Motivating continuation semantics

Input/output

$\textsc{Tiny}^{+++}$

$$S \in \mathbf{Stmt} ::= \ldots \mid \mathbf{read}\ x \mid \mathbf{write}\ e$$

Semantic domains

$$\mathbf{Stream} = \mathbf{Int} \times \mathbf{Stream} + \{\mathbf{eof}\}$$
$$\mathbf{Input} = \mathbf{Stream}$$
$$\mathbf{Output} = \mathbf{Stream}$$
$$\mathbf{State} = \mathbf{Store} \times \mathbf{Input} \times \mathbf{Output}$$

WARNING: new, different $\mathbf{State}$!
forget about the old one!

Actually:

$$\boxed{\mathbf{Stream} = (\mathbf{Int} \otimes_L \mathbf{Stream}) \oplus \{\mathbf{eof}\}_\perp}$$

$\mathbf{Stream}$ includes:

- finite lists, ended by $\mathbf{eof}$
- unfinished finite lists
- infinite lists

Fixed-point definitions work in $\mathbf{Stream}$

Semantic functions

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

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

Only one clause to modify here:

$$\boxed{\mathcal{E}[\![x]\!]\,\rho_V\,\langle s, i, o\rangle = s\,l \text{ where } l = \rho_V\,x}$$

Semantics of statements

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

Again, one clause to change:

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

(plus a similar change in $\mathcal{D}_V[\![\mathbf{var}\ x; D_V]\!]\ \ldots = \ldots$) and two clauses to add:

$$\mathcal{S}[\![\mathbf{read}\ x]\!]\,\rho_V\,\rho_P\,\langle s, i, o\rangle = \langle s[l \mapsto n], i', o\rangle \text{ where } l = \rho_V\,x, \langle n, i'\rangle = i$$

$$\mathcal{S}[\![\mathbf{write}\ e]\!]\,\rho_V\,\rho_P\,\langle s, i, o\rangle = \langle s, i, \langle n, o\rangle\rangle \text{ where } n = \mathcal{E}[\![e]\!]\,\rho_V\,\langle s, i, o\rangle$$

$\boxed{\langle n, i'\rangle = i}$ yields $??$ when $i = \mathbf{eof}$

Programs

New syntactic domain:

$$\boxed{\mathbf{Prog} ::= \mathbf{prog}\ S}$$

with obvious semantic function:

$$\boxed{\mathcal{P} : \mathbf{Prog} \to \underbrace{\mathbf{Input} \rightharpoonup (\mathbf{Output} + \{??\})}_{\mathbf{PROG}}}$$

given by:

$$\boxed{\begin{array}{l}\mathcal{P}[\![\mathbf{prog}\ S]\!]\, i = o' \text{ where } \mathcal{S}[\![S]\!]\, \rho_V^{\emptyset}\, \rho_P^{\emptyset}\, \langle s^{\emptyset}, i, \mathbf{eof}\rangle = \langle s', i', o'\rangle, \\ \qquad\qquad\qquad\qquad \rho_V^{\emptyset}\, x = ??, \rho_P^{\emptyset}\, p = ??, s^{\emptyset}\, next = 0, s^{\emptyset}\, l = ??\end{array}}$$

Looks okay, but...

- *Do we want to write in the reverse order?*
- *Do we want to disregard outputs from infinite loops?*
- *Don't we want to disallow statements to erase or modify earlier outputs?*

denotational semantics so far: **direct semantics**

Other problems:

- exits, jumps, exceptions, ...

Continuation semantics

History, late 60s, 70s:

- Wadsworth (PRG/Oxford) 1971-73
- Mazurkiewicz (Warsaw) 1969-71

Changing philosophy

From: What happens now?

To: What the overall result will be?

