SWEN430 - Compiler Engineering

Lecture 6 - Operational Semantics II

Dr David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Semantics for $\lambda_{\mathbb{W}}$

- Semantics specified in **small step** style as transitions:

$$\langle \Sigma, \mathtt{e} \rangle \longrightarrow \langle \Sigma', \mathtt{e}' \rangle$$

  - Here, Σ represents the **call stack**
  - Call stack made up from one or more **stack frames** (or **activation records**), σ, separated by "::" (i.e. $\Sigma = \{\sigma_1 :: \sigma_2 :: \ldots :: \sigma_n\}$)
  - Each stack frame is a map from variable names to values
  - We will use notation $\Sigma(n)$ to look up the value stored in variable named *n* by checking the stack frames *from the top of the stack downwards*
  - We will use notation $\Sigma :: \{n \mapsto v\}$ to add a new mapping from variable named *n* to value *v* to the top of the stack
  - Finally, we will use notation $\Sigma[n \mapsto v]$ to replace a mapping already at the top of the stack, and if its missing, to add new mapping for *n* to the topmost stack frame.

Examples

$$\langle \{\mathtt{x} \mapsto 1\}, \mathtt{x} + 1 \rangle \longrightarrow \langle \{\mathtt{x} \mapsto 1\}, 1 + 1 \rangle \longrightarrow \langle \{\mathtt{x} \mapsto 1\}, 2 \rangle$$

$$\langle \{\mathtt{x} \mapsto 1\}, \mathtt{x} = \mathtt{x} + 1; \ \mathtt{return}\ \mathtt{x} \rangle$$

$$\hookrightarrow \langle \{\mathtt{x} \mapsto 1\}, \mathtt{x} = 1 + 1; \ \mathtt{return}\ \mathtt{x} \rangle$$

$$\hookrightarrow \langle \{\mathtt{x} \mapsto 1\}, \mathtt{x} = 2; \ \mathtt{return}\ \mathtt{x} \rangle$$

$$\hookrightarrow \langle \{\mathtt{x} \mapsto 2\}, \mathtt{return}\ \mathtt{x} \rangle$$

$$\hookrightarrow \langle \varnothing, 2 \rangle$$

Semantics for $\lambda_{\mathbb{W}}$ (Part 1 — Expressions)

$$\frac{\Sigma(n) = v}{\langle \Sigma, n \rangle \longrightarrow \langle \Sigma, v \rangle} \quad \text{(R-VAR)}$$

$$\frac{\langle \Sigma, e_1 \rangle \longrightarrow \langle \Sigma, e_1' \rangle}{\langle \Sigma, e_1(e_2) \rangle \longrightarrow \langle \Sigma, e_1'(e_2) \rangle} \quad \text{(R-APP1)}$$

$$\frac{\langle \Sigma, e_2 \rangle \longrightarrow \langle \Sigma, e_2' \rangle}{\langle \Sigma, v_1(e_2) \rangle \longrightarrow \langle \Sigma, v_1(e_2') \rangle} \quad \text{(R-APP2)}$$

$$\frac{v_1 = \lambda x.\overline{s}}{\langle \Sigma, v_1(v_2) \rangle \longrightarrow \langle \Sigma :: \{x \mapsto v_2\}, \overline{s} \rangle} \quad \text{(R-APP3)}$$

$$\frac{\langle \Sigma, e_1 \rangle \longrightarrow \langle \Sigma, e_1' \rangle}{\langle \Sigma, e_1 \ \mathrm{op} \ e_2 \rangle \longrightarrow \langle \Sigma, e_1' \ \mathrm{op} \ e_2 \rangle} \quad \text{(R-BINARY1)}$$

$$\frac{\langle \Sigma, e_2 \rangle \longrightarrow \langle \Sigma, e_2' \rangle}{\langle \Sigma, v_1 \ \mathrm{op} \ e_2 \rangle \longrightarrow \langle \Sigma, v_1 \ \mathrm{op} \ e_2' \rangle} \quad \text{(R-BINARY2)}$$

$$\frac{\vdash v_1 \ \mathrm{op} \ v_2 = v_3}{\langle \Sigma, v_1 \ \mathrm{op} \ v_2 \rangle \longrightarrow \langle \Sigma, v_3 \rangle} \quad \text{(R-BINARY3)}$$

