03—simPL: Dyn + Stat Semantics & Implementation

CS4215: Programming Language Implementation

Martin Henz

January 27, 2012

Generated on Friday 27 January, 2012, 11:56

Motivation for simPL

- built-in conditionals,
- function definition and application, and
- recursive function definitions.

simPL allows us to study typing, realistic interpretation and virtual machines in detail.

Types

$$\frac{}{\texttt{int}} \qquad \frac{}{\texttt{bool}} \qquad \frac{t_1 \quad \cdots \quad t_n \quad t}{t_1 \texttt{*} \cdots \texttt{*} t_n \texttt{ -> } t}$$

Expressions

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

$$\frac{E}{p_1[E]} \qquad \frac{E_1 \quad E_2}{p_2[E_1, E_2]}$$

Expressions (cont'd)

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

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

Expressions (cont'd)

$$\frac{E}{\texttt{fun } \{t_1 \texttt{*} \cdots \texttt{*} t_n \texttt{ -> } t\}\ x_1 \cdots x_n \texttt{ -> } E \texttt{ end}}$$

if $t_1, \ldots, t_n$ and t are types, $n \geq 1$. The variables $x_1, \ldots, x_n$ must be pairwise distinct.

$$\frac{E}{\texttt{recfun } f\ \{t_1 \texttt{*} \cdots \texttt{*} t_n \texttt{ -> } t\}\ x_1 \cdots x_n \texttt{ -> } E \texttt{ end}}$$

if $t_1, \ldots, t_n$ and t are types, $n \geq 1$. The variables $f, x_1, \cdots, x_n$ must be pairwise distinct.

Syntactic Conventions

- Parentheses
- Infix and prefix notation for operators

  `x + x * y > 10 - x`

  stands for

  `>[+[x,*[x,y]],-[10,x]]`
- `->` is right-associative, so that the type

  `int -> int -> int`

  is equivalent to

  `int -> (int -> int)`

Example

```
fun {int -> int -> int} x ->
   fun {int -> int} y -> x + y end
end
```

takes an integer `x` as argument and returns a function, whereas the function

```
fun {(int -> int) -> int} f -> (f 2) end
```

takes a function `f` as argument and returns an integer.

Let Expressions

$$\texttt{let}\ \{t_1\}\ x_1 = E_1 \cdots \{t_n\}\ x_n = E_n\ \texttt{in}\ \{t\}\ E\ \texttt{end}$$

stands for

$$\texttt{(fun}\ \{t_1\texttt{*}\cdots\texttt{*}t_n\ \texttt{->}\ t\}\ x_1 \cdots x_n\ \texttt{->}\ E\ \texttt{end}\ E_1 \cdots E_n\texttt{)}$$

Example

```
let {int} AboutPi = 3
    {int -> int} Square =
         fun {int -> int} x -> x * x end
in  {int} 4 * AboutPi * (Square 6371)
end
```

Example (continued)

```
(fun {int * (int -> int) -> int}
    AboutPi Square
    ->
    4 * AboutPi * (Square 6371)
 end
 3
 fun {int -> int} x -> x * x end)
```

Power Function

```
recfun power {int * int -> int}
   x y ->
   if y = 0
   then 1
   else x * (power x y - 1)
   end
end
```

Values

In simPL, functions are values, although their bodies may not be values.

```
fun {int -> int} x -> 3 * 4 end
```

Value

A simPL value is:

- an integer, or
- a boolean value, or
- a function definition `fun` · · · `->` · · · `end`, or
- a recursive function definition `recfun` · · · `->` · · · `end`).