Changing philosophy

Direct semantics

$\mathcal{S}[\![S]\!]$: "a present" (a current state) $\mapsto$ "a future present" (a future state)

Continuation semantics

$\mathcal{S}[\![S]\!]$: "a future" (from a current state) $\mapsto$ "a past future" (from a past state)

Direct semantics

begin $\ldots$; $\ldots$; $\ldots$ **end**

$$s^{\emptyset} \xrightarrow{\mathcal{S}[\![\ldots]\!]} s_i \xrightarrow{\mathcal{S}[\![\ldots]\!]} s_j \xrightarrow{\mathcal{S}[\![\ldots]\!]} s' \rightsquigarrow \text{“overall result”}$$

Continuation semantics

begin $\ldots$; $\ldots$; $\ldots$ **end**

$$\kappa^{\emptyset} : \rightsquigarrow \text{“overall result”}$$

$$\kappa_j \xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa^{\emptyset}$$

$$\kappa_j : \rightsquigarrow \text{“overall result”}$$

$$\kappa_i \xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa_j$$

$$\kappa_i : \rightsquigarrow \text{“overall result”}$$

$$\kappa' \xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa_i$$

$$\kappa' : \rightsquigarrow \text{“overall result”}$$

Continuations

1st approximation

$$\mathbf{Cont} = \mathbf{State} \to \mathbf{Res}$$

Now:

- states do not include outputs
- overall results are outputs
- these are continuations for statements; semantics for statements is given by:

$$\mathbf{State} = \mathbf{Store} \times \mathbf{Input}$$
$$\mathbf{Res} = \mathbf{Output}$$

$$\mathcal{S} : \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont} \to \mathbf{Cont}}_{\mathbf{STMT}}$$

That is: $\mathbf{STMT} = \mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont} \to \mathbf{State} \to \mathbf{Res}$

Expression and declaration continuations

- continuations for other syntactic categories may be additionally parameterised by whatever these pass on:
  - expressions pass on values, so

$$\boxed{\begin{array}{ll} \mathbf{Cont_E} = \mathbf{Int} \to \mathbf{State} \to \mathbf{Res} & (= \mathbf{Int} \to \mathbf{Cont}) \\ \mathbf{Cont_B} = \mathbf{Bool} \to \mathbf{State} \to \mathbf{Res} & (= \mathbf{Bool} \to \mathbf{Cont}) \end{array}}$$

  - declarations pass on environments, so

$$\boxed{\begin{array}{ll} \mathbf{Cont_{D_V}} = \mathbf{VEnv} \to \mathbf{State} \to \mathbf{Res} & (= \mathbf{VEnv} \to \mathbf{Cont}) \\ \mathbf{Cont_{D_P}} = \mathbf{PEnv} \to \mathbf{State} \to \mathbf{Res} & (= \mathbf{PEnv} \to \mathbf{Cont}) \end{array}}$$

TINY^{+++}

$$
\begin{array}{rcl}
N \in \mathbf{Num} & ::= & 0 \mid 1 \mid 2 \mid \cdots \\
x \in \mathbf{Var} & ::= & \cdots \\
p \in \mathbf{IDE} & ::= & \cdots \\
e \in \mathbf{Exp} & ::= & N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2 \\
b \in \mathbf{BExp} & ::= & \mathbf{true} \mid \mathbf{false} \mid e_1 \leq e_2 \mid \neg b' \mid b_1 \wedge b_2 \\
S \in \mathbf{Stmt} & ::= & x := e \mid \mathbf{skip} \mid S_1; S_2 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2 \mid \mathbf{while}\ b\ \mathbf{do}\ S' \\
 & & \mid \mathbf{begin}\ D_V\ D_P\ S\ \mathbf{end} \mid \mathbf{call}\ p \mid \mathbf{read}\ x \mid \mathbf{write}\ e \\
D_V \in \mathbf{VDecl} & ::= & \mathbf{var}\ x; D_V \mid \varepsilon \\
D_P \in \mathbf{PDecl} & ::= & \mathbf{proc}\ p\ \mathbf{is}\ (S); D_P \mid \varepsilon \\
\mathbf{Prog} & ::= & \mathbf{prog}\ S
\end{array}
$$

Semantic domains

$$\mathbf{Int} = \ldots$$
$$\mathbf{Int}^{??} = \mathbf{Int} + \{??\}$$
$$\mathbf{Bool} = \ldots$$
$$\mathbf{Bool}^{??} = \mathbf{Bool} + \{??\}$$
$$\mathbf{Loc} = \ldots$$
$$\mathbf{Store} = \ldots$$
$$\mathbf{VEnv} = \ldots$$

$$\mathbf{Input} = (\mathbf{Int} \otimes_L \mathbf{Input}) \oplus \{\mathbf{eof}\}_\perp$$
$$\mathbf{State} = \mathbf{Store} \times \mathbf{Input}$$
$$\mathbf{Output} = (\mathbf{Int} \otimes_L \mathbf{Output}) \oplus \{\mathbf{eof}, ??\}_\perp$$

outputs may end also with ??

