} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow \mathbf{Store} \rightarrow (\mathbf{Store} + \{??\})}_{\mathbf{STMT}}$$

$$\mathcal{D}_P: \mathbf{PDecl} \rightarrow \underbrace{\mathbf{PEnv} \rightarrow (\mathbf{PEnv} + \{??\})}_{\mathbf{PDECL}}$$

## Semantic clauses

$\mathcal{S}[[x := e]] \rho_V \rho_P s = s[l \mapsto n]$  where  $l = \rho_V x, n = \mathcal{E}[[e]] \rho_V s$

$\mathcal{S}[[\text{skip}]] \rho_V \rho_P = id_{\text{Store}}$

$\mathcal{S}[[S_1; S_2]] \rho_V \rho_P = \mathcal{S}[[S_1]] \rho_V \rho_P; \mathcal{S}[[S_2]] \rho_V \rho_P$

$\mathcal{S}[[\text{if } b \text{ then } S_1 \text{ else } S_2]] \rho_V \rho_P = \text{cond}(\mathcal{B}[[b]] \rho_V, \mathcal{S}[[S_1]] \rho_V \rho_P, \mathcal{S}[[S_2]] \rho_V \rho_P)$

$\mathcal{S}[[\text{while } b \text{ do } S]] \rho_V \rho_P =$   
 $\text{cond}(\mathcal{B}[[b]] \rho_V, \mathcal{S}[[S]] \rho_V \rho_P; \mathcal{S}[[\text{while } b \text{ do } S]] \rho_V \rho_P, id_{\text{Store}})$

$\mathcal{S}[[\text{call } p]] \rho_V \rho_P = P \rho_V \rho_P$  where  $P = \rho_P p$

$\mathcal{S}[[\text{begin } D_V \ D_P \ S \ \text{end}]] \rho_V \rho_P s =$   
 $\mathcal{S}[[S]] \rho'_V \rho'_P s'$  where  $\langle \rho'_V, s' \rangle = \mathcal{D}_V[[D_V]] \rho_V s, \rho'_P = \mathcal{D}_P[[D_P]] \rho_P$

$\mathcal{D}_P[[\epsilon]] = id_{\text{PEnv}}$

$\mathcal{D}_P[[\text{proc } p \text{ is } (S); D_P]] \rho_P = \mathcal{D}_P[[D_P]] \rho_P[p \mapsto \mathcal{S}[[S]]]$



## Recursion

```
begin var  $x$ ;  
  proc  $NO$  is (if  $101 \leq x$  then  $x := x - 10$   
    else ( $x := x + 11$ ; call  $NO$ ; call  $NO$ ) );  
   $x := 54$ ;  
  call  $NO$   
end
```

# Semantic domains and functions

## Static binding

$$\mathbf{PEnv} = \mathbf{IDE} \rightarrow (\mathbf{PROC}_0 + \{??\})$$
$$\mathbf{PROC}_0 = \mathbf{Store} \rightarrow (\mathbf{Store} + \{??\})$$
$$\mathcal{S}: \mathbf{Stmt} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow \mathbf{Store} \rightarrow (\mathbf{Store} + \{??\})}_{\mathbf{STMT}}$$
$$\mathcal{D}_P: \mathbf{PDecl} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow (\mathbf{PEnv} + \{??\})}_{\mathbf{PDECL}}$$

## Semantic clauses

$\mathcal{S}[[x := e]] \rho_V \rho_P s = s[l \mapsto n]$  where  $l = \rho_V x, n = \mathcal{E}[[e]] \rho_V s$

$\mathcal{S}[[\text{skip}]] \rho_V \rho_P = id_{\text{Store}}$

$\mathcal{S}[[S_1; S_2]] \rho_V \rho_P = \mathcal{S}[[S_1]] \rho_V \rho_P; \mathcal{S}[[S_2]] \rho_V \rho_P$

