Denotational Semantics of simPL

YSC3208: Programming Language Design & Implementation

Răzvan Voicu

Week 5, Feb 6 - 10, 2017

A Critique of Current Approach

- Contraction relies on substitution; mathematically rather complex.
- Primitive operations that are not total functions, such as division, make the evaluation process get stuck. We want a more explicit way of handling runtime errors.
- simPL contains many “unreasonable” programs, which complicates the definition of a dynamic semantics

A Glimpse of Denotational Semantics

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{+[E_1, E_2] \rightarrowtail v_1 + v_2}$$

- syntactic domains E
- semantic domains v
- semantic functions $E \rightarrowtail v$

Sublanguages of simPL

- simPL0; integer and boolean expressions
- simPL1; add `let` and `if`
- simPL2; add division
- simPL3; add functions
- simPL4; add recursive functions

Syntactic Domain of simPL0

$$\frac{}{n} \qquad \frac{}{\texttt{true}} \qquad \frac{}{\texttt{false}}$$

$$\frac{E_1 \quad E_2}{p[E_1, E_2]} \; p \in \{\texttt{|},\texttt{\&},\texttt{+},\texttt{-},\texttt{*},\texttt{=},\texttt{>},\texttt{<}\} \qquad \frac{E}{p[E]} \; p \in \{\texttt{\textbackslash},\texttt{~}\}$$

Semantic Domains

Domain name	Definition	Explanation
Bool	$\{$*true*, *false*$\}$	ring of booleans
Int	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$	ring of integers
EV	**Bool** $+$ **Int**	expressible values

Expressible values **EV** are values that are the result of evaluating an expression.

Semantic Function

The semantic function

$$\cdot \rightarrowtail \cdot : \textbf{simPL0} \rightarrow \textbf{EV}$$

expresses the meaning of elements of **simPL0**, by defining the value of each element.

Semantic Rules

$$\frac{}{\texttt{true} \rightarrowtail \mathit{true}} \qquad \frac{}{\texttt{false} \rightarrowtail \mathit{false}} \qquad \frac{n \rightarrowtail_{\mathbf{N}} i}{n \rightarrowtail i}$$

The function $\rightarrowtail_{\mathbf{N}}$ transforms the simPL0 integer syntax into an element of **Int**.

Semantic Rules (cont'd)

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1\texttt{+}E_2 \rightarrowtail v_1 + v_2}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1\texttt{-}E_2 \rightarrowtail v_1 - v_2}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1\texttt{*}E_2 \rightarrowtail v_1 \cdot v_2}$$

Semantic Rules (cont'd)

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 \& E_2 \rightarrowtail v_1 \wedge v_2}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 \mid E_2 \rightarrowtail v_1 \vee v_2}$$

$$\frac{E \rightarrowtail v}{\backslash E \rightarrowtail \neg v}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1 {=} E_2 \rightarrowtail v_1 \equiv v_2}$$

Semantic Rules (cont'd)

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1\texttt{>}E_2 \rightarrowtail v_1 > v_2}$$

$$\frac{E_1 \rightarrowtail v_1 \qquad E_2 \rightarrowtail v_2}{E_1\texttt{<}E_2 \rightarrowtail v_1 < v_2}$$

Example

$1\ \texttt{+}\ 2\ \texttt{>}\ 3 \rightarrowtail \textit{false}$ holds because $1\ \texttt{+}\ 2 \rightarrowtail 3$ and $3 > 3$ is *false*.

simPL1

Add the following to simPL0:

- conditionals,
- identifiers,
- let

Syntactic Domain of simPL1

$$\frac{E \qquad E_1 \qquad E_2}{\texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end}}$$

$$\frac{}{x}$$

$$\frac{E \quad E_1 \quad \ldots \quad E_n}{\texttt{let } x_1 = E_1 \ \cdots x_n = E_n \texttt{ in } E \texttt{ end}}$$

Motivation: Semantic Domains of simPL1

We need to introduce environments that allow us to keep track of the binding of identifiers. These environments map identifiers to *denotable values*.