expressions may pass ??

$$\mathbf{Cont} = \mathbf{State} \to \mathbf{Output}$$
$$\mathbf{Cont_E} = \mathbf{Int}^{??} \to \mathbf{Cont}$$
$$\mathbf{Cont_B} = \mathbf{Bool}^{??} \to \mathbf{Cont}$$
$$\mathbf{Cont_{DV}} = \mathbf{VEnv} \to \mathbf{Cont}$$
$$\mathbf{Cont_{DP}} = \mathbf{PEnv} \to \mathbf{Cont}$$

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

Semantic functions

$$\mathcal{E}: \mathbf{Exp} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont_E} \to \mathbf{Cont}}_{\mathbf{EXP}}$$

$$\mathcal{B}: \mathbf{BExp} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont_B} \to \mathbf{Cont}}_{\mathbf{BEXP}}$$

$$\mathcal{S}: \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont} \to \mathbf{Cont}}_{\mathbf{STMT}}$$

$$\mathcal{D}_V: \mathbf{VDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont_{D_V}} \to \mathbf{Cont}}_{\mathbf{VDECL}}$$

$$\mathcal{D}_P: \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont_{D_P}} \to \mathbf{Cont}}_{\mathbf{PDECL}}$$

$$\mathcal{P}: \mathbf{Prog} \to \underbrace{\mathbf{Input} \to \mathbf{Output}}_{\mathbf{PROG}}$$

Sample semantic clauses

Programs:

$$\mathcal{P}[\![\mathbf{prog}\ S]\!]\, i = \mathcal{S}[\![S]\!]\, \rho_V^{\emptyset}\, \rho_P^{\emptyset}\, \kappa^{\emptyset}\, \langle s^{\emptyset}, i\rangle$$

$$\text{where } \rho_V^{\emptyset}\, x = ??, \rho_P^{\emptyset}\, p = ??, \kappa^{\emptyset}\, s = \mathbf{eof}, s^{\emptyset}\, next = 0, s^{\emptyset}\, l = ??$$

Declarations:

$$\mathcal{D}_P[\![\varepsilon]\!]\, \rho_V\, \rho_P\, \kappa_P = \kappa_P\, \rho_P$$

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

$$\begin{aligned}\mathcal{D}_V[\![\mathbf{var}\ x; D_V]\!]\, \rho_V\, \kappa_V\, \langle s, i\rangle &= \\ &\mathcal{D}_V[\![D_V]\!]\, \rho_V'\, \kappa_V\, \langle s', i\rangle \text{ where } l = s\, next, \rho_V' = \rho_V[x \mapsto l], \\ &\qquad s' = s[l \mapsto ??, next \mapsto l+1]\end{aligned}$$

No continuations really used here, if desired may be rewritten to a more standard continuation style

Sample semantic clauses

Expressions:

$$\mathcal{E}[\![x]\!]\,\rho_V\,\kappa_E = \lambda\langle s,i\rangle{:}\mathbf{State}.\kappa_E\,n\,\langle s,i\rangle \text{ where } l = \rho_V\,x, n = s\,l$$

this means: ?? if $\rho_V\,x =$?? or $s\,l =$??

$$\mathcal{E}[\![e_1 + e_2]\!]\,\rho_V\,\kappa_E =$$
$$\mathcal{E}[\![e_1]\!]\,\rho_V\,\lambda n_1{:}\mathbf{Int}^{??}.\mathcal{E}[\![e_2]\!]\,\rho_V\,\lambda n_2{:}\mathbf{Int}^{??}.\kappa_E\,(n_1+n_2)$$

Boolean expressions:

$$\mathcal{B}[\![\mathbf{true}]\!]\,\rho_V\,\kappa_B = \kappa_B\,\mathbf{tt}$$

$$\mathcal{B}[\![e_1 \leq e_2]\!]\,\rho_V\,\kappa_B =$$
$$\mathcal{E}[\![e_1]\!]\,\rho_V\,\lambda n_1{:}\mathbf{Int}^{??}.\mathcal{E}[\![e_2]\!]\,\rho_V\,\lambda n_2{:}\mathbf{Int}^{??}.$$
$$\kappa_B\,\mathit{ifte}_{\mathbf{Bool}^{??}}(n_1 \leq n_2, \mathbf{tt}, \mathbf{ff})$$

Keep checking the types!

