Blocks & declarations

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

TINY$^+$

$$S \in \mathbf{Stmt} ::= \ldots \mid \mathbf{begin}\ D_V\ S\ \mathbf{end}$$

$$D_V \in \mathbf{VDecl} ::= \mathbf{var}\ x; D_V \mid \varepsilon$$

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*

$$\mathbf{Int} = \{0, 1, -1, 2, -2, \ldots\}$$

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

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

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

$$combine : \mathbf{VEnv} \times \mathbf{Store} \to \mathbf{State}$$

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

Inaccurate, but right...

Semantic functions

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

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

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

and then for instance

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

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

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"

$$\mathcal{S} : \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \times \mathbf{Store} \rightharpoonup (\mathbf{VEnv} \times \mathbf{Store} + \{??\})}_{\mathbf{STMT}}$$

- environments flow "statically", following program block structure
- stores change "dynamically", following program execution

BUT:

Statements do not modify the environment!

Hence:

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

$$\mathcal{S}[\![S]\!]\,\rho_v : \mathbf{Store} \rightharpoonup (\mathbf{Store} + \{??\})$$

Semantic clauses

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

$\mathcal{S}[\![\mathbf{skip}]\!]\,\rho_V\;s = s$

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

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

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

fixed-point equation for $\mathcal{S}[\![\mathbf{while}\;b\;\mathbf{do}\;S]\!]\,\rho_V$ in $\mathbf{Store} \rightharpoonup (\mathbf{Store} + \{??\})$

Relate this to the previous semantics using $combine$

More compact version

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

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

$$\mathcal{S}[\![\mathbf{skip}]\!]\,\rho_V = id_{\mathbf{Store}}$$

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

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

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

The missing clause for blocks in a moment

Declarations modify environments

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

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

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

$$\text{where } l = newloc(s), \rho'_V = \rho_V[x \mapsto l], s' = s[l \mapsto ??]$$

Trouble: We want $newloc\colon \mathbf{Store} \to \mathbf{Loc}$ 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:

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

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

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

Semantic clauses then:

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

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

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

$$\mathbf{begin\ var}\ y;\mathbf{var}\ x\ x:=1;\mathbf{begin\ var}\ x\ y:=2;x:=5\ \mathbf{end};y:=x\ \mathbf{end}$$

may be marked as follows to indicate the relevant declarations:

$$\mathbf{begin}\ \boxed{\mathbf{var}\ y};\boxed{\mathbf{var}\ x}\ \boxed{x}:=1;\mathbf{begin}\ \boxed{\mathbf{var}\ x}\ \boxed{y}:=2;\boxed{x}:=5\ \mathbf{end};\boxed{y}:=\boxed{x}\ \mathbf{end}$$

Procedures

naming statements/blocks for multiple use

TINY^{++}

$$S \in \mathbf{Stmt} ::= \ldots \mid \mathbf{begin}\ D_V\ D_P\ S\ \mathbf{end} \mid \mathbf{call}\ p$$

$$D_V \in \mathbf{VDecl} ::= \mathbf{var}\ x; D_V \mid \varepsilon$$

$$D_P \in \mathbf{PDecl} ::= \mathbf{proc}\ p\ \mathbf{is}\ (S); D_P \mid \varepsilon$$

- 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

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

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

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

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

Semantic clauses

$\mathcal{S}$: **Stmt** $\to$ **STMT**

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

$$\mathcal{S}[\![\mathbf{skip}]\!]\,\rho_V\,\rho_P = id_{\mathbf{Store}}$$

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

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

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

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

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

$\mathcal{D}_P$: **PDecl** $\to$ **PDECL**

$$\mathcal{D}_P[\![\varepsilon]\!] = id_{\mathbf{PEnv}}$$

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

Recursion

$$
\begin{array}{l}
\textbf{begin var } x; \\
\qquad \textbf{proc } NO \textbf{ is } (\textbf{if } 101 \leq x \textbf{ then } x := x - 10 \\
\qquad\qquad\qquad \textbf{else } (x := x + 11; \textbf{call } NO; \textbf{call } NO)\ ); \\
\qquad x := 54; \\
\qquad \textbf{call } NO \\
\textbf{end}
\end{array}
$$

