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`

Strona tego wykładu:

`http://www.mimuw.edu.pl/~klin/sem19-20.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

This course: http://www.mimuw.edu.pl/~klin/sem19-20.html

Direct semantics

$$\textbf{begin} \;\ldots\; ; \;\ldots\; ; \;\ldots\; \textbf{end}$$

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

Continuation semantics

$$\textbf{begin} \;\ldots\; ; \;\ldots\; ; \;\ldots\; \textbf{end}$$

$$\kappa' : \rightsquigarrow \text{"overall result"}$$

$$\xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa_i : \rightsquigarrow \text{"overall result"}$$

$$\xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa_j : \rightsquigarrow \text{"overall result"}$$

$$\xleftarrow{\mathcal{S}[\![\ldots]\!]} \kappa^{\emptyset} : \rightsquigarrow \text{"overall result"}$$

Continuations

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

Now:

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

- states do not include outputs
- answers are outputs (or errors)
- these are continuations for statements; semantics for statements is given by:

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

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

Expression and declaration continuations

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

$$\boxed{\begin{aligned} \mathbf{Cont_E} &= \mathbf{Int} \to \mathbf{State} \to \mathbf{Ans} \\ \mathbf{Cont_B} &= \mathbf{Bool} \to \mathbf{State} \to \mathbf{Ans} \end{aligned}}$$

  - declarations pass on environments, so

$$\boxed{\begin{aligned} \mathbf{Cont_{D_V}} &= \mathbf{VEnv} \to \mathbf{State} \to \mathbf{Ans} \\ \mathbf{Cont_{D_P}} &= \mathbf{PEnv} \to \mathbf{State} \to \mathbf{Ans} \end{aligned}}$$

TINY^{+++}

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

Semantic domains

$$\begin{aligned}
\mathbf{Int} &= \ldots \\
\mathbf{Bool} &= \ldots \\
\mathbf{Loc} &= \ldots \\
\mathbf{Store} &= \ldots \\
\mathbf{VEnv} &= \ldots
\end{aligned}$$

$$\begin{aligned}
\mathbf{Input} &= \mathbf{Int} \times \mathbf{Input} + \{\mathbf{eof}\} \\
\mathbf{State} &= \mathbf{Store} \times \mathbf{Input} \\
\mathbf{Output} &= \mathbf{Int} \times \mathbf{Output} + \{\mathbf{eof}, ??\}
\end{aligned}$$

$$\begin{aligned}
\mathbf{Cont} &= \mathbf{State} \to \mathbf{Output} \\
\mathbf{Cont_E} &= \mathbf{Int} \to \mathbf{Cont} \\
\mathbf{Cont_B} &= \mathbf{Bool} \to \mathbf{Cont} \\
\mathbf{Cont_{D_V}} &= \mathbf{VEnv} \to \mathbf{Cont} \\
\mathbf{Cont_{D_P}} &= \mathbf{PEnv} \to \mathbf{Cont}
\end{aligned}$$

$$\begin{aligned}
\mathbf{PROC}_0 &= \mathbf{Cont} \to \mathbf{Cont} \\
\mathbf{PROC}_1^{\mathsf{vr}} &= \mathbf{Loc} \to \mathbf{PROC}_0 \\
\mathbf{PEnv} &= \\
&\quad \mathbf{IDE} \to (\mathbf{PROC}_0 + \mathbf{PROC}_1^{\mathsf{vr}} + \{??\})
\end{aligned}$$

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:

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

Continuations not really used here, just passed around

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 $s\,l = ??$

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

check the types!

Boolean expressions:

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

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

Back to declarations

Recall:

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

What would happen if variable declarations included initializing expressions?

$$\mathcal{D}_V[\![\mathbf{var}\ x = e; D_V]\!]\ \rho_V\ \kappa_V\ \langle s, i\rangle = \mathcal{E}[\![e]\!]\ \rho_V\ (\lambda n{:}\mathbf{Int}.\mathcal{D}_V[\![D_V]\!]\ \rho'_V\ \kappa_V\ \langle s', i\rangle) \text{ where } l = s\ next, \rho'_V = \rho_V[x \mapsto l], s' = s[l \mapsto n, next \mapsto l+1]$$

Statements

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

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

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

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

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

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

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

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

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

Blocks

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

This got separated, because we will want to add jumps...