$\mathcal{S}[[\text{if } b \text{ then } S_1 \text{ else } S_2]] \rho_V \rho_P = \text{cond}(\mathcal{B}[[b]] \rho_V, \mathcal{S}[[S_1]] \rho_V \rho_P, \mathcal{S}[[S_2]] \rho_V \rho_P)$

$\mathcal{S}[[\text{while } b \text{ do } S]] \rho_V \rho_P =$   
 $\text{cond}(\mathcal{B}[[b]] \rho_V, \mathcal{S}[[S]] \rho_V \rho_P; \mathcal{S}[[\text{while } b \text{ do } S]] \rho_V \rho_P, id_{\text{Store}})$

$\mathcal{S}[[\text{call } p]] \rho_V \rho_P = P$  where  $P = \rho_P p$

$\mathcal{S}[[\text{begin } D_V D_P S \text{ end}]] \rho_V \rho_P s =$   
 $\mathcal{S}[[S]] \rho'_V \rho'_P s'$  where  $\langle \rho'_V, s' \rangle = \mathcal{D}_V[[D_V]] \rho_V s, \rho'_P = \mathcal{D}_P[[D_P]] \rho'_V \rho_P$

$\mathcal{D}_P[[\epsilon]] \rho_V = id_{\text{PEnv}}$

$\mathcal{D}_P[[\text{proc } p \text{ is } (S); D_P]] \rho_V \rho_P =$   
 $\mathcal{D}_P[[D_P]] \rho_V \rho_P[p \mapsto P]$  where  $P = \mathcal{S}[[S]] \rho_V \rho_P[p \mapsto P]$

## Parameters

*Parameter passing:*

- call by value
- call by variable
- call by name

We will do static binding only

$$S \in \mathbf{Stmt} ::= \dots \mid \mathbf{begin} D_V D_P S \mathbf{end}$$
$$\mid \mathbf{call} p \mid \mathbf{call} p(\mathbf{vr} x) \mid \mathbf{call} p(\mathbf{vl} e) \mid \mathbf{call} p(\mathbf{nm} e)$$
$$D_V \in \mathbf{VDecl} ::= \mathbf{var} x; D_V \mid \varepsilon$$
$$D_P \in \mathbf{PDecl} ::= \mathbf{proc} p \mathbf{is} (S); D_P \mid \mathbf{proc} p(\mathbf{vr} x) \mathbf{is} (S); D_P$$
$$\mid \mathbf{proc} p(\mathbf{vl} x) \mathbf{is} (S); D_P \mid \mathbf{proc} p(\mathbf{nm} x) \mathbf{is} (S); D_P \mid \varepsilon$$

## Semantic domains

$$\mathbf{PEnv} = \mathbf{IDE} \rightarrow (\mathbf{PROC}_0 + \mathbf{PROC}_1^{\text{vr}} + \mathbf{PROC}_1^{\text{vl}} + \mathbf{PROC}_1^{\text{nm}} + \{??\})$$

$$\mathbf{PROC}_0 = \mathbf{Store} \rightarrow (\mathbf{Store} + \{??\})$$

$$\mathbf{PROC}_1^{\text{vr}} = \mathbf{Loc} \rightarrow \mathbf{PROC}_0$$

$$\mathbf{PROC}_1^{\text{vl}} = \mathbf{Int} \rightarrow \mathbf{PROC}_0$$

$$\mathbf{PROC}_1^{\text{nm}} = (\mathbf{Store} \rightarrow (\mathbf{Int} + \{??\})) \rightarrow \mathbf{PROC}_0$$

## Semantic functions

As before:

$$\begin{array}{l} \mathcal{S}: \mathbf{Stmt} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow \mathbf{Store} \rightarrow (\mathbf{Store} + \{??\})}_{\mathbf{STMT}} \\ \mathcal{D}_P: \mathbf{PDecl} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow (\mathbf{PEnv} + \{??\})}_{\mathbf{PDECL}} \end{array}$$

