

Type-Checking

Lecture 2

Outline

- A language
 - Lambda calculus
- Types
 - Type checking

The Typed Lambda Calculus

- Lambda calculus
 - But types are assigned to bound variables.
 - Pascal, or C
- Add integers, addition, if-then-else
- Note: Not every expression generated by this grammar is a properly typed term.

$e = x \mid \lambda x : \tau . e \mid e \ e \mid i \mid e + e \mid \text{if } e \ e \ e$

Why Lambda Calculus?

- History
 - Introduced by Church in the 1930's
 - Small, paradigmatic language
- Computation via *beta reduction*
 - Function call
 - substitution of argument for formal parameter
 - Roughly:

$$(\lambda x:\tau.e) e' , \beta e[x * e']$$

Simple Examples

$$(\lambda x:\tau.x) \gamma, \beta \gamma$$

$$(\lambda x:\text{int}.x+1) 3, \beta 3+1, 4$$

Types

- Function types
- Integers
- Type variables
 - Stand for definite, but unknown, types

$$\tau = \alpha \mid \tau \rightarrow \tau \mid \text{int}$$

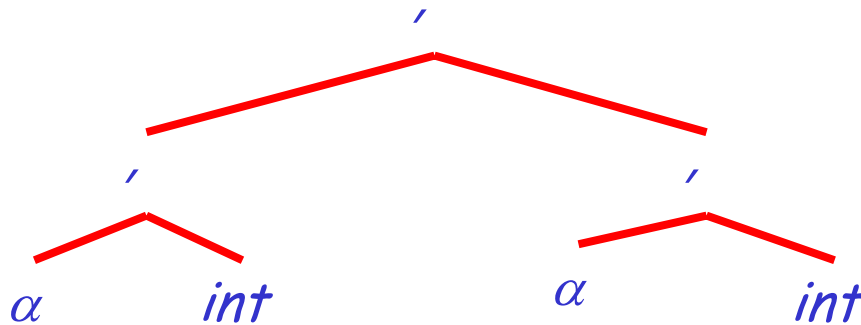
Function Types

- Intuitively, a type τ_1 , τ_2 stands for the set of functions that map arguments of type τ_1 to results of type τ_2 .
- Placeholder for any other structured datatype
 - Lists
 - Trees
 - Arrays

Types are Trees

- Types are terms
- Any term can be represented by a tree
 - The parse tree of the term
 - Tree representation is important in algorithms

$(\alpha, int), \alpha, int$



Examples

- We write $e:t$ for the statement “ e has type t .”

$\lambda x:\alpha. x : \alpha \rightarrow \alpha$

$\lambda x:\alpha. \lambda y:\beta. x : \alpha \rightarrow \beta \rightarrow \alpha$

$\lambda f:\alpha \rightarrow \beta. \lambda g:\beta \rightarrow \gamma. \lambda x:\alpha. g(f\ x) : (\alpha \rightarrow \beta) \rightarrow (\beta \rightarrow \gamma) \rightarrow \alpha \rightarrow \gamma$

$\lambda f:\alpha \rightarrow \beta \rightarrow \gamma. \lambda g:\alpha \rightarrow \beta. \lambda x:\alpha. (f\ x)\ (g\ x) : (\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \gamma$

Type Environments

- To determine whether the types in an expression are consistent we perform *type checking*.
- But we need types for free variables, too!
- A *type environment* is a function from variables to types. The syntax of environments is:

$$A = \emptyset \mid A, x : \tau$$

- The meaning is:

$$(A, x : \tau)(y) = \begin{array}{ll} \tau & \text{if } x = y \\ A(y) & \text{if } x \neq y \end{array}$$

Type Checking Rules

- Type checking is done by structural induction.
 - One inference rule for each form
 - Assumptions contain types of free variables
 - A term is *well-typed* if $A \vdash e : \tau$

$$\begin{array}{c}
 \frac{A(x) = \tau}{A \vdash x : \tau} \quad \frac{A, x : \tau \vdash e : \tau'}{A \vdash \lambda x : \tau. e : \tau \rightarrow \tau'} \quad \frac{A \vdash e_1 : \tau \rightarrow \tau' \quad A \vdash e_2 : \tau}{A \vdash e_1 e_2 : \tau'} \\
 \\
 \frac{}{A \vdash i : \text{int}} \quad \frac{A \vdash e_1 : \text{int} \quad A \vdash e_2 : \text{int}}{A \vdash e_1 + e_2 : \text{int}} \quad \frac{A \vdash e_1 : \text{int} \quad A \vdash e_2 : \tau \quad A \vdash e_3 : \tau}{A \vdash \text{if } e_1 e_2 e_3 : \tau}
 \end{array}$$

Example

$$\frac{\frac{x : \alpha, y : \beta \vdash x : \alpha}{x : \alpha \vdash \lambda y : \beta. x : \beta \rightarrow \alpha}}{\emptyset \vdash \lambda x : \alpha. \lambda y : \beta. x : \alpha \rightarrow \beta \rightarrow \alpha}$$

Type Checking Algorithm

- There is a simple algorithm for type checking
- Observe that there is only one possible "shape" of the type derivation
 - only one inference rule applies to each form.

$$\frac{\frac{? \vdash x : ?}{? \vdash \lambda y : \beta. x : ?}}{\emptyset \vdash \lambda x : \alpha. \lambda y : \beta. x : ?}$$

Algorithm (Cont.)