Semantics for $\lambda_{\mathbb{W}}$ (Part 2 — Statements & Programs)

$$\frac{}{\langle \sigma_1 :: \ldots :: \sigma_n :: \sigma_{n+1}, \texttt{return}\ v \rangle \longrightarrow \langle \sigma_1 :: \ldots :: \sigma_n, v \rangle} \quad \text{(R-RETURN1)}$$

$$\frac{\langle \Sigma, e \rangle \longrightarrow \langle \Sigma, e' \rangle}{\langle \Sigma, \texttt{return}\ e \rangle \longrightarrow \langle \Sigma, \texttt{return}\ e' \rangle} \quad \text{(R-RETURN2)}$$

$$\frac{\Sigma' = \Sigma[n \mapsto v]}{\langle \Sigma, n = v\ ;\ \overline{s} \rangle \longrightarrow \langle \Sigma', \overline{s} \rangle} \quad \text{(R-ASSIGN1)}$$

$$\frac{\langle \Sigma, e \rangle \longrightarrow \langle \Sigma, e' \rangle}{\langle \Sigma, n = e\ ;\ \overline{s} \rangle \longrightarrow \langle \Sigma, n = e'\ ;\ \overline{s} \rangle} \quad \text{(R-ASSIGN2)}$$

$$\frac{f_1 = T_1\ n_1 (T_2\ n_2)\ \{\ \overline{s}\ \}}{\langle \Sigma, f_1 \ldots f_n\ e \rangle \longrightarrow \langle \Sigma[n_1 \mapsto \lambda n_2 : T_2.\overline{s}], f_2 \ldots f_n\ e \rangle} \quad \text{(R-PROG)}$$

An Example Reduction

- $\left\langle \emptyset,\ \texttt{int f(int x)\{ return x \} f(1)} \right\rangle$

$\hookrightarrow \left\langle \{\texttt{f} \mapsto \lambda\texttt{x.(return x)}\},\ \texttt{f(1)} \right\rangle$ (R-PROG)

$\hookrightarrow \left\langle \{\texttt{f} \mapsto \lambda\texttt{x.(return x)}\},\ (\lambda\texttt{x.(return x)})(1) \right\rangle$ (R-VAR, R-APP1)

$\hookrightarrow \left\langle \{\texttt{f} \mapsto \lambda\texttt{x.(return x)}\} :: \{\texttt{x} \mapsto 1\},\ (\texttt{return x}) \right\rangle$ (R-APP3)

$\hookrightarrow \left\langle \{\texttt{f} \mapsto \lambda\texttt{x.(return x)}\} :: \{\texttt{x} \mapsto 1\},\ (\texttt{return } 1) \right\rangle$ (R-VAR,R-RETURN2)

$\hookrightarrow \left\langle \{\texttt{f} \mapsto \lambda\texttt{x.(return x)}\},\ 1 \right\rangle$ (R-RETURN1)

Curious λ_W Programs

- Consider the following λ_W programs:

  1) `1 + true`

  2) `int f(int x){ x = true; return x } f(1)`

  3) `int f(int x){ return x } f(true)`

Q) Which of these will **still reduce** under our semantics?

Extending λ_W

- There are lots of **missing features** from λ_W!
  - **Conditional** statements — i.e. `if(e) { ... } else { ... }`
  - **Loop** statements — i.e. `while(e) { ... }`
  - **Variable Declarations** — i.e. `int x = 1`
  - **Record data types** — i.e. `{int x, int y}, r.f, etc`
  - **Array data types** — i.e. `[int], [1,2,3], |xs|, etc`
- Let's consider how to **extend** λ_W with these features...