Statements

$$\mathcal{S}[\![x := e]\!]\,\rho_V\,\rho_P\,\kappa = \mathcal{E}[\![e]\!]\,\rho_V\,\lambda n{:}\mathbf{Int}^{??}.\lambda\langle s,i\rangle{:}\mathbf{State}.\kappa\,\langle s[l \mapsto n], i\rangle \text{ where } l = \rho_V\,x$$

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

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

$$\mathcal{S}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]\,\rho_V\,\rho_P\,\kappa = \\ \quad \mathcal{B}[\![b]\!]\,\rho_V\,\lambda v{:}\mathbf{Bool}^{??}.\mathit{ifte}_{\mathbf{Cont}}(v, \mathcal{S}[\![S_1]\!]\,\rho_V\,\rho_P\,\kappa, \mathcal{S}[\![S_2]\!]\,\rho_V\,\rho_P\,\kappa)$$

$$\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\,\rho_V\,\rho_P\,\kappa = \\ \quad \mathcal{B}[\![b]\!]\,\rho_V\,\lambda v{:}\mathbf{Bool}^{??}.\mathit{ifte}_{\mathbf{Cont}}(v, \mathcal{S}[\![S]\!]\,\rho_V\,\rho_P\,(\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\,\rho_V\,\rho_P\,\kappa), \kappa)$$

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

$$\mathcal{S}[\![\mathbf{read}\ x]\!]\,\rho_V\,\rho_P\,\kappa\,\langle s,i\rangle = \kappa\,\langle s[l \mapsto n], i'\rangle \text{ where } l = \rho_V\,x, \langle n, i'\rangle = i$$

$$\mathcal{S}[\![\mathbf{write}\ e]\!]\,\rho_V\,\rho_P\,\kappa = \mathcal{E}[\![e]\!]\,\rho_V\,\lambda n{:}\mathbf{Int}^{??}.\lambda\langle s,i\rangle{:}\mathbf{State}.\langle n, \kappa\,\langle s,i\rangle\rangle$$

Blocks

$$\mathcal{S}[\![\mathbf{begin}\ D_V\ D_P\ S\ \mathbf{end}]\!]\,\rho_V\,\rho_P\,\kappa =$$
$$\qquad \mathcal{D}_V[\![D_V]\!]\,\rho_V\,\lambda\rho'_V{:}\mathbf{VEnv}.\mathcal{D}_P[\![D_P]\!]\,\rho'_V\,\rho_P\,\lambda\rho'_P{:}\mathbf{PEnv}.\mathcal{S}[\![S]\!]\,\rho'_V\,\rho'_P\,\kappa$$

This got separated, because we will add jumps later...

Warming up:

Exceptions

$$S \in \mathbf{Stmt} ::= \ldots \mid \mathbf{do}\ S_1\ \mathbf{catch}\ exn \Rightarrow S_2 \mid \mathbf{raise}\ exn$$
$$exn \in \mathbf{XName} ::= \ldots$$

- Raising an exception named exn in its corresponding S_1
  - interrupts S_1 (skipping the rest of it), and
  - starts S_2 in the current state.
- Raising exn outside its corresponding S_1 causes an error.
- If exn is not raised within its corresponding S_1, $\mathbf{catch}\ exn \Rightarrow S_2$ is disregarded.