Semantic Domains of simPL1

Semantic domain	Definition	Explanation
Bool	$\{$*true*, *false*$\}$	ring of booleans
Int	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$	ring of integers
EV	**Bool** $+$ **Int**	expressible values
DV	**Bool** $+$ **Int**	denotable values
Id	alphanumeric string	identifiers
Env	**Id** $\rightsquigarrow$ **DV**	environments

Operations on Environments

We introduce an operation $\Delta[x \leftarrow v]$,
which denotes an environment that works like Δ,
except that $\Delta[x \leftarrow v](x) = v$

Semantic Functions for simPL1

Define $\cdot \rightarrowtail \cdot$ using auxiliary semantic function $\cdot \Vdash \cdot \rightarrowtail \cdot$ that gets an environment as additional argument.

$$\cdot \rightarrowtail \cdot : \textbf{simPL1} \rightarrow \textbf{EV}$$

$$\frac{\emptyset \Vdash E \rightarrowtail v}{E \rightarrowtail v}$$

Auxiliary Semantic Function for simPL1

$$\cdot \Vdash \cdot \rightarrowtail \cdot : \textsf{\textbf{Env}} * \mathrm{simPL1} \to \textsf{\textbf{EV}}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{}{\Delta \Vdash \texttt{true} \rightarrowtail true}$$

$$\frac{}{\Delta \Vdash \texttt{false} \rightarrowtail false}$$

$$\frac{n \rightarrowtail_{\mathbf{N}} i}{\Delta \Vdash n \rightarrowtail i}$$

$$\frac{}{\Delta \Vdash x \rightarrowtail \Delta(x)}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1\texttt{+}E_2 \rightarrowtail v_1 + v_2}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1\texttt{-}E_2 \rightarrowtail v_1 - v_2}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1\texttt{*}E_2 \rightarrowtail v_1 \cdot v_2}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 \& E_2 \rightarrowtail v_1 \wedge v_2}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 \mid E_2 \rightarrowtail v_1 \vee v_2}$$

$$\frac{\Delta \Vdash E \rightarrowtail v}{\Delta \Vdash \backslash E \rightarrowtail \neg v} \qquad \frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 {=} E_2 \rightarrowtail v_1 \equiv v_2}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 \texttt{>} E_2 \rightarrowtail v_1 > v_2}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 \texttt{<} E_2 \rightarrowtail v_1 < v_2}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta \Vdash E \rightarrowtail \mathit{true} \quad \Delta \Vdash E_1 \rightarrowtail v_1}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail v_1}$$

$$\frac{\Delta \Vdash E \rightarrowtail \mathit{false} \quad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail v_2}$$

Auxiliary Semantic Function for simPL1 (cont'd)

$$\frac{\Delta[x_1 \leftarrow v_1]\cdots[x_n \leftarrow v_n] \Vdash E \rightarrowtail v \quad \Delta \Vdash E_1 \rightarrowtail v_1 \quad \cdots \quad \Delta \Vdash E_n \rightarrowtail v_n}{\Delta \Vdash \texttt{let } x_1 \texttt{ = } E_1 \cdots x_n \texttt{ = } E_n \texttt{ in } E \texttt{ end} \rightarrowtail v}$$

Example

$$\dfrac{\dfrac{\dfrac{\cdots}{\emptyset[\texttt{AboutPi} \leftarrow 3] \Vdash \texttt{AboutPi + 2} \rightarrowtail 5}}{\emptyset \Vdash \texttt{let AboutPi = 3 in AboutPi + 2 end} \rightarrowtail 5}}{\texttt{let AboutPi = 3 in AboutPi + 2 end} \rightarrowtail 5}$$

Syntactic Domain of simPL2

The language simPL2 adds division to simPL1.

$$\frac{E_1 \quad E_2}{E_1 / E_2}$$

The difficulty lies in the fact that division on integers is a partial function, not being defined for 0 as second argument.

Semantic Domains of simPL2

Domain name	Definition	Explanation
Bool	$\{\mathit{true}, \mathit{false}\}$	ring of booleans
Int	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$	ring of integers
EV	$\mathbf{Bool} + \mathbf{Int} + \{\bot\}$	expressible values
DV	$\mathbf{Bool} + \mathbf{Int}$	denotable values
Id	alphanumeric string	identifiers
Env	$\mathbf{Id} \rightsquigarrow \mathbf{DV}$	environments

Modify Auxiliary Semantic Function

The semantic function $\cdot \Vdash \cdot \rightarrowtail \cdot$ is modified to take the occurrence of the error value $\perp$ into account.