Conditional Statements

s	$::=$	$\ldots$	*statements*
		$\texttt{if(e)}\,\{\overline{s_t}\}\,\texttt{else}\,\{\overline{s_f}\}$	*conditional*

$$\frac{\langle \Sigma, e\rangle \longrightarrow \langle \Sigma, e'\rangle}{\langle \Sigma, \texttt{if(e)}\,\{\overline{s_t}\}\,\texttt{else}\,\{\overline{s_f}\}\rangle \longrightarrow \langle \Sigma, \texttt{if(e')}\,\{\overline{s_t}\}\,\texttt{else}\,\{\overline{s_f}\}\rangle} \quad \text{(R-IF1)}$$

$$\frac{}{\langle \Sigma, \texttt{if(true)}\,\{\overline{s_t}\}\,\texttt{else}\,\{\overline{s_f}\}\rangle \longrightarrow \langle \Sigma, \overline{s_t}\rangle} \quad \text{(R-IF2)}$$

$$\frac{}{\langle \Sigma, \texttt{if(false)}\,\{\overline{s_t}\}\,\texttt{else}\,\{\overline{s_f}\}\rangle \longrightarrow \langle \Sigma, \overline{s_f}\rangle} \quad \text{(R-IF3)}$$

- Example:

$$\begin{aligned}
&\langle\, \{x \mapsto \texttt{true}\}, \texttt{if(x)}\,\{\texttt{return 1}\}\,\texttt{else}\,\{\texttt{return 2}\}\,\rangle \\
&\hookrightarrow \langle\, \{x \mapsto \texttt{true}\}, \texttt{if(true)}\,\{\texttt{return 1}\}\,\texttt{else}\,\{\texttt{return 2}\}\,\rangle \quad \text{(R-VAR, R-IF1)} \\
&\hookrightarrow \langle\, \{x \mapsto \texttt{true}\}, \texttt{return 1}\,\rangle \quad \text{(R-IF2)}
\end{aligned}$$

Loops

`s`	`::=`	*statements*
	`...`	
	`while(e){` $\overline{\texttt{s}}$ `}`	*while loop*

$$\frac{}{\langle \Sigma, \texttt{while(e)\{}\ \overline{\texttt{s}}\ \texttt{\}}\rangle \longrightarrow \langle \Sigma, \texttt{if(e)\{}\ \overline{\texttt{s}}\ \texttt{; while(e)\{}\ \overline{\texttt{s}}\ \texttt{\}\}else\{\}}\rangle}\ \text{(R-WHILE)}$$

Loops Example

$$
\begin{array}{lr}
\langle\, \{\mathtt{x} \mapsto 1\}, \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\{\ldots\}\,\rangle & \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 1\}, \mathtt{if}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1; \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\}\{\ldots\}\,\rangle & \text{(R-WHILE)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 1\}, \mathtt{if}(1 < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1; \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\}\{\ldots\}\,\rangle & \text{(R-VAR, R-IF1)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 1\}, \mathtt{if}(\mathtt{true})\,\{\ \mathtt{x} = \mathtt{x} + 1; \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\}\{\ldots\}\,\rangle & \text{(R-BINARY3, R-IF1)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 1\}, \mathtt{x} = \mathtt{x} + 1; \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\{\ldots\}\,\rangle & \text{(R-IF2)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 1\}, \mathtt{x} = 2; \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\{\ldots\}\,\rangle & \text{(R-VAR,R-BINARY3)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 2\}, \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\{\ldots\}\,\rangle & \text{(R-ASSIGN1)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 2\}, \mathtt{if}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1; \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\}\{\ldots\}\,\rangle & \text{(R-WHILE)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 2\}, \mathtt{if}(2 < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1; \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\}\{\ldots\}\,\rangle & \text{(R-VAR,R-IF1)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 2\}, \mathtt{if}(\mathtt{false})\,\{\ \mathtt{x} = \mathtt{x} + 1; \mathtt{while}(\mathtt{x} < 2)\,\{\ \mathtt{x} = \mathtt{x} + 1\ \}\}\{\ldots\}\,\rangle & \text{(R-BINARY3,R-IF1)} \\
\hookrightarrow \langle\, \{\mathtt{x} \mapsto 2\}, \{\ldots\}\,\rangle & \text{(R-IF3)}
\end{array}
$$

