Types

CS242

Lecture 5

Type Systems

- There is a split in the world of programming between
  - Typed languages
  - Untyped languages
- Leave the religious debate aside for now ...
  - We will come back to why there is a debate at all
- Today the focus is on the basics of typed languages

What is a Type?

- Consensus
  - A set of values

- Examples
  - Int is the set of all integers
  - Float is the set of all floats
  - Bool is the set {true, false}

More Examples

- List(Int) is the set of all lists of integers
  - List is a *type constructor*
  - A function from types to types

- Foo, in Java, is the set of all objects of class Foo

- Int $\rightarrow$ Int is the set of functions mapping an integer to an integer
  - E.g., increment, decrement, and many others

What is a Type?

- Consensus
  - A set of values
- In typed languages
  - Every concrete value is an element of some type or types
  - Every legal program has a type
- Type systems have a well-developed notation
  - Useful for more than just type systems ...

Rules of Inference

- Inference rules have the form

  If Hypothesis is true, then Conclusion is true

- Type checking computes via reasoning

  If E_1 and E_2 have certain types, then E_3 has a certain type

- Rules of inference are a compact notation for "If-Then" statements

From English to an Inference Rule

- Start with a simplified system and gradually add features

- Building blocks
  - Symbol $\wedge$ is "and"
  - Symbol $\Rightarrow$ is "if-then"
  - $x:T$ is "x has type T"

From English to an Inference Rule (2)

If e_1 has type Int and e_2 has type Int, then $e_1 + e_2$ has type Int

$(e_1 \text{ has type Int} \wedge e_2 \text{ has type Int}) \Rightarrow e_1 + e_2 \text{ has type Int}$

$(e_1\text{: Int} \wedge e_2\text{: Int}) \Rightarrow e_1 + e_2\text{: Int}$

From English to an Inference Rule (3)

The statement

$$(e_1\text{: Int} \wedge e_2\text{: Int}) \Rightarrow e_1 + e_2\text{: Int}$$

is a special case of

$$\text{Hypothesis}_1 \wedge \ldots \wedge \text{Hypothesis}_n \Rightarrow \text{Conclusion}$$

This is an inference rule.

Notation for Inference Rules

- By tradition inference rules are written

$$\frac{\vdash \text{Hypothesis}_1 \; . \; . \; . \vdash \text{Hypothesis}_n}{\vdash \text{Conclusion}}$$

- Type rules have hypotheses and conclusions

$$\vdash e : T$$

- $\vdash$ means “it is provable that . . .”

Two Rules

$$\frac{\text{i is an integer}}{\vdash \text{i : Int}} \quad \text{[Int]}$$

$$\frac{\begin{array}{c}\vdash e_1 : \text{Int} \\ \vdash e_2 : \text{Int}\end{array}}{\vdash e_1 + e_2 : \text{Int}} \quad \text{[Add]}$$

Two Rules (Cont.)

- These rules give templates describing how to type integers and + expressions

- By filling in the templates, we can produce complete typings for expressions

- Note that
  - Hypotheses prove facts about subexpressions
  - Conclusions prove facts about the entire expression

Example: 1 + 2

$$\dfrac{\dfrac{1 \text{ is an integer}}{\vdash 1: \text{Int}} \qquad \dfrac{2 \text{ is an integer}}{\vdash 2: \text{Int}}}{\vdash 1 + 2: \text{Int}}$$

A Problem

- What is the type of a variable reference?

$$\frac{x \text{ is a variable}}{\vdash x: ?} \quad [\text{Var}]$$

- The rule does not carry enough information to give x a type.

A Solution

- Put more information in the rules!

- An *environment* gives types for free variables
  - An environment is a function from variables to types
  - Recall that a variable is free in an expression if it is not defined within the expression

Type Environments

Let A be a function from Variables to Types

The sentence $A \vdash e : T$ is read:

Under the assumption that variables have the types given by A, it is provable that the expression e has the type T

Modified Rules

The type environment is added to all rules:

$$\frac{A \vdash e_1 : \text{Int} \qquad A \vdash e_2 : \text{Int}}{A \vdash e_1 + e_2 : \text{Int}} \quad \text{[Add]}$$

New Rules

And we can write new rules:

$$\frac{A(x) = T}{A \vdash x : T} \quad [\text{Var}]$$

Summary

Describe type systems using logics of the form:

$$\frac{\begin{array}{c} A' \vdash e' : T' \\ A'' \vdash e'' : T'' \end{array}}{A \vdash e : T}$$

Type of an expression is recursively defined using types of subexpressions.

Assumptions about the free variables of e.

The type for e.

The program (or program fragment) to be analyzed.