Contraction

$$\frac{}{p_1[v_1] >_{\text{simPL}} v}\,[\text{OpVals}] \qquad \frac{}{p_2[v_1, v_2] >_{\text{simPL}} v}\,[\text{OpVals}]$$

$$\frac{}{\texttt{if true then } E_1 \texttt{ else } E_2 \texttt{ end} \quad >_{\text{simPL}} \quad E_1}\,[\text{IfTrue}]$$

Contraction (cont'd)

$$\frac{}{\texttt{if false then } E_1 \texttt{ else } E_2 \texttt{ end} \quad >_{\text{simPL}} \quad E_2}\ [\text{IfFalse}]$$

Contraction of Function Application

- Free variables
- Substitution
- Contraction of function application

Free Variables

```
(fun {int -> int} x -> 4 * (square x) end 3)
```

Goal:

$$\bowtie : \text{simPL} \times 2^{V}$$

Example

$\texttt{4 * (square x)} \bowtie \{\texttt{square,x}\}$

Read: "the set of free variables of the expression `4 * (square x)` is {square,x}.

Definition of $\bowtie$

$$\frac{}{x \bowtie \{x\}} \qquad \frac{}{n \bowtie \emptyset} \qquad \frac{}{\texttt{true} \bowtie \emptyset} \qquad \frac{}{\texttt{false} \bowtie \emptyset}$$

Definition of $\bowtie$ (cont'd)

$$\frac{E \bowtie X}{p_1[E] \bowtie X}$$

$$\frac{E_1 \bowtie X_1 \qquad E_2 \bowtie X_2}{p_2[E_1, E_2] \bowtie X_1 \cup X_2}$$

Definition of $\bowtie$ (cont'd)

$$\frac{E_1 \bowtie X_1 \qquad E_2 \bowtie X_2 \qquad E_3 \bowtie X_3}{\texttt{if}\ E_1\ \texttt{then}\ E_2\ \texttt{else}\ E_3\ \texttt{end} \bowtie X_1 \cup X_2 \cup X_3}$$

Definition of $\bowtie$ (cont'd)

$$\frac{E \bowtie X}{\texttt{fun}\ \{\ \cdot\ \}\ x_1 \cdots x_n\ \texttt{->}\ E\ \texttt{end} \bowtie X - \{x_1, \ldots, x_n\}}$$

Definition of $\bowtie$ (cont'd)

$$\frac{E \bowtie X}{\texttt{recfun}\ \{\,\cdot\,\}\ f\ x_1 \cdots x_n\ \texttt{->}\ E\ \texttt{end} \bowtie X - \{f, x_1, \ldots, x_n\}}$$

Substitution

Goal: For function application, replace all free occurrences of the formal parameters in the function body by the actual arguments.

```
(fun {int -> int} x -> x * x end 4)
```

Replace every free occurrence of `x` in `x * x` by the actual parameter 4, resulting in

```
4 * 4
```

Substitution

Define the substitution relation

$$\cdot[\cdot \leftarrow \cdot]\cdot : \mathrm{simPL} \times V \times \mathrm{simPL} \times \mathrm{simPL}$$

such that $\mathtt{x} \;*\; \mathtt{x}[\mathtt{x} \leftarrow 4]4 \;*\; 4$ holds.

Definition of Substitution

$$\frac{}{v[v \leftarrow E_1]E_1} \text{for any variable } v$$

$$\frac{}{x[v \leftarrow E_1]x} \text{for any variable } x \neq v$$

Definition of Substitution (cont'd)

$$\frac{E_1[v \leftarrow E]E_1' \qquad E_2[v \leftarrow E]E_2'}{(E_1\ E_2)[v \leftarrow E](E_1'\ E_2')}$$

Definition of Substitution (cont'd)

$$\frac{}{\texttt{fun}\ \{\cdot\}\ v\texttt{->}E\ \texttt{end}\ [v \leftarrow E_1]\texttt{fun}\ \{\cdot\}\ v\ \texttt{->}\ E\ \texttt{end}}$$

$$\frac{E\ [v \leftarrow E_1]E' \qquad x \neq v \qquad E_1 \bowtie X_1 \qquad x \notin X_1}{\texttt{fun}\ \{\cdot\}\ x\texttt{->}E\ \texttt{end}\ [v \leftarrow E_1]\ \texttt{fun}\ \{\cdot\}\ x\ \texttt{->}\ E'\ \texttt{end}}$$