Auxiliary Semantic Function (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail \perp}{\Delta \Vdash E_1\texttt{+}E_2 \rightarrowtail \perp} \qquad \frac{\Delta \Vdash E_2 \rightarrowtail \perp}{\Delta \Vdash E_1\texttt{+}E_2 \rightarrowtail \perp}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1\texttt{+}E_2 \rightarrowtail v_1 + v_2} \text{ if } v_1, v_2 \neq \perp$$

Auxiliary Semantic Function (cont'd)

$$\frac{\Delta \Vdash E_1 \rightarrowtail \bot}{\Delta \Vdash E_1 / E_2 \rightarrowtail \bot} \qquad \frac{\Delta \Vdash E_2 \rightarrowtail \bot}{\Delta \Vdash E_1 / E_2 \rightarrowtail \bot}$$

$$\frac{\Delta \Vdash E_2 \rightarrowtail 0}{\Delta \Vdash E_1 / E_2 \rightarrowtail \bot}$$

$$\frac{\Delta \Vdash E_1 \rightarrowtail v_1 \qquad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash E_1 / E_2 \rightarrowtail v_1 / v_2} \quad \text{if } v_1, v_2 \neq \bot \text{ and } v_2 \neq 0$$

Auxiliary Semantic Function (cont'd)

$$\frac{\Delta \Vdash E \rightarrowtail \bot}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail \bot}$$

$$\frac{\Delta \Vdash E \rightarrowtail \mathit{true} \quad \Delta \Vdash E_1 \rightarrowtail v_1}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail v_1}$$

$$\frac{\Delta \Vdash E \rightarrowtail \mathit{false} \quad \Delta \Vdash E_2 \rightarrowtail v_2}{\Delta \Vdash \texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end} \rightarrowtail v_2}$$

Auxiliary Semantic Function (cont'd)

$$\frac{\Delta \Vdash E_i \rightarrowtail \perp}{\Delta \Vdash \texttt{let } x_1 \texttt{ = } E_1 \cdots x_n \texttt{ = } E_n \texttt{ in } E \texttt{ end} \rightarrowtail \perp} \quad \text{for } i, 1 \leq i \leq n$$

$$\frac{\Delta[x_1 \leftarrow v_1] \cdots [x_n \leftarrow v_n] \Vdash E \rightarrowtail v \quad \Delta \Vdash E_1 \rightarrowtail v_1 \cdots \Delta \Vdash E_n \rightarrowtail v_n}{\Delta \Vdash \texttt{let } x_1 \texttt{ = } E_1 \cdots x_n \texttt{ = } E_n \texttt{ in } E \texttt{ end} \rightarrowtail v}$$

Example

For any environment Δ,
$\Delta \Vdash$5+(3/0) $\rightarrowtail \perp$, since
$\Delta \Vdash$3/0 $\rightarrowtail \perp$.

Syntactic Domain of simPL3

Add (non-recursive) function definition and application.

$$\frac{E}{\texttt{fun}\ \{\ t\ \}\ x_1 \ldots x_n \texttt{->} E\ \texttt{end}}$$

$$\frac{E \quad E_1 \quad \cdots \quad E_n}{\texttt{(}E\ E_1\ \ldots\ E_n\texttt{)}}$$

Semantic Domains of simPL3

Domain name	Definition	Explanation
Bool	$\{$*true*, *false*$\}$	ring of booleans
Int	$\{\ldots, -2, -1, 0, 1, 2, \ldots\}$	ring of integers
EV	$\mathbf{Bool} + \mathbf{Int} + \{\perp\} + \mathbf{Fun}$	expressible values
DV	$\mathbf{Bool} + \mathbf{Int} + \mathbf{Fun}$	denotable values
Id	alphanumeric string	identifiers
Env	$\mathbf{Id} \rightsquigarrow \mathbf{DV}$	environments
Fun	$\mathbf{DV} * \cdots * \mathbf{DV} \rightsquigarrow \mathbf{EV}$	function values