Semantics — sketch

- Another environment:

$$\boxed{\mathbf{XEnv} = \mathbf{XName} \to (\mathbf{Cont} + \{??\})}$$

- The semantic function for statements gets another environment parameter:

$$\boxed{\mathcal{S} \colon \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{XEnv} \to \mathbf{Cont} \to \mathbf{Cont}}_{\mathbf{STMT}}}$$

- Semantic clauses for statements of the "old" forms take the extra parameter and disregard it (passing it "down" if needed), for instance:

$$\mathcal{S}[\![\mathbf{skip}]\!]\, \rho_V\, \rho_P\, \rho_X = id_{\mathbf{Cont}}$$
$$\mathcal{S}[\![S_1; S_2]\!]\, \rho_V\, \rho_P\, \rho_X\, \kappa = \mathcal{S}[\![S_1]\!]\, \rho_V\, \rho_P\, \rho_X\, (\mathcal{S}[\![S_2]\!]\, \rho_V\, \rho_P\, \rho_X\, \kappa)$$

- Semantic clause for new statements:

$$\mathcal{S}[\![\mathbf{do}\ S_1\ \mathbf{catch}\ exn \Rightarrow S_2]\!]\,\rho_V\,\rho_P\,\rho_X\,\kappa = \mathcal{S}[\![S_1]\!]\,\rho_V\,\rho_P\,\rho_X[exn \mapsto \mathcal{S}[\![S_2]\!]\,\rho_V\,\rho_P\,\rho_X\,\kappa]\,\kappa$$

$$\mathcal{S}[\![\mathbf{raise}\ exn]\!]\,\rho_V\,\rho_P\,\rho_X\,\kappa = \rho_X\ exn$$

or perhaps a more explicit version of the clause for raising exception:

$$\mathcal{S}[\![\mathbf{raise}\ exn]\!]\,\rho_V\,\rho_P\,\rho_X\,\kappa = \lambda\langle s,i\rangle{:}\mathbf{State}.\kappa_{exn}\,\langle s,i\rangle \text{ where } \kappa_{exn} = \rho_X\ exn$$

- Semantic clause for programs introduces an initial exception environment with no exceptions declared:

$$\mathcal{P}[\![\mathbf{prog}\ S]\!]\,i = \mathcal{S}[\![S]\!]\,\rho_V^{\emptyset}\,\rho_P^{\emptyset}\,\rho_X^{\emptyset}\,\kappa^{\emptyset}\,\langle s^{\emptyset}, i\rangle$$
$$\text{where } \rho_V^{\emptyset}\,x = ??, \rho_P^{\emptyset}\,p = ??, \rho_X^{\emptyset}\,exn = ??, \kappa^{\emptyset}\,s = \mathbf{eof}, s^{\emptyset}\,next = 0, s^{\emptyset}\,l = ??$$

. . . not done yet

Exceptions in procedures

Static binding of exception names

"raising in its corresponding S_1" $\equiv$ "statically (textually) within S_1"

Then:

$$\mathbf{PROC_0} = \mathbf{Cont} \to \mathbf{Cont}$$

$$\mathcal{D}_P : \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{XEnv} \to \mathbf{Cont_{D_P}} \to \mathbf{Cont}}_{\mathbf{PDECL}}$$

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

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

Exceptions in procedures — expected alternative

Dynamic binding of exception names

"raising in its corresponding S_1" $\equiv$ "dynamically during the execution of S_1"

Then:

$$\mathbf{PROC}_0 = \mathbf{XEnv} \to \mathbf{Cont} \to \mathbf{Cont}$$

$$\mathcal{D}_P : \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont}_{\mathbf{D_P}} \to \mathbf{Cont}}_{\mathbf{PDECL}}$$

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

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

Goto's

$$S \in \mathbf{Stmt} ::= \ldots \mid L{:}S \mid \mathbf{goto}\ L$$
$$L \in \mathbf{LAB} ::= \ldots$$

- Labels are visible (statically) inside the block in which they are declared
- No jumps into blocks are allowed; jumps into other statements are okay

Clarification: programs and procedure bodies are treated as blocks

Semantics — sketch

- Yet another environment:

$$\boxed{\mathbf{LEnv} = \mathbf{LAB} \to (\mathbf{Cont} + \{??\})}$$

- The appropriate semantic functions get another environment parameter:

$$\mathcal{S}:\quad \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{LEnv} \to \mathbf{Cont} \to \mathbf{Cont}}_{\mathbf{STMT}}$$

$$\mathcal{D}_P : \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{LEnv} \to \mathbf{Cont}_{\mathbf{D_P}} \to \mathbf{Cont}}_{\mathbf{PDECL}}$$

- Semantic clauses for declarations and statements of the "old" forms (except blocks) take the extra parameter and disregard it (passing it "down"); semantics for programs introduces label environment with no label declared.
  Change required: programs and procedure bodies should be treated as blocks.

