SWEN430 - Compiler Engineering (2018)

Lecture 8 - Typing II : Formal Type Systems

Lindsay Groves & David J. Pearce

School of Engineering and Computer Science
Victoria University of Wellington

Defining Type Systems

- We can define the semantics and type systems for programming languages as sets of *inference rules*, of the form:

  $$\frac{A_1,\ \ldots,\ A_n}{B}\ \textit{(Rule-Name)}$$

  If the *premises* (A_1 ... A_n) all hold, then the *conclusion* (B) must also hold.

- Type correctness is expressed in terms of *judgements* of the form:
  - $\Gamma \vdash s\ OK$ "*s* is well-typed in Γ"
  - $\Gamma \vdash e : T$ "*s* is well-typed has type *T* in Γ"

- Γ is an *environment*, recording declarations that are in scope.

The λ-calculus (a minor variation of)

$\mathtt{t}$	$::=$		(Terms)
	$\mid$	$\mathtt{x}$	(Variables)
	$\mid$	$\mathtt{v}$	(Values)
	$\mid$	$\mathtt{t}\ \mathtt{t}$	(App)
$\mathtt{v}$	$::=$		(Values)
	$\mid$	$\lambda\mathtt{x.t}$	(Function)
	$\mid$	$\mathtt{c}$	(Integer)

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(R-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(R-App2)}$$

$$\frac{}{(\lambda\mathtt{x.t}_1)\ \mathtt{v}_1 \longrightarrow \mathtt{t}_1[\mathtt{x} \mapsto \mathtt{v}_1]} \quad \text{(R-App3)}$$

- A **simple** language — *syntax* and *semantics* on one slide!
- Useful starting point for **formalising** programming languages
- Term $\mathtt{t}_1[\mathtt{x} \mapsto \mathtt{t}_2]$ is $\mathtt{t}_1$ with all occurences of $\mathtt{x}$ replaced with $\mathtt{t}_2$

Example λ-calculus programs

- $(\lambda \mathtt{x}.\mathtt{x})\ (\lambda \mathtt{y}.1) \longrightarrow (\lambda \mathtt{y}.1)$

- $\Big(\big(\lambda \mathtt{x}.(\lambda \mathtt{y}.\mathtt{x}\ \mathtt{y})\big)\big(\lambda \mathtt{z}.\mathtt{z}\big)\Big)\ 1 \longrightarrow \Big(\lambda \mathtt{y}.\big((\lambda \mathtt{z}.\mathtt{z})\ \mathtt{y}\big)\Big)\ 1$
  $\longrightarrow (\lambda \mathtt{y}.\mathtt{y})\ 1 \longrightarrow 1$

- $(\lambda \mathtt{x}.\mathtt{x}\ \mathtt{x})(\lambda \mathtt{y}.\mathtt{y}\ \mathtt{y}) \longrightarrow ?$

- $\Bigg(\Big(\lambda \mathtt{x}.(\lambda \mathtt{y}.\mathtt{x}\ \mathtt{y})\Big)\ \Big((\lambda \mathtt{z}.\mathtt{z})\ 1\Big)\Bigg)\ 1 \longrightarrow ?$

A few notes on the λ-calculus

- How does $(\lambda \mathtt{y}.(\lambda \mathtt{y}.\mathtt{y}\ 1))\ (\lambda \mathtt{x}.\mathtt{x})$ reduce?
  - This is the *Variable Capture* problem
  - We assume that λ parameters have unique names!
  - Can enforce this by renaming parameters in body of λ term

- *Currying* gives functions with multiple arguments!
  - $\lambda \mathtt{x}, \mathtt{y}.(\ldots)$ is equivalent to $\lambda \mathtt{x}.(\lambda \mathtt{y}.(\ldots))$

- All control-structures (e.g. `if`, `while`) can be implemented in λ-Calculus!

- Where to find more?
  - *Types and Programming Languages*, by Benjamin Pierce is an excellent book!
  - There are a lot of resources on the internet as well (e.g. *wikipedia*)

Some Notes on the Notation

- $\frac{A}{B}$ (Rule-Name) is used to show what requirements (A) B has. The rule is called Rule-Name.
  - If A is empty B is always true
  - To prove B you recursively apply more rules, until no more are necessary.

Type Checking

$$\left(\left(\lambda \mathtt{x}.(\lambda \mathtt{y}.\mathtt{x}\ \mathtt{y})\right)\ \left((\lambda \mathtt{z}.\mathtt{z})\ \mathtt{1}\right)\right)\ \mathtt{1}$$

- This program gets **stuck** before producing a *value*!
  - In languages like C, programs never get stuck ... they just crash!
  - In typesafe languages (e.g. Java), programs always raise errors before doing bad things
- Type checking checks our program will not get stuck
  - This is the approach used in **statically typed** languages
- Type checking suffers from **limitations of precision**
  - I.e. Correct programs can fail to type check

