

Motivating continuation semantics

Input/output

TINY⁺⁺⁺

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

Semantic domains

$$\text{Stream} = \text{Int} \times \text{Stream} + \{\text{eof}\}$$

$$\text{Input} = \text{Stream}$$

$$\text{Output} = \text{Stream}$$

$$\text{State} = \text{Store} \times \text{Input} \times \text{Output}$$

Actually:

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

Stream includes:

- finite lists, ended by **eof**
- unfinished finite lists
- infinite lists

WARNING: new, different **State**!
forget about the old one!

*Fixed-point definitions work in **Stream***

Semantic functions

$$\begin{array}{l} \mathcal{E}: \mathbf{Exp} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{State} \rightarrow (\mathbf{Int} + \{??\})}_{\mathbf{EXP}} \\ \mathcal{B}: \mathbf{BExp} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{State} \rightarrow (\mathbf{Bool} + \{??\})}_{\mathbf{BEXP}} \end{array}$$

Only one clause to modify here:

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

Semantics of statements

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

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 $\mathcal{D}_V[\![\text{var } x; D_V]\!] \dots = \dots$) and two clauses to add:

$$\begin{aligned} \mathcal{S}[\![\text{read } x]\!] \rho_V \rho_P \langle s, i, o \rangle &= \langle s[l \mapsto n], i', o \rangle \text{ where } l = \rho_V x, \langle n, i' \rangle = i \\ \mathcal{S}[\![\text{write } e]\!] \rho_V \rho_P \langle s, i, o \rangle &= \langle s, i, \langle n, o \rangle \rangle \text{ where } n = \mathcal{E}[\![e]\!] \rho_V \langle s, i, o \rangle \end{aligned}$$

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

Programs

New syntactic domain:

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

with obvious semantic function:

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

given by:

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

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 ... ; ... ; ... **end**

$s^\emptyset \xrightarrow{\mathcal{S}[\dots]} s_i \xrightarrow{\mathcal{S}[\dots]} s_j \xrightarrow{\mathcal{S}[\dots]} s' \rightsquigarrow \text{"overall result"}$

Continuation semantics

begin ... ; ... ; ... **end**

$\kappa' : \rightsquigarrow \xleftarrow{\mathcal{S}[\dots]} \kappa_i : \rightsquigarrow \xleftarrow{\mathcal{S}[\dots]} \kappa_j : \rightsquigarrow \xleftarrow{\mathcal{S}[\dots]} \kappa^\emptyset : \rightsquigarrow$

 "overall result"

1st approximation

Continuations

$$\text{Cont} = \text{State} \rightarrow \text{Res}$$

Now:

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

$$\begin{aligned}\text{State} &= \text{Store} \times \text{Input} \\ \text{Res} &= \text{Output}\end{aligned}$$

$$\mathcal{S}: \text{Stmt} \rightarrow \underbrace{\text{VEnv} \rightarrow \text{PEnv} \rightarrow \text{Cont} \rightarrow \text{Cont}}_{\text{STMT}}$$

That is:

$$\text{STMT} = \text{VEnv} \rightarrow \text{PEnv} \rightarrow \text{Cont} \rightarrow \text{State} \rightarrow \text{Res}$$

Expression and declaration continuations

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

$$\begin{aligned}\mathbf{Cont_E} &= \mathbf{Int} \rightarrow \mathbf{State} \rightarrow \mathbf{Res} & (= \mathbf{Int} \rightarrow \mathbf{Cont}) \\ \mathbf{Cont_B} &= \mathbf{Bool} \rightarrow \mathbf{State} \rightarrow \mathbf{Res} & (= \mathbf{Bool} \rightarrow \mathbf{Cont})\end{aligned}$$

- declarations pass on environments, so

$$\begin{aligned}\mathbf{Cont_{D_V}} &= \mathbf{VEnv} \rightarrow \mathbf{State} \rightarrow \mathbf{Res} & (= \mathbf{VEnv} \rightarrow \mathbf{Cont}) \\ \mathbf{Cont_{D_P}} &= \mathbf{PEnv} \rightarrow \mathbf{State} \rightarrow \mathbf{Res} & (= \mathbf{PEnv} \rightarrow \mathbf{Cont})\end{aligned}$$

$$N \in \mathbf{Num} ::= 0 \mid 1 \mid 2 \mid \dots$$

$$x \in \mathbf{Var} ::= \dots$$

$$p \in \mathbf{IDE} ::= \dots$$

$$e \in \mathbf{Exp} ::= N \mid x \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 - e_2$$

$$b \in \mathbf{BExp} ::= \mathbf{true} \mid \mathbf{false} \mid e_1 \leq e_2 \mid \neg b' \mid b_1 \wedge b_2$$

$$S \in \mathbf{Stmt} ::= x := e \mid \mathbf{skip} \mid S_1; S_2 \mid \mathbf{if} \ b \ \mathbf{then} \ S_1 \ \mathbf{else} \ S_2 \mid \mathbf{while} \ b \ \mathbf{do} \ S' \\ \mid \mathbf{begin} \ D_V \ D_P \ S \ \mathbf{end} \mid \mathbf{call} \ p \mid \mathbf{read} \ x \mid \mathbf{write} \ e$$

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

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

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

Semantic domains

outputs may end also with ??