Simply Typed Lambda Calculus

A Language of Typed Functions

Untyped lambda calculus:

$$e \rightarrow x \;|\; \lambda x.e \;|\; e\ e$$

Simply typed lambda calculus:

$$e \rightarrow x \;|\; \lambda x{:}\, t.e \;|\; e\ e \;|\; i$$

$$t \rightarrow \alpha \;|\; t \rightarrow t \;|\; int$$

Type Rules

$$\frac{}{A, x : t \vdash x : t} \quad [\text{Var}]$$

$$\frac{A, x : t \vdash e : t'}{A \vdash \lambda x{:}t.e : t \rightarrow t'} \quad [\text{Abs}]$$

$$\frac{}{A \vdash i : \text{int}} \quad [\text{Int}]$$

$$\frac{A \vdash e_1 : t \rightarrow t' \qquad A \vdash e_2 : t}{A \vdash e_1\, e_2 : t'} \quad [\text{App}]$$

Examples

$$\frac{x{:}\alpha \vdash x{:}\ \alpha}{\vdash \lambda x{:}\alpha.\ x : \alpha \to \alpha}$$

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

$$\frac{\dfrac{z{:}\ \alpha \to \beta \to \alpha \vdash z{:}\ \alpha \to \beta \to \alpha}{\vdash \lambda z{:}\ \alpha \to \beta \to \alpha\ .\ z : (\alpha \to \beta \to \alpha) \to (\alpha \to \beta \to \alpha)} \qquad \dfrac{\dfrac{x{:}\ \alpha,\ y{:}\beta \vdash x : \alpha}{x{:}\ \alpha \vdash \lambda y{:}\beta.\ x : \beta \to \alpha}}{\vdash \lambda x{:}\alpha.\ \lambda y{:}\beta.\ x : \alpha \to \beta \to \alpha}}{\vdash (\lambda z{:}\ \alpha \to \beta \to \alpha\ .\ z)\ (\lambda x{:}\alpha.\ \lambda y{:}\beta.\ x : \alpha \to \beta \to \alpha){:}\ \alpha \to \beta \to \alpha}$$

Examples

$$\frac{x{:}? \vdash 1 : int \qquad x{:}? \vdash x : ?}{\vdash \lambda x{:}?.\ 1\ \ x : \ ?}$$

$$\frac{\dfrac{x{:}? \vdash x : ? \qquad x{:}? \vdash x : ?}{x{:}? \vdash x\ x : ?}}{\vdash \lambda x{:}?.\ x\ \ x : \ ?}$$

$$\frac{x{:}? \vdash x : ?}{\vdash (\lambda x{:}?.\ x\ \ x)\ (\lambda y{:}?.y) : ?}$$

$$\frac{\begin{array}{l}\vdash (\lambda x{:}\alpha \to \alpha.\ x) : (\alpha \to \alpha) \to (\alpha \to \alpha) \\ \vdash\ \lambda y{:}\ \alpha.y : \alpha \to \alpha\end{array}}{\vdash (\lambda x{:}\ (\alpha \to \alpha).\ x)\ (\lambda y{:}\ \alpha.y) : \alpha \to \alpha}$$

Discussion

These examples illustrate two common issues with type systems:

- Code duplication may be required to type programs
  - Each version of the identity used at a different type requires a separate function definition
  - Historical example: Pascal
- The programmer may be required to write in lots of types
  - At least variable declarations are required for type checking

Type Inference

Idea

- Instead of the programmer writing in the types, have an algorithm infer the needed types automatically.
- Obviously results in less typing and less cluttered code
- Less obviously, makes it easier to reuse code
- Type inference is becoming more common

Type Rules

$$\frac{}{A, x{:}\ \alpha_x \vdash x : \alpha_x} \text{[Var]}$$

$$\frac{A, x{:}\ \alpha_x \vdash e : t}{A \vdash \lambda x{:}\ \alpha_x . e : \alpha_x \rightarrow t} \text{[Abs]}$$

$$\frac{t = t' \rightarrow \beta \quad A \vdash e_1 : t \quad A \vdash e_2 : t'}{A \vdash e_1\, e_2 : \beta} \text{[App]}$$

Discussion

- Every place a type is required, a fresh type variable is used
  - Stands for some definite, but unknown type

- At function applications, an equation captures what must be true of the types for the program to type check
  - The expression in function position must have a function type
  - The function domain and the function argument must have the same type

- Two steps to constructing a valid typing (or showing none exists)
  - Solve the equations
  - Substitute the solution back into the type derivation to obtain a valid proof