## Semantic clauses

*No parameters*

$\mathcal{S}[\mathbf{call} \ p] \ \rho_V \ \rho_P = P$  where  $P = \rho_P \ p \in \mathbf{PROC}_0$

$\mathcal{D}_P[\mathbf{proc} \ p \ \mathbf{is} \ (S); \ D_P] \ \rho_V \ \rho_P =$

$\mathcal{D}_P[D_P] \ \rho_V \ \rho_P [p \mapsto P]$  where  $P = \mathcal{S}[S] \ \rho_V \ \rho_P [p \mapsto P]$

*Parameter called by variable*

$\mathcal{S}[\mathbf{call} \ p(\mathbf{vr} \ y)] \ \rho_V \ \rho_P = P \ l$  where  $P = \rho_P \ p \in \mathbf{PROC}_1^{\mathbf{vr}}$ ,  $l = \rho_V \ y$

$\mathcal{D}_P[\mathbf{proc} \ p(\mathbf{vr} \ x) \ \mathbf{is} \ (S); D_P] \ \rho_V \ \rho_P =$

$\mathcal{D}_P[D_P] \ \rho_V \ \rho_P[p \mapsto P]$  where  $P \ l = \mathcal{S}[S] \ \rho_V[x \mapsto l] \ \rho_P[p \mapsto P]$



*Parameter called by value*

$\mathcal{S}[\mathbf{call} \ p(\mathbf{vl} \ e)] \ \rho_V \ \rho_P \ s = P \ n \ s$  where  $P = \rho_P \ p \in \mathbf{PROC}_1^{\mathbf{vl}}$ ,  $n = \mathcal{E}[e] \ \rho_V \ s$

$\mathcal{D}_P[\mathbf{proc} \ p(\mathbf{vl} \ x) \ \mathbf{is} \ (S); D_P] \ \rho_V \ \rho_P =$

$\mathcal{D}_P[D_P] \ \rho_V \ \rho_P [p \mapsto P]$  where

$P \ n \ s = \mathcal{S}[S] \ \rho'_V \ \rho_P [p \mapsto P] \ s'$  where

$l = s \ next, \ \rho'_V = \rho_V [x \mapsto l], \ s' = s[l \mapsto n, \ next \mapsto l + 1]$

*Parameter called by name*

$\mathcal{S}[\mathbf{call} \ p(\mathbf{nm} \ e)] \ \rho_V \ \rho_P = P \ (\mathcal{E}[e] \ \rho_V)$  where  $P = \rho_P \ p \in \mathbf{PROC}_1^{\mathbf{nm}}$

$\mathcal{D}_P[\mathbf{proc} \ p(\mathbf{nm} \ x) \ \mathbf{is} \ (S); D_P] \ \rho_V \ \rho_P =$

$\mathcal{D}_P[D_P] \ \rho_V \ \rho_P [p \mapsto P]$  where  $P \ E = \mathcal{S}[S] \ \rho_V [x \mapsto E] \ \rho_P [p \mapsto P]$

OOOPS!

$\rho_V [x \mapsto E] \notin \mathbf{VEnv}$

*Corrections necessary!*

$\mathbf{VEnv} = \mathbf{Var} \rightarrow (\mathbf{Loc} + (\mathbf{Store} \rightarrow (\mathbf{Int} + \{??\}))) + \{??\}$

$\mathcal{E}[x] \ \rho_V \ s = \mathbf{let} \ v = \rho_V \ x \ \mathbf{in} \ \mathbf{if} \ v \in \mathbf{Loc} \ \mathbf{then} \ s \ v$

$\mathbf{if} \ v \in (\mathbf{Store} \rightarrow (\mathbf{Int} + \{??\})) \ \mathbf{then} \ v \ s$

This allows for evaluation of called-by-name parameters,  
but not for assignments to variables passed in such a way