SWEN430 - Compiler Engineering

Lecture 8 - Typing II

Lindsay Groves & David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Syntax for $\lambda_{\mathtt{W}}$ — a subset of While

$\mathtt{p}$	$::=$	$\mathtt{f}_1 \ldots \mathtt{f}_n\ \mathtt{e}$	*programs*
$\mathtt{f}$	$::=$	$\mathtt{T}_1\ \mathtt{n}_1(\mathtt{T}_2\ \mathtt{n}_2)\{\ \overline{\mathtt{s}}\ \}$	*functions*
$\mathtt{e}$	$::=$		*expressions*
	$\mid$	$\mathtt{b}$	*logical constants*
	$\mid$	$\mathtt{c}$	*numeric constants*
	$\mid$	$\mathtt{n}$	*variables*
	$\mid$	$\mathtt{e}_1\ \mathtt{op}\ \mathtt{e}_2$	*binary*
	$\mid$	$\mathtt{n}(\mathtt{e})$	*application*
$\mathtt{s}$	$::=$	$\mathtt{n} = \mathtt{e}\,;\ \mid\ \mathtt{if}\ (\mathtt{e})\ \mathtt{s}_1\ \mathtt{else}\ \mathtt{s}_2\ \mid\ \mathtt{return}\ \mathtt{e}\,;$	*statements*
$\mathtt{b}$	$::=$	$\mathtt{true} \mid \mathtt{false}$	*truth values*
$\mathtt{c}$	$::=$	$\ldots \mid -1 \mid 0 \mid 1 \mid \ldots$	*numeric values*
$\mathtt{T}$	$::=$	$\mathtt{bool} \mid \mathtt{int}$	*types*
$\mathtt{op}$	$::=$	$`==' \mid `!=' \mid '+' \mid '-' \mid '*'$	*operators*
	$\mid$	$`<' \mid `<=' \mid `>=' \mid `>'$	

This uses a form of grammar notation often used in PL theory work.

Simplifications in $\lambda_{\mathtt{W}}$

- Functions accept **one parameter** and always **return something**
- No **variable declarations** (other than for parameters)
- No **unary operators**, and only a few **binary operators**
- Only types are `int` and `bool`
- No **compound data types** (i.e. records, lists)
- Only statements are **assignment**, **if** and **return**
- Every **program** has an expression to be evaluated (rather than have `main()` function)

Can easily be extended to include other features.

Example $\lambda_{\mathtt{W}}$ Programs

- **Valid** Programs:

  1) `1 + 1`

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

  3) ```
     int f(int x) { return x + 1; }
     int g(bool x) { return 1; }
     f(g(true))
     ```

- **Invalid** (but syntactically correct) Programs:

  1) `1 + true`

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

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

Some Notes on the Notation

- $\dfrac{\text{A}}{\text{B}}$ (Rule-Name) is used to show what requirements (A) B has.
  - The rule is called Rule-Name.
  - If A holds, then B holds. (Forwards)
  - You can show that B holds by showing that A holds. (Backwards)
  - If A is empty B is always true
  - To prove A you recursively apply more rules, until no more are necessary.
- Write $\Gamma \vdash e : T$ to mean that e has type T in environment Γ.
- Write $\Gamma \vdash s\ OK$ to mean that s is well-typed.
- Γ is an *environment*, recording declarations that are in scope.

Type rules for expressions

$$\frac{}{\vdash \mathtt{n} : \mathtt{int}} \text{ (T-Num)} \qquad \frac{}{\vdash \mathtt{b} : \mathtt{bool}} \text{ (T-Bool)} \qquad \frac{\mathtt{x} : \mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \text{ (T-Var)}$$

$$\frac{\Gamma \vdash e_1 : int,\ \Gamma \vdash e_2 : int,\ op \in \{`+', `-', `*'\}}{\vdash \mathtt{e}_1\ \mathtt{op}\ \mathtt{e}_2 : \mathtt{int}} \text{ (T-AOp)}$$

$$\frac{\Gamma \vdash e_1 : int,\ \Gamma \vdash e_2 : int,\ op \in \{`<', `<=', `>', `=>'\}}{\vdash \mathtt{e}_1\ \mathtt{op}\ \mathtt{e}_2 : \mathtt{bool}} \text{ (T-Rel)}$$

$$\frac{\Gamma \vdash e_1 : T_1,\ \Gamma \vdash e_2 : T_2, T_1 = T_2,\ op \in \{`==', `!='\}}{\vdash \mathtt{e}_1\ \mathtt{op}\ \mathtt{e}_2 : \mathtt{bool}} \text{ (T-Eq)}$$

Type rules for statements, functions and programs

$$\frac{\Gamma \vdash \mathtt{x} : \mathtt{T}_1, \quad \Gamma \vdash \mathtt{e} : \mathtt{T}_2, \quad T_1 = T_2}{x = e\ OK} \text{ (T-Asgn)}$$

$$\frac{\Gamma \vdash \mathtt{e} : \mathtt{bool}, \quad \Gamma \vdash \mathtt{s}_1\ \mathtt{OK}, \quad \Gamma \vdash \mathtt{s}_2\ \mathtt{OK}}{\Gamma \vdash \text{if } (e)\ s_1 \text{ else } s_2 \text{ OK}} \text{ (T-If)}$$

$$\frac{\Gamma \cup \{n_2 : T_2\} \vdash \overline{\mathtt{s}} : \mathtt{T}_1}{\Gamma \vdash T_1 n_1 (T_2 n_2) \overline{\mathtt{s}}\ OK} \text{ (T-Fun)}$$

$$\frac{\Gamma \vdash \overline{m}\ OK, \quad \Gamma \cup \{\overline{m}\} \vdash \mathtt{e} : \mathtt{T}}{\Gamma \vdash \overline{m}\ \mathtt{e} : \mathtt{T}} \text{ (T-Prog)}$$

Ex: What's missing?

Precision of Type Checking

Progress Theorem (Soundness)

A well-typed term `t` is not stuck (either `t` is a value or there exists some transition $\mathtt{t} \longrightarrow \mathtt{t}'$)

Preservation Theorem (Soundness)

If a well-typed term is evaluated one step, then the resulting term is also well typed (in fact, it has the same type)

Completeness

If evaluating term `t` does not get **stuck**, then there exists a valid typing of `t`