$\text{Int} = \dots$
 $\text{Int}^? = \text{Int} + \{??\}$
 $\text{Bool} = \dots$
 $\text{Bool}^? = \text{Bool} + \{??\}$
 $\text{Loc} = \dots$
 $\text{Store} = \dots$
 $\text{VEnv} = \dots$

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

expressions may pass ??

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

$\text{Cont} = \text{State} \rightarrow \text{Output}$
 $\text{Cont}_E = \text{Int}^? \rightarrow \text{Cont}$
 $\text{Cont}_B = \text{Bool}^? \rightarrow \text{Cont}$
 $\text{Cont}_{D_V} = \text{VEnv} \rightarrow \text{Cont}$
 $\text{Cont}_{D_P} = \text{PEnv} \rightarrow \text{Cont}$

Semantic functions

$$\mathcal{E}: \text{Exp} \rightarrow \underbrace{\text{VEnv} \rightarrow \text{Cont}_{\text{E}} \rightarrow \text{Cont}}_{\text{EXP}}$$

$$\mathcal{B}: \text{BExp} \rightarrow \underbrace{\text{VEnv} \rightarrow \text{Cont}_{\text{B}} \rightarrow \text{Cont}}_{\text{BEXP}}$$

$$\mathcal{S}: \text{Stmt} \rightarrow \underbrace{\text{VEnv} \rightarrow \text{PEnv} \rightarrow \text{Cont} \rightarrow \text{Cont}}_{\text{STMT}}$$

$$\mathcal{D}_V: \text{VDecl} \rightarrow \underbrace{\text{VEnv} \rightarrow \text{Cont}_{\text{D}_V} \rightarrow \text{Cont}}_{\text{VDECL}}$$

$$\mathcal{D}_P: \text{PDecl} \rightarrow \underbrace{\text{VEnv} \rightarrow \text{PEnv} \rightarrow \text{Cont}_{\text{D}_P} \rightarrow \text{Cont}}_{\text{PDECL}}$$

$$\mathcal{P}: \text{Prog} \rightarrow \underbrace{\text{Input} \rightarrow \text{Output}}_{\text{PROG}}$$

Sample semantic clauses

Programs:

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

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

Declarations:

$$\begin{aligned} \mathcal{D}_P[\varepsilon] \ \rho_V \ \rho_P \ \kappa_P &= \kappa_P \ \rho_P \\ \mathcal{D}_P[\mathbf{proc} \ p \ \mathbf{is} \ (S); D_P] \ \rho_V \ \rho_P &= \\ &\mathcal{D}_P[D_P] \ \rho_V \ \rho_P[p \mapsto P] \text{ where } P = \mathcal{S}[S] \ \rho_V \ \rho_P[p \mapsto P] \\ \mathcal{D}_V[\mathbf{var} \ x; D_V] \ \rho_V \ \kappa_V \ \langle s, i \rangle &= \\ &\mathcal{D}_V[D_V] \ \rho'_V \ \kappa_V \ \langle s', i \rangle \text{ where } l = s \text{ next}, \rho'_V = \rho_V[x \mapsto l], \\ &\quad s' = s[l \mapsto ??, \text{next} \mapsto l + 1] \end{aligned}$$

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

Sample semantic clauses

Expressions:

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

this means: ?? if $\rho_V x = ??$ or $sl = ??$

$$\mathcal{E}[[e_1 + e_2]] \rho_V \kappa_E =$$

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

Boolean expressions:

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

$$\mathcal{B}[[e_1 \leq e_2]] \rho_V \kappa_B =$$

$$\mathcal{E}[[e_1]] \rho_V \lambda n_1 : \mathbf{Int}^{??}. \mathcal{E}[[e_2]] \rho_V \lambda n_2 : \mathbf{Int}^{??}.$$

$$\kappa_B \text{ ifte}_{\mathbf{Bool}^{??}} (n_1 \leq n_2, \mathbf{tt}, \mathbf{ff})$$

Keep checking the types!

Statements

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

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

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

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

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

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

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

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

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

Warming up:

Exceptions

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

- 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 , $\mathbf{catch} \mathit{exn} \Rightarrow S_2$ is disregarded.