Semantic domains and functions

Static binding

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

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

$$\mathcal{S} \colon \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Store} \rightharpoonup (\mathbf{Store} + \{??\})}_{\mathbf{STMT}}$$

$$\mathcal{D}_P \colon \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to (\mathbf{PEnv} + \{??\})}_{\mathbf{PDECL}}$$

Semantic clauses

$\mathcal{S}: \textbf{Stmt} \to \textbf{STMT}$

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

$$\mathcal{S}[\![\textbf{skip}]\!]\,\rho_V\,\rho_P = id_{\textbf{Store}}$$

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

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

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

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

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

$\mathcal{D}_P: \textbf{PDecl} \to \textbf{PDECL}$

$$\mathcal{D}_P[\![\varepsilon]\!]\,\rho_V = id_{\textbf{PEnv}}$$

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

Recursion

Static binding, procedure calls:

$\mathcal{S}[\![\mathbf{call}\ p]\!]\,\rho_V\,\rho_P = P$ where $P = \rho_P\,p$

- Static binding, no recursive calls:

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

- Static binding, recursive calls permitted:

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

or perhaps using fixed-point operator explicitly:

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

- Dynamic binding, recursion "for free" (though only seemingly so):

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

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

Parameters

Parameter passing:

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

We will do **static binding only**

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

Call by name

$$
\begin{array}{l}
\textbf{begin var } sum;\ \textbf{var } i \\
\qquad \textbf{proc } ADD(\textbf{vl } a, b, \textbf{vr } x, \textbf{nm } step, f) \\
\qquad\quad \textbf{is } (sum := 0; x := a; \\
\qquad\qquad \textbf{while } x \leq b \textbf{ do } (sum := sum + f; x := step)\ ) \\
\qquad \textbf{call } ADD(\textbf{vl } 1, 10, \textbf{vr } i, \textbf{nm } i+1, i); \qquad \{sum = 55\} \\
\qquad \textbf{call } ADD(\textbf{vl } 1, 10, \textbf{vr } i, \textbf{nm } 2*i, i); \qquad \{sum = 15\} \\
\qquad \textbf{call } ADD(\textbf{vl } 1, 10, \textbf{vr } i, \textbf{nm } i+2, i*i) \quad \{sum = 165\} \\
\textbf{end}
\end{array}
$$

Semantic domains

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

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

$$\mathbf{PROC}_1^{\mathsf{vl}} = \mathbf{Int} \to \mathbf{PROC}_0$$

$$\mathbf{PROC}_1^{\mathsf{vr}} = \mathbf{Loc} \to \mathbf{PROC}_0$$

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

Semantic functions

As before:

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

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

Semantic clauses

No parameters

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

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

Parameter called by value

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

$$\mathcal{D}_P[\![\mathbf{proc}\ p(\mathbf{vl}\ x)\ \mathbf{is}\ (S); D_P]\!]\,\rho_V\,\rho_P =$$
$$\mathcal{D}_P[\![D_P]\!]\,\rho_V\,\rho_P[p \mapsto P] \text{ where}$$
$$P\,n\,s = \mathcal{S}[\![S]\!]\,\rho'_V\,\rho_P[p \mapsto P]\,s' \text{ where}$$
$$l = s\,next,\ \rho'_V = \rho_V[x \mapsto l],\ s' = s[l \mapsto n, next \mapsto l+1]$$

Parameter called by variable

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

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

Parameter called by name

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

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

$\qquad \mathcal{D}_P[\![D_P]\!]\,\rho_V\,\rho_P[p \mapsto P]$ 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!

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

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

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

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