Auxiliary Semantic Function (cont'd)

$$\frac{}{\Delta \Vdash \texttt{fun}\ \{\ t\ \}\ x_1 \ldots x_n \texttt{->} E\ \texttt{end} \rightarrowtail f}$$

f is function such that
$f(y_1, \ldots, y_n) = v$, where
$\Delta[x_1 \leftarrow y_1] \cdots [x_n \leftarrow y_n] \Vdash E \rightarrowtail v$

Such a function can be syntactically denoted by closure of the form:

$$\texttt{CLS}(\Delta, [x_1 \cdots x_n], E)$$

Evaluation of Function Application

$$\frac{\Delta \Vdash E \rightarrowtail f \quad \Delta \Vdash E_1 \rightarrowtail v_1 \quad \cdots \quad \Delta \Vdash E_n \rightarrowtail v_n}{\Delta \Vdash (E\ E_1\ \ldots\ E_n) \rightarrowtail f(v_1, \ldots, v_n)}$$

Syntactic Domain of simPL4

$$\frac{E}{\texttt{recfun}\ f\ \{\ t\ \}\ x_1 \ldots x_n \texttt{->} E\ \texttt{end}}$$

Discussion simPL4

We would like to add the following rule to our definition of $\rightarrowtail$.

$$\frac{}{\Delta \Vdash \texttt{recfun}\ g\ \{\ t\ \}\ x_1 \ldots x_n \texttt{->} E\ \texttt{end} \rightarrowtail f}$$

f is function such that
$f(y_1, \ldots, y_n) = v$, where
$\Delta[x_1 \leftarrow y_1] \cdots [x_n \leftarrow y_n]\ [g \leftarrow f] \Vdash E \rightarrowtail v$

This can be syntactically denoted by circular closure :

$$v = \mathrm{CLS}(\Delta[g{\leftarrow}v], [x_1 \cdots x_n], E)$$

Problem

The symbol f occurs on the right hand side of the definition of f. How do we know that such a function f exists? Some expressions have unique solutions for f, others have multiple solutions. Example:

```
(recfun f {int -> int} x -> (f x) end 0)
```

Theory of fixpoints ...

Preprocessor for simPL

- type checking/inference for simPL
- simPL ====> simPL core

  This pre-processing can perform the following tasks:

  - Each Let construct is converted to a function application.
  - Each partial applications is converted to full application.

Input Language simPL

AST of simPL in OCaml

```
type sPL_expr =
   | BoolConst of bool
   | IntConst of int
   | Var of id
   | UnaryPrimApp of op_id * sPL_expr
   | BinaryPrimApp of op_id * sPL_expr * sPL_expr
   | Cond of sPL_expr * sPL_expr * sPL_expr
   | Func of sPL_type * (id list) * sPL_expr
   | RecFunc of sPL_type * id * (id list) * sPL_expr
   | Appln of sPL_expr * sPL_type option * (sPL_expr list)
   | Let of ((sPL_type * id * sPL_expr) list)
            * sPL_type * sPL_expr
```

Core Language simPL

Core Language of simPL in OCaml

```
type sPL_expr =
  | BoolConst of bool
  | IntConst of int
  | Var of id
  | UnaryPrimApp of op_id * sPL_expr
  | BinaryPrimApp of op_id * sPL_expr * sPL_expr
  | Cond of sPL_expr * sPL_expr * sPL_expr
  | Func of sPL_type * (id list) * sPL_expr
  | RecFunc of sPL_type * id * (id list) * sPL_expr
  | Appln of sPL_expr * sPL_type option * (sPL_expr list)
```

Interpreters for simPL

- By substitution:

```
evaluate (e:sPL_expr): sPL_expr =
```

- By type environment (similar to denotational semantics):

```
type env_val = (sPL_value ref) Environ.et
evaluate (env:env_val) (e:sPL_expr): sPL_value =
```

Values of simPL

Values of simPL (including function-valued closure)

```
type sPL_value =
  | BOT (* denotes an error *)
  | VInt of int
  | VBool of bool
  | CLS of ((sPL_value ref) Environ.et)
              * (id list) * sPL_expr
```