Simply Typed λ-Calculus ($\lambda_\rightarrow$) - Syntax and Semantics

$$\begin{array}{lll} \mathtt{t} ::= & & \text{(Terms)} \\ & \mid \mathtt{x} & \text{(Variables)} \\ & \mid \mathtt{v} & \text{(Values)} \\ & \mid \mathtt{t}\ \mathtt{t} & \text{(Apps)} \end{array}$$

$$\begin{array}{lll} \mathtt{v} ::= & & \text{(Values)} \\ & \mid \lambda \mathtt{x} : \mathtt{T}.\mathtt{t} & \text{(Function)} \\ & \mid \mathtt{c} & \text{(Integer)} \end{array}$$

$$\begin{array}{lll} \mathtt{T} ::= & & \text{(Types)} \\ & \mid \mathtt{T} \rightarrow \mathtt{T} & \text{(Fun type)} \\ & \mid \mathtt{int} & \text{(Int type)} \end{array}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(R-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(R-App2)}$$

$$\frac{}{(\lambda \mathtt{x} : \mathtt{T}.\mathtt{t}_2)\ \mathtt{v}_1 \longrightarrow \mathtt{t}_2[\mathtt{x} \mapsto \mathtt{v}_1]} \quad \text{(R-App3)}$$

Some Notation Used in Typing

- Γ is the **typing environment**
  - It's a set of pairs $(\mathtt{v}, \mathtt{T})$ which maps variables to their types
  - It remembers what type you used when declaring a variable (e.g. x is an int)
- $\vdash$ is used to say that the things on the left can be used to say that everything on the right is ok.
  - $\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1$ means: *in the typing environment* Γ*, term* t_1 *can be shown to have type* T_1 *using the typing rules above*

Type checking rules for $\lambda_{\rightarrow}$

$$\frac{}{\vdash \mathtt{n} : \mathtt{int}} \quad \text{(T-Int)} \qquad \frac{\mathtt{x} : \mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \quad \text{(T-Var)}$$

$$\frac{\Gamma \cup \{\mathtt{x} : \mathtt{T}_1\} \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x} : \mathtt{T}_1 . \mathtt{t}_2 : \mathtt{T}_1 \longrightarrow \mathtt{T}_2} \quad \text{(T-Fun)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1 \longrightarrow \mathtt{T}_2 \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_1}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_2} \quad \text{(T-App)}$$

Examples: 1 $\quad (\lambda x : int.x) \quad (\lambda x : int \rightarrow int.x)$

Example derivation tree for $\lambda_{\rightarrow}$

- A typed version of our **stuck** term:

$$\bigg(\Big(\lambda \mathtt{x} : \mathtt{int} \rightarrow \mathtt{int}.(\lambda \mathtt{y} : \mathtt{int}.\mathtt{x}\ \mathtt{y})\Big)\Big((\lambda \mathtt{z} : \mathtt{int}.\mathtt{z})\ 1\Big)\bigg)\ 1$$

- A *derivation tree* can be used to check the term's type:

$$\dfrac{\dfrac{\{\mathtt{z} : \mathtt{int}\} \vdash \mathtt{z} : \mathtt{int}}{\vdash (\lambda \mathtt{z} : \mathtt{int}.\mathtt{z}) : \mathtt{int} \rightarrow \mathtt{int}}\ \text{T-Fun} \qquad \dfrac{}{\vdash 1 : \mathtt{int}}\ \text{T-Int}}{\vdash ((\lambda \mathtt{z} : \mathtt{int}.\mathtt{z})\ 1) : \mathbf{int}}\ \text{T-App}$$

$$\dfrac{\dfrac{\{\mathtt{x} : \mathtt{int} \rightarrow \mathtt{int}, \mathtt{y} : \mathtt{int}\} \vdash (\mathtt{x}\ \mathtt{y}) : \mathtt{int}}{\{\mathtt{x} : \mathtt{int} \rightarrow \mathtt{int}\} \vdash (\lambda \mathtt{y} : \mathtt{int}.(\mathtt{x}\ \mathtt{y})) : \mathtt{int}}\ \text{T-Fun}}{\vdash (\lambda \mathtt{x} : \mathtt{int} \rightarrow \mathtt{int}.(\lambda \mathtt{y} : \mathtt{int}.(\mathtt{x}\ \mathtt{y}))) : (\mathbf{int} \rightarrow \mathbf{int}) \rightarrow (\mathbf{int} \rightarrow \mathbf{int})}\ \text{T-Fun}$$

- No valid typing for `int` applied to $(\mathtt{int} \rightarrow \mathtt{int}) \rightarrow (\mathtt{int} \rightarrow \mathtt{int})$
  - So, our **stuck** term will not type check

Precision of Type Checking

Progress Theorem (Soundness)

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

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 $\mathtt{t}$ does not get **stuck**, then there exists a valid typing of $\mathtt{t}$

- The $\lambda_{\to}$ type system is sound, but not complete (this is impossible)
- There are valid programs which cannot be typed, such as:

$$\Big(\lambda\mathtt{f}.\mathtt{f}\ \mathtt{f}\ \mathtt{1}\Big)\ \lambda\mathtt{x}.\mathtt{x} \longrightarrow \Big((\lambda\mathtt{x}.\mathtt{x})\ (\lambda\mathtt{x}.\mathtt{x})\Big)\ \mathtt{1} \longrightarrow (\lambda\mathtt{x}.\mathtt{x})\ \mathtt{1} \longrightarrow \mathtt{1}$$