Solving the Constraints

Apply the following rewrite rules until no new constraints can be added

$S, t = \alpha \quad \Rightarrow \quad S, t = \alpha, \alpha = t$ [Symmetry]

$S, \alpha = t_1, \alpha = t_2 \quad \Rightarrow \quad S, \alpha = t_1, \alpha = t_2, t_1 = t_2$ [Transitivity]

$S, t_1 \rightarrow t_2 = t_3 \rightarrow t_4 \quad \Rightarrow \quad S, t_1 \rightarrow t_2 = t_3 \rightarrow t_4, t_1 = t_3, t_2 = t_4$ [Structure]

Solutions

When no new constraints can be added, the constraints are *saturated*

- If $x \to y = int$ is in the saturated constraints, then there is no solution, and the program has a type error.
- If $x = \ldots \to \ldots x \ldots$ or $x = \ldots x \ldots \to \ldots$ is implied by the saturated constraints, then there are no finite solutions
  - Treat the constraints as an undirected graph of equalities, if x occurs inside a $\to$ reachable from x
  - Example: $x = x \to x$
  - An *occurs check*
- If there are finite solutions, then they can be obtained by back substitution

Example

$$
\dfrac{
\dfrac{z : \alpha \to \beta \to \alpha \vdash z : \alpha \to \beta \to \alpha}{\vdash \lambda z : \alpha \to \beta \to \alpha\, .\, z : (\alpha \to \beta \to \alpha) \to (\alpha \to \beta \to \alpha)}
\qquad
\dfrac{\dfrac{x : \alpha,\ y : \beta \vdash x : \alpha}{x : \alpha \vdash \lambda y : \beta.\ x : \beta \to \alpha}}{\vdash \lambda x : \alpha.\ \lambda y : \beta.\ x : \alpha \to \beta \to \alpha}
}{
\vdash (\lambda z : \alpha \to \beta \to \alpha\, .\, z)\ (\lambda x : \alpha.\ \lambda y : \beta.\ x : \alpha \to \beta \to \alpha) : \alpha \to \beta \to \alpha
}
$$

Example

$$\dfrac{\dfrac{z: \alpha_z \vdash z: \alpha_z}{\vdash \lambda z: \alpha_z .\ z : \alpha_z \to \alpha_z} \qquad \dfrac{\dfrac{x: \alpha_x, y: \alpha_y \vdash x : \alpha_x}{x: \alpha_x \vdash \lambda y: \alpha_y .\ x : \alpha_y \to \alpha_x}}{\vdash \lambda x: \alpha_x .\ \lambda y: \alpha_y .\ x : \alpha_x \to \alpha_y \to \alpha_x} \qquad \alpha_z \to \alpha_z = (\alpha_x \to \alpha_y \to \alpha_x) \to \beta}{\vdash (\lambda z: \alpha_z .\ z)\ (\lambda x: \alpha_x .\ \lambda y: \alpha_y .\ x : \alpha_x) : \beta}$$

Solving …

$\alpha_z \to \alpha_z = (\alpha_x \to \alpha_y \to \alpha_x) \to \beta$

$\alpha_z = \alpha_x \to \alpha_y \to \alpha_x$

$\alpha_z = \beta$

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \alpha_z$

Convert the Solution to a Substitution

$\alpha_z \to \alpha_z = (\alpha_x \to \alpha_y \to \alpha_x) \to \beta$

$\alpha_z = \alpha_x \to \alpha_y \to \alpha_x$

$\alpha_z = \beta$

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \alpha_z$

Repeat in any order:

Pick any equation between variables $\alpha = \beta$ in the solution. Replace one variable by the other in the solution *and in the type derivation*.

Drop any equations $\alpha = \alpha$

Drop any repeated equations

Drop any equations without a variable on the lhs

Example

$$\dfrac{\dfrac{z: \alpha_z \vdash z: \alpha_z}{\vdash \lambda z{:}\, \alpha_z.\ z : \alpha_z \to \alpha_z} \qquad \dfrac{\dfrac{x: \alpha_x,\ y: \alpha_y \vdash x : \alpha_x}{x: \alpha_x \vdash \lambda y{:}\, \alpha_y.\ x : \alpha_y \to \alpha_x}}{\vdash \lambda x{:}\, \alpha_x.\ \lambda y{:}\, \alpha_y.\ x : \alpha_x \to \alpha_y \to \alpha_x} \qquad \alpha_z \to \alpha_z = (\alpha_x \to \alpha_y \to \alpha_x) \to \beta}{\vdash (\lambda z{:}\, \alpha_z.\ z)\ (\lambda x{:}\, \alpha_x.\ \lambda y{:}\, \alpha_y.\ x) : \beta}$$

Convert the Solution to a Substitution

$\alpha_z \to \alpha_z = (\alpha_x \to \alpha_y \to \alpha_x) \to \beta$

$\alpha_z = \alpha_x \to \alpha_y \to \alpha_x$

$\alpha_z = \beta$

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \alpha_z$

Repeat in any order:

Pick any equation between variables $\alpha = \beta$ in the solution. Replace one variable by the other in the solution *and in the type derivation*.