Abrupt termination

Let us forget input/output for now, fall back to the language $\textsc{Tiny}^{++}$with (paremeterless) procedures.

Extend it with:

$$\boxed{S \in \mathbf{Stmt} ::= \ldots \mid \mathbf{abort}}$$

The killer application of continuations is non-local control flow.

This is the simplest example.

Semantic domains

$$
\begin{aligned}
\mathbf{Ans} &= \mathbf{Store} \\
\mathbf{Cont} &= \mathbf{Store} \to \mathbf{Ans} \\
\mathbf{Cont_E} &= \mathbf{Int} \to \mathbf{Ans} \\
\mathbf{Cont_B} &= \mathbf{Bool} \to \mathbf{Ans} \\
\mathbf{Cont_{D_V}} &= \mathbf{VEnv} \to \mathbf{Cont} \\
\mathbf{Cont_{D_P}} &= \mathbf{PEnv} \to \mathbf{Ans} \\
\mathbf{PROC} &= \mathbf{Cont} \to \mathbf{Cont} \\
\mathbf{PEnv} &= \mathbf{IDE} \to (\mathbf{PROC} + \{??\})
\end{aligned}
$$

Most continuation types got simplified, since expressions or procedure declarations do not produce new $\mathbf{Store}$'s. We could have done that previously, too.

Semantic functions

As before:

$$
\begin{aligned}
\mathcal{E} :\quad & \mathbf{Exp} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont}_{\mathbf{E}} \to \mathbf{Cont}}_{\mathbf{EXP}} \\
\mathcal{B} :\quad & \mathbf{BExp} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont}_{\mathbf{B}} \to \mathbf{Cont}}_{\mathbf{BEXP}} \\
\mathcal{S} :\quad & \mathbf{Stmt} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont} \to \mathbf{Cont}}_{\mathbf{STMT}} \\
\mathcal{D}_V :\quad & \mathbf{VDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{Cont}_{\mathbf{D_V}} \to \mathbf{Cont}}_{\mathbf{VDECL}} \\
\mathcal{D}_P :\quad & \mathbf{PDecl} \to \underbrace{\mathbf{VEnv} \to \mathbf{PEnv} \to \mathbf{Cont}_{\mathbf{D_P}} \to \mathbf{Cont}}_{\mathbf{PDECL}}
\end{aligned}
$$

Sample semantic clauses

Roughly as before. A few get simpler because of simpler continuation types, e.g.:

$$\boxed{\mathcal{E}[\![x]\!]\ \rho_V\ \kappa_E\ s = \kappa_E\ n \text{ where } l = \rho_V\ x, n = s\ l}$$

But a few get more complicated because the simpler types require more explicit state passing, e.g.:

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

One new clause:

$$\boxed{\mathcal{S}[\![\mathbf{abort}]\!]\ \rho_V\ \rho_P\ \kappa = id_{\mathbf{Store}}}$$

Compare it to:

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

Exceptions

Exception throwing is a more fancy kind of abrupt termination, where only part of a program gets terminated.

We will throw and catch named exceptions without parameters.

$$
\boxed{\begin{array}{l} S \in \mathbf{Stmt} ::= \ldots \mid \mathbf{try}\ S_1\ \mathbf{catch}(\chi)\ S_2 \mid \mathbf{throw}\ \chi \\ \chi \in \mathbf{EXN} ::= \ldots \end{array}}
$$

- A thrown exception may erase a part of the procedure-call stack, but it does not erase changes to the store.

Semantic domains

- A new kind of environment:

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

- The appropriate semantic functions get another environment parameter:

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

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

Semantic clauses

- Semantic clauses for declarations and statements of the "old" forms take the extra environment parameter and disregard it (passing it "down").
- New clauses:

$$\mathcal{S}[\![\mathbf{try}\ S_1\ \mathbf{catch}(\chi)\ S_2]\!]\ \rho_V\ \rho_P\ \rho_X\ \kappa =$$
$$\qquad \mathcal{S}[\![S_1]\!]\ \rho_V\ \rho_P\ \rho_X[\chi \mapsto \kappa']\ \kappa \text{ where } \kappa' = \mathcal{S}[\![S_2]\!]\ \rho_V\ \rho_P\ \rho_X\ \kappa$$

$$\mathcal{S}[\![\mathbf{throw}\ \chi]\!]\ \rho_V\ \rho_P\ \rho_X\ \kappa = \rho_X\ \chi$$