- Walk the proof tree from the root to the leaves, generating the correct environments.
- Assumptions are simply gathered from lambda abstractions.

$$\frac{x : \alpha, y : \beta \vdash x : ?}{x : \alpha \vdash \lambda y : \beta. x : ?}$$
$$\frac{}{\emptyset \vdash \lambda x : \alpha. \lambda y : \beta. x : ?}$$

Algorithm (Cont.)

- In a walk from the leaves to the root, calculate the type of each expression.
- The types are completely determined by the type environment and the types of subexpressions.

$$\frac{x : \alpha, y : \beta \vdash x : \alpha}{x : \alpha \vdash \lambda y : \beta. x : \beta \rightarrow \alpha}$$

$$\emptyset \vdash \lambda x : \alpha. \lambda y : \beta. x : \alpha \rightarrow \beta \rightarrow \alpha$$

A Bigger Example

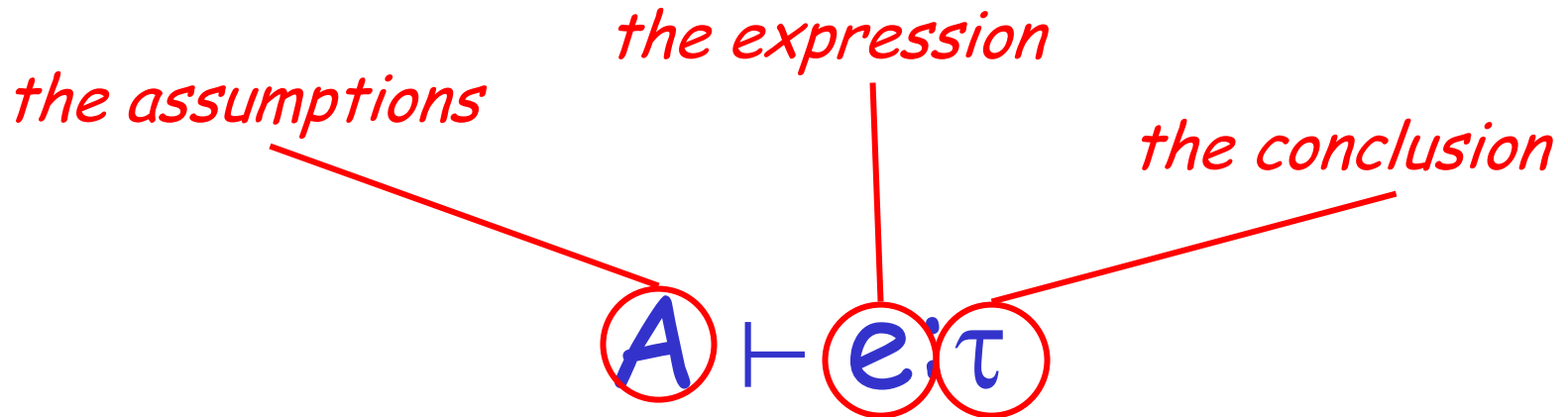
$$\frac{\frac{x : \alpha \rightarrow \alpha, y : \beta \vdash x : \alpha \rightarrow \alpha}{x : \alpha \rightarrow \alpha \vdash \lambda y : \beta. x : \beta \rightarrow \alpha \rightarrow \alpha}}{\emptyset \vdash \lambda x : \alpha \rightarrow \alpha. \lambda y : \beta. x : (\alpha \rightarrow \alpha) \rightarrow \beta \rightarrow \alpha \rightarrow \alpha} \quad \frac{z : \alpha \vdash z : \alpha}{\emptyset \vdash \lambda z : \alpha. z : \alpha \rightarrow \alpha}}{\emptyset \vdash (\lambda x : \alpha \rightarrow \alpha. \lambda y : \beta. x) \lambda z : \alpha. z : (\alpha \rightarrow \alpha) \rightarrow \beta \rightarrow \alpha \rightarrow \alpha}$$

What Do Types Mean?

- Thm. If $A \vdash e:\tau$ and $e \rightarrow_{\beta} d$, then $A \vdash d:\tau$
 - Evaluation preserves types.
- This is the basis of a claim that there can be no runtime type errors
 - functions applied to data of the wrong type
 - Adding to a function
 - Using an integer as a function

Commentary

- The use of logics to express formal derivations on programs is standard
- Always follows the same basic recipe:



Next Time ...

- LCLint
 - a debugging tool
 - based on dataflow analysis
 - compensates for C's weak type system
- Preparation
 - read 2 LCLint papers
 - also finish "Type Systems"