Definition of Substitution (cont'd)

$$\frac{E_1 \bowtie X_1 \qquad x \in X_1 \qquad E \bowtie X \qquad E[x \leftarrow z]E' \qquad E'[v \leftarrow E_1]E'' \qquad x \neq v}{\texttt{fun}\ \{\cdot\}\ x\texttt{->}E\ \texttt{end}\ [v \leftarrow E_1]\ \texttt{fun}\ \{\cdot\}\ z\ \texttt{->}\ E''\ \texttt{end}}$$

where we choose z such that $z \notin X_1 \cup X$.

Examples

- `fun {int -> int} factor -> factor * 4 * y end`
  `[factor←  x + 1]`
  `fun {int -> int} factor -> factor * 4 * y end`
- `fun {int -> int} factor -> factor * 4 * y end`
  `[y←  x + 1]`
  `fun {int -> int} factor -> factor * 4 * (x + 1) end`

Examples

- `fun {int -> int} factor -> factor * 4 * y end`
  [y← `factor + 1`]
  ```
  fun {int -> int} newfactor ->
     newfactor * 4 * (factor + 1) end
  end
  ```

Contraction of Function Application

$$\frac{E[x \leftarrow v]E'}{(\texttt{fun}\ \{\cdot\}\ x\ \texttt{->}\ E\ \texttt{end}\quad v) >_{\text{simPL}} E'}\ [\text{CallFun}]$$

Contraction of Recursive Function Application

$$\frac{E[f \leftarrow \texttt{recfun}\ \{\cdot\}\ f\ x\ \texttt{->}\ E\ \texttt{end}]E' \quad E'[x \leftarrow v]E''}{(\texttt{recfun}\ f\ x\ \texttt{->}\ E\ \texttt{end}\quad v) >_{\text{simPL}} E''}\,[\text{RF}]$$

One-Step Evaluation

$$\frac{E >_{\text{simPL}} E'}{E \mapsto_{\text{simPL}} E'}[\text{Contraction}]$$

$$\frac{E \mapsto_{\text{simPL}} E'}{p_1[E] \mapsto_{\text{simPL}} p_1[E']}[\text{OpArg}_1]$$

One-Step Evaluation (cont'd)

$$\frac{E_1 \mapsto_{\text{simPL}} E_1'}{p_2[E_1, E_2] \mapsto_{\text{simPL}} p_2[E_1', E_2]} [\text{OpArg}_2]$$

$$\frac{E_2 \mapsto_{\text{simPL}} E_2'}{p_2[v_1, E_2] \mapsto_{\text{simPL}} p_2[v_1, E_2']} [\text{OpArg}_3]$$

One-Step Evaluation (cont'd)

$$\frac{E \mapsto_{\text{simPL}} E'}{\texttt{if } E \texttt{ then } E_1 \texttt{ else } E_2 \texttt{end} \mapsto_{\text{simPL}} \texttt{if } E' \texttt{ then } E_1 \texttt{ else } E_2 \texttt{ end}}$$

One-Step Evaluation (cont'd)

$$\frac{E \mapsto_{\rm simPL} E'}{(E\ E_1 \ldots E_n) \mapsto_{\rm simPL} (E'\ E_1 \ldots E_n)} [\text{AppFun}]$$

One-Step Evaluation (cont'd)

$$\frac{E_i \mapsto_{\text{simPL}} E_i'}{(v\ \ v_1 \ldots v_{i-1}\ \ E_i \ldots E_n) \mapsto_{\text{simPL}} (v\ \ v_1 \ldots v_{i-1}\ \ E_i' \ldots E_n)}\ [\text{AppArg}]$$

Evaluation of simPL Programs

As for ePL, evaluation of simPL is defined by the evalution relation $\mapsto^*_{\mathrm{simPL}}$, the reflexive transitive closure of $\mapsto_{\mathrm{simPL}}$.