Drop any equations $\alpha = \alpha$

Drop any repeated equations

Drop any equations without a variable on the lhs

Convert the Solution to a Substitution

$\alpha_z = \alpha_x \to \alpha_y \to \alpha_x$

$\alpha_z = \beta$

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \alpha_z$

Repeat in any order:

Pick any equation between variables $\alpha = \beta$ in the solution. Replace one variable by the other in the solution *and in the type derivation*.

Drop any equations $\alpha = \alpha$

Drop any repeated equations

Drop any equations without a variable on the lhs

Convert the Solution to a Substitution

$\alpha_z = \alpha_x \to \alpha_y \to \alpha_x$

$\alpha_z = \beta$

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \alpha_z$

Repeat in any order:

Pick any equation between variables $\alpha = \beta$ in the solution. Replace one variable by the other in the solution *and in the type derivation*.

Drop any equations $\alpha = \alpha$

Drop any repeated equations

Drop any equations without a variable on the lhs

Convert the Solution to a Substitution

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \beta$

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \beta$

Repeat in any order:

Pick any equation between variables $\alpha = \beta$ in the solution. Replace one variable by the other in the solution *and in the type derivation*.

Drop any equations $\alpha = \alpha$

Drop any repeated equations

Drop any equations without a variable on the lhs

Example

$$\dfrac{\dfrac{z:\beta \vdash z:\beta}{\vdash \lambda z{:}\,\beta.\ z : \beta \to \beta} \qquad \dfrac{\dfrac{x:\alpha_x,\ y:\alpha_y \vdash x : \alpha_x}{x:\alpha_x \vdash \lambda y{:}\,\alpha_y.\ x : \alpha_y \to \alpha_x}}{\vdash \lambda x{:}\,\alpha_x.\ \lambda y{:}\,\alpha_y.\ x : \alpha_x \to \alpha_y \to \alpha_x}}{\vdash (\lambda z{:}\,\alpha_z.\ z)\ (\lambda x{:}\,\alpha_x.\ \lambda y{:}\,\alpha_y.\ x : \alpha_x) : \beta}$$

Convert the Solution to a Substitution

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \beta$

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \beta$

Repeat in any order:

Pick any equation between variables $\alpha = \beta$ in the solution. Replace one variable by the other in the solution *and in the type derivation*.

Drop any equations $\alpha = \alpha$

Drop any repeated equations

Drop any equations without a variable on the lhs

Convert the Solution to a Substitution

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

Repeat in any order:

Pick any equation between variables $\alpha = \beta$ in the solution. Replace one variable by the other in the solution *and in the type derivation*.

Drop any equations $\alpha = \alpha$

Drop any repeated equations

Drop any equations without a variable on the lhs

Convert the Solution to a Substitution

$\beta = \alpha_x \to \alpha_y \to \alpha_x$

Repeat in any order:

Pick any equation between variables $\alpha = \beta$ in the solution. Replace one variable by the other in the solution *and in the type derivation*.

Drop any equations $\alpha = \alpha$

Drop any repeated equations

Drop any equations without a variable on the lhs

Apply the Final Substitution to the Derivation

$$
\dfrac{
\dfrac{z: \alpha_x \to \alpha_y \to \alpha_x \vdash z: \alpha_x \to \alpha_y \to \alpha_x}{\vdash \lambda z: \alpha_x \to \alpha_y \to \alpha_x\ .\ z : (\alpha_x \to \alpha_y \to \alpha_x) \to (\alpha_x \to \alpha_y \to \alpha_x)}
\qquad
\dfrac{\dfrac{x: \alpha_x, y: \alpha_y \vdash x : \alpha_x}{x: \alpha_x \vdash \lambda y: \alpha_y.\ x : \alpha_y \to \alpha_x}}{\vdash \lambda x: \alpha_x.\ \lambda y: \alpha_y.\ x : \alpha_x \to \alpha_y \to \alpha_x}
}{
\vdash (\lambda z: \alpha_x \to \alpha_y \to \alpha_x.\ z)\ (\lambda x: \alpha_x.\ \lambda y: \alpha_y.\ x : \alpha_x) : \alpha_x \to \alpha_y \to \alpha_x
}
$$

Next Time ...

- More on types!