Goto's — sketch of the semantics continues

- We add a declaration-like semantics for statements:

$$\boxed{\mathcal{D}_S : \mathbf{Stmt} \to \mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{LEnv} \to \mathbf{Cont} \to \mathbf{LEnv}}$$

- With a few trivial clauses, like:

$$\boxed{\mathcal{D}_S[\![x := e]\!]\, \rho_V\, \rho_P\, \rho_L\, \kappa = \rho_L}$$

and similarly for **skip**, **call** p, **read** x, **write** e and **goto** L, where no visible labels can be introduced. Perhaps surprisingly, also:

$$\boxed{\mathcal{D}_S[\![\mathbf{begin}\ D_V\ \ D_P\ \ S\ \mathbf{end}]\!]\, \rho_V\, \rho_P\, \rho_L\, \kappa = \rho_L}$$

. . . to be continued

Goto's — sketch of the semantics continues

- And then a few not quite so trivial clauses follow:

$$\mathcal{D}_S[\![S_1; S_2]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa = \\ \quad (\mathcal{D}_S[\![S_1]\!]\,\rho_V\,\rho_P\,\rho_L\,(\mathcal{S}[\![S_2]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa)) \rtimes (\mathcal{D}_S[\![S_2]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa)$$

$$\mathcal{D}_S[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa = \\ \quad (\mathcal{D}_S[\![S_1]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa) \rtimes (\mathcal{D}_S[\![S_2]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa)$$

$$\mathcal{D}_S[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa = \\ \quad \mathcal{D}_S[\![S]\!]\,\rho_V\,\rho_P\,\rho_L\,(\mathcal{S}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa)$$

$$\mathcal{D}_S[\![L{:}S]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa = \\ \quad (\mathcal{D}_S[\![S]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa)[L \mapsto \mathcal{S}[\![S]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa]$$

The only extra thing to explain here is "updating":

$$(\rho_L \rtimes \rho'_L)\,L = \begin{cases} \rho_L\,L & \text{if } \rho'_L\,L = ?? \\ \rho'_L\,L & \text{if } \rho'_L\,L \neq ?? \end{cases}$$

. . . to be continued

Goto's — sketch of the semantics continues

- Finally we need new clauses for the (usual) semantics of labelled statements, of jumps (trivial) and of blocks — rather complicated

. . . and perhaps not quite right?

$$\mathcal{S}[\![L{:}S]\!] = \mathcal{S}[\![S]\!]$$

$$\mathcal{S}[\![\mathbf{goto}\ L]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa = \kappa_L \text{ where } \kappa_L = \rho_L\,L$$

$$\begin{aligned}&\mathcal{S}[\![\mathbf{begin}\ D_V\ D_P\ S\ \mathbf{end}]\!]\,\rho_V\,\rho_P\,\rho_L\,\kappa = \\ &\quad \mathcal{D}_V[\![D_V]\!]\,\rho_V\,\lambda\rho'_V{:}\mathbf{VEnv}.\mathcal{D}_P[\![D_P]\!]\,\rho'_V\,\rho_P\,\rho_L\,\lambda\rho'_P{:}\mathbf{PEnv}. \\ &\qquad \mathcal{S}[\![S]\!]\,\rho'_V\,\rho'_P\,\rho'_L\,\kappa \text{ where } \rho'_L = \mathcal{D}_S[\![S]\!]\,\rho'_V\,\rho'_P\,(\rho_L \rtimes \rho'_L)\,\kappa\end{aligned}$$

Requires a few final (easy!) touches

and change for procedure declarations and programs similar to that for blocks

- make labels within a block visible within procedure declarations in this block
- require that the labels within a block are unique (and check this)
- restrict modification of the label environment for S to the labels introduced by S

"Standard semantics"

- continuations (to built overall results, to handle flow of control, and to simplify notation)
- careful classification of various domains of values (assignable, storable, output-able, closures, etc) with the corresponding semantics of expressions (of various kinds)
- Scott domains and domain equations
- continuous functions only
- ...

... coming in due course ...