Variable Declarations

$$\begin{array}{lll} \mathtt{s} & ::= & \ldots \qquad\qquad \textit{statements} \\ & & \mathtt{T\ n = e} \quad \textit{variable declaration} \end{array}$$

$$\frac{\Sigma' = \Sigma[\mathtt{n} \mapsto \mathtt{v}]}{\langle \Sigma, \mathtt{T\ n = v\ ;\ \overline{s}} \rangle \longrightarrow \langle \Sigma', \mathtt{\overline{s}} \rangle} \quad \text{(R-VarDecl1)}$$

$$\frac{\langle \Sigma, \mathtt{e} \rangle \longrightarrow \langle \Sigma, \mathtt{e'} \rangle}{\langle \Sigma, \mathtt{T\ n = e\ ;\ \overline{s}} \rangle \longrightarrow \langle \Sigma, \mathtt{T\ n = e'\ ;\ \overline{s}} \rangle} \quad \text{(R-VarDecl2)}$$

- Variable declarations are **surprisingly straightforward...**

  ... at least from the perspective of **semantics**!

Records

e	$::=$	$\ldots$	*expressions*
		$\{\overline{n : e}\}$	*record constructor*
		$e.f$	*field access*
T	$::=$	$\ldots$	*types*
		$\{\overline{T\ n}\}$	*record type*

$$\frac{\langle \Sigma, e_k \rangle \longrightarrow \langle \Sigma, e'_k \rangle}{\langle \Sigma, \{\overline{n : v}, n_k : e_k, \overline{n : e}\} \rangle \longrightarrow \langle \Sigma, \{\overline{n : v}, n_k : e'_k, \overline{n : e}\} \rangle} \quad \text{(R-RECORD)}$$

$$\frac{\langle \Sigma, e \rangle \longrightarrow \langle \Sigma, e' \rangle}{\langle \Sigma, e.f \rangle \longrightarrow \langle \Sigma, e'.f \rangle} \quad \text{(R-FIELD1)}$$

$$\frac{}{\langle \Sigma, \{f : v, \overline{n : v}\}.f \rangle \longrightarrow \langle \Sigma, v \rangle} \quad \text{(R-FIELD2)}$$

$$\langle \{x \mapsto 1\}, \{f_1 : x, f_2 : 3\}.f_1 \rangle \longrightarrow \langle \{x \mapsto 1\}, \{f_1 : 1, f_2 : 3\}.f_1 \rangle \longrightarrow \langle \{x \mapsto 1\}, 1 \rangle$$

Arrays

$$
\begin{array}{lcll}
e & ::= & \ldots & \textit{expressions} \\
 & & [\overline{e}] & \textit{array constructor} \\
 & & e_1[e_2] & \textit{array access} \\
 & & & \\
T & ::= & \ldots & \textit{types} \\
 & & T[] & \textit{array type}
\end{array}
$$

$$\frac{\langle \Sigma, e_k \rangle \longrightarrow \langle \Sigma, e_k' \rangle}{\langle \Sigma, [\overline{v}, e_k, \overline{e}] \rangle \longrightarrow \langle \Sigma, [\overline{v}, e_k', \overline{e}] \rangle} \quad \text{(R-ARRAY)}$$

$$\frac{\langle \Sigma, e_1 \rangle \longrightarrow \langle \Sigma, e_1' \rangle}{\langle \Sigma, e_1[e_2] \rangle \longrightarrow \langle \Sigma, e_1'[e_2] \rangle} \quad \text{(R-ACCESS1)}$$

$$\frac{\langle \Sigma, e_2 \rangle \longrightarrow \langle \Sigma, e_2' \rangle}{\langle \Sigma, v_1[e_2] \rangle \longrightarrow \langle \Sigma, v_1[e_2'] \rangle} \quad \text{(R-ACCESS2)}$$

$$\frac{v = k - 1}{\langle \Sigma, [v_1, \ldots, v_k, \ldots, v_n][v] \rangle \longrightarrow \langle \Sigma, v_k \rangle} \quad \text{(R-ACCESS3)}$$

$$\langle \{x \mapsto 1\}, [1, x, 2][x] \rangle \rightarrow \langle \{x \mapsto 1\}, [1, 1, 2][x] \rangle \rightarrow \langle \{x \mapsto 1\}, [1, 1, 2][1] \rangle \rightarrow \langle \{x \mapsto 1\}, 1 \rangle$$