Semantics — sketch

- Another environment:

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

- The semantic function for statements gets another environment parameter:

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

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

$$\begin{aligned}\mathcal{S}[\mathbf{skip}] \rho_V \rho_P \rho_X &= id_{\mathbf{Cont}} \\ \mathcal{S}[S_1; S_2] \rho_V \rho_P \rho_X \kappa &= \mathcal{S}[S_1] \rho_V \rho_P \rho_X (\mathcal{S}[S_2] \rho_V \rho_P \rho_X \kappa)\end{aligned}$$

- Semantic clause for new statements:

$$\begin{aligned}\mathcal{S}[\text{do } S_1 \text{ catch } exn \Rightarrow S_2] \rho_V \rho_P \rho_X \kappa &= \\ &\mathcal{S}[S_1] \rho_V \rho_P \rho_X [exn \mapsto \mathcal{S}[S_2] \rho_V \rho_P \rho_X \kappa] \kappa \\ \mathcal{S}[\text{raise } exn] \rho_V \rho_P \rho_X \kappa &= \rho_X exn\end{aligned}$$

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

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

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

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

...not done yet

Exceptions in procedures

Static binding of exception names

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

Then:

$\mathbf{PROC}_0 = \mathbf{Cont} \rightarrow \mathbf{Cont}$

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

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

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

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:

$$\mathbf{PROC}_0 = \mathbf{XEnv} \rightarrow \mathbf{Cont} \rightarrow \mathbf{Cont}$$
$$\mathcal{D}_P : \mathbf{PDecl} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow \mathbf{Cont}_{\mathbf{D}_P}}_{\mathbf{PDECL}} \rightarrow \mathbf{Cont}$$
$$\mathcal{D}_P[\mathbf{proc} \ p \ \mathbf{is} \ (S); D_P] \rho_V \rho_P = \\ \mathcal{D}_P[D_P] \rho_V \rho_P[p \mapsto P] \text{ where } P = \mathcal{S}[S] \rho_V \rho_P[p \mapsto P]$$
$$\mathcal{S}[\mathbf{call} \ p] \rho_V \rho_P \rho_X = P \rho_X \text{ where } P = \rho_P p$$

Goto's

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

- 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:

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

- The appropriate semantic functions get another environment parameter:

$$\begin{array}{l} \mathcal{S}: \mathbf{Stmt} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow \mathbf{LEnv} \rightarrow \mathbf{Cont} \rightarrow \mathbf{Cont}}_{\mathbf{STMT}} \\ \mathcal{D}_P: \mathbf{PDecl} \rightarrow \underbrace{\mathbf{VEnv} \rightarrow \mathbf{PEnv} \rightarrow \mathbf{LEnv} \rightarrow \mathbf{Cont}_{\mathbf{D}_P} \rightarrow \mathbf{Cont}}_{\mathbf{PDECL}} \end{array}$$

- 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:

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

- With a few trivial clauses, like:

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

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

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



Goto's — sketch of the semantics continues

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

$$\begin{aligned}\mathcal{D}_S[S_1; S_2] \rho_V \rho_P \rho_L \kappa &= \\ &(\mathcal{D}_S[S_1] \rho_V \rho_P \rho_L (\mathcal{S}[S_2] \rho_V \rho_P \rho_L \kappa)) \rtimes (\mathcal{D}_S[S_2] \rho_V \rho_P \rho_L \kappa) \\ \mathcal{D}_S[\text{if } b \text{ then } S_1 \text{ else } S_2] \rho_V \rho_P \rho_L \kappa &= \\ &(\mathcal{D}_S[S_1] \rho_V \rho_P \rho_L \kappa) \rtimes (\mathcal{D}_S[S_2] \rho_V \rho_P \rho_L \kappa) \\ \mathcal{D}_S[\text{while } b \text{ do } S] \rho_V \rho_P \rho_L \kappa &= \\ &\mathcal{D}_S[S] \rho_V \rho_P \rho_L (\mathcal{S}[\text{while } b \text{ do } S] \rho_V \rho_P \rho_L \kappa) \\ \mathcal{D}_S[L:S] \rho_V \rho_P \rho_L \kappa &= \\ &(\mathcal{D}_S[S] \rho_V \rho_P \rho_L \kappa)[L \mapsto \mathcal{S}[S] \rho_V \rho_P \rho_L \kappa]\end{aligned}$$

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

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

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

$$\mathcal{S}[[\text{begin } D_V \ D_P \ S \ \text{end}]] \rho_V \rho_P \rho_L \kappa =$$

$$\mathcal{D}_V[[D_V]] \rho_V \lambda \rho'_V : \mathbf{VEnv}. \mathcal{D}_P[[D_P]] \rho'_V \rho_P \rho_L \lambda \rho'_P : \mathbf{PEnv}.$$

$$\mathcal{S}[[S]] \rho'_V \rho'_P \rho'_L \kappa \text{ where } \rho'_L = \mathcal{D}_S[[S]] \rho'_V \rho'_P (\rho_L \rtimes \rho'_L) \kappa$$

... and perhaps not quite right?

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 ...