ソフトウェア論理
Logic in Computer Software

Yukiyoshi Kameyama

Dept. of Computer Science, U. of Tsukuba

Week 2

Summary of Last Week

- Introduction to a (computational) calculus based on lambda calculus
  - Syntax in the handout (last week)
  - Semantics in the slide: based on SOS (Structural Operational Semantics)

型システム: Type System

Observation: the syntax of the last-week's language was too coarse; we want to exclude BAD programs, for example:

- Non-function in the function position.

$$(10\ \texttt{true}) + (\texttt{false}\ 30)$$

- Testing non-boolean.

$$\texttt{if}\ 100\ \texttt{then}\ 20\ \texttt{else}\ 30$$

Question: how to detect such BAD programs before executing them (i.e. at the compile time) ?

Static Type System

To distinguish good programs from bad programs.

- If a program e is well typed, the execution of e never goes wrong. ("wrong" means a run-time mismatch of data, e.g., adding boolean and string. Similar to "Segmentation Fault" error in C.)

Other merits.

- Types improve readability, maintainability, modularity.
- Compilers can make use of type information to generate efficient codes.
- Types provide a means of abstraction.

Assigning Types to Programs

Examples:

$10 + 20 : \texttt{int}$

$\texttt{if true then } 10 \texttt{ else } 20 : \texttt{int}$

$\texttt{if true then } 10 \texttt{ else } \textit{true}$: Ill typed

$\lambda x.\ \texttt{if true then } x + 10 \texttt{ else } 20 : \texttt{int} \to \texttt{int}$

$\lambda (x, y).\ \texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : (\texttt{int}, \texttt{bool}) \to \texttt{int}$

$\lambda y.\ \lambda x.\ \texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{bool} \to (\texttt{int} \to \texttt{int})$

From now on, we will use Church-style presentation so that variables are explicitly annotated with their types when they are *bound*:

$\lambda (x : \texttt{int}).\ \texttt{if true then } x + 10 \texttt{ else } 20 : \texttt{int} \to \texttt{int}$

$\lambda (x : \texttt{int}, y : \texttt{bool}).\ \texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : (\texttt{int}, \texttt{bool}) \to \texttt{int}$

Assigning Types to Programs

More Examples:

$\texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{int}$

$\texttt{if } y = 30 \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{int}$

$(\texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20) + (\texttt{if } y = 30 \texttt{ then } x + 10 \texttt{ else } 20) : \texttt{int}$?

We need to "remember" the type of free variables, too.

Assigning Types to Programs

Types of free variables and a program:

$x : \texttt{int}, y : \texttt{bool} \vdash \texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{int}$

$x : \texttt{int}, y : \texttt{int} \vdash \texttt{if } y = 30 \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{int}$

Typing Judgment:

$$\Gamma \vdash e : \tau$$

where 、 Γ is $x_1 : \sigma_1, \cdots, x_n : \sigma_n$, e is a term (an expression), and τ is a type.

The Set of Types

The set of types depend on programming languages. In this language, we have:

$$\sigma, \tau ::= \texttt{int}, \texttt{bool}, \sigma \to \tau$$

Example of types:

$\texttt{int}$

$\texttt{bool}$

$\texttt{int} \to \texttt{int}$

$\texttt{int} \to (\texttt{bool} \to \texttt{int})$

$(\texttt{int} \to \texttt{int}) \to (\texttt{int} \to \texttt{int})$

Type System-1

$$\frac{}{\Gamma \vdash x : \sigma}\ \text{var},((x:\sigma) \in \Gamma)$$

$$\frac{}{\Gamma \vdash m : \texttt{int}}\ \text{const1},(m \text{ is an integer literal})$$

$$\frac{}{\Gamma \vdash b : \texttt{bool}}\ \text{const2},(b \text{ is } \texttt{true} \text{ or } \texttt{false})$$

Type System-2

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 + e_2 : \texttt{int}}\ \text{plus}$$

$$\frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \sigma \quad \Gamma \vdash e_3 : \sigma}{\Gamma \vdash \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \sigma}\ \text{if}$$

Type System-3

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda(x:\sigma).\ e : \sigma \to \tau}\ \text{fun}$$

$$\frac{\Gamma \vdash e : \sigma \to \tau \quad \Gamma \vdash f : \sigma}{\Gamma \vdash e\ f : \tau}\ \text{apply}$$

Type System-4

$$\frac{\Gamma, f : \sigma \to \tau, x : \sigma \vdash e : \tau}{\Gamma \vdash (\texttt{fix}\ (f:\sigma \to \tau)(x:\sigma).\ e) : \sigma \to \tau}\ \text{fix}$$

Note. $\texttt{fix}(f)(x).\ e$ is the function such that $f(x) = e$.

Example of Type Derivation

$$\dfrac{\dfrac{}{\cdot \vdash 0 : \mathtt{int}} \quad \dfrac{}{\cdot \vdash 1 : \mathtt{int}}}{\cdot \vdash 0 + 1 : \mathtt{int}}$$

Example of Type Derivation

Let $\Gamma = x : \mathtt{int}, y : \mathtt{bool}$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \mathtt{bool}} \quad \dfrac{}{\Gamma \vdash x : \mathtt{int}} \quad \dfrac{\dfrac{}{\Gamma \vdash x : \mathtt{int}} \quad \dfrac{}{\Gamma \vdash 1 : \mathtt{int}}}{\Gamma \vdash x + 1 : \mathtt{int}}}{\Gamma \vdash \mathtt{if}\ y\ \mathtt{then}\ x\ \mathtt{else}\ x + 1 : \mathtt{int}}}{y : \mathtt{bool} \vdash \lambda(x : \mathtt{int}).\ \mathtt{if}\ y\ \mathtt{then}\ x\ \mathtt{else}\ x + 1 : \mathtt{int} \to \mathtt{int}}}{\cdot \vdash \lambda(y : \mathtt{bool}).\ \lambda(x : \mathtt{int}).\ \mathtt{if}\ y\ \mathtt{then}\ x\ \mathtt{else}\ x + 1 : \mathtt{bool} \to (\mathtt{int} \to \mathtt{int})}$$

Example of Type Derivation

Let $\mathtt{BB} = \mathtt{bool} \to \mathtt{bool}$ and $\Delta = x : BB, y : \mathtt{bool}$,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta \vdash x : \mathtt{BB}} \quad \dfrac{\dfrac{}{\Delta \vdash x : \mathtt{BB}} \quad \dfrac{}{\Delta \vdash y : \mathtt{bool}}}{\Delta \vdash xy : \mathtt{bool}}}{\Delta \vdash x(xy) : \mathtt{bool}}}{x : \mathtt{BB} \vdash \lambda(y : \mathtt{bool}).\ x(xy) : \mathtt{BB}}}{\cdot \vdash \lambda(x : \mathtt{BB}).\ \lambda(y : \mathtt{bool}).\ x(xy)\ :\ \mathtt{BB} \to \mathtt{BB}}$$

Type System-5

$$\dfrac{\Gamma, x_1 : \sigma_1, \cdots, x_n : \sigma_n \vdash e : \tau}{\Gamma \vdash \lambda(x_1 : \sigma_1, \cdots, x_n : \sigma_n).\ e : (\sigma_1, \ldots, \sigma_n) \to \tau}\ \text{fun'}$$

$$\dfrac{\Gamma \vdash e : (\sigma_1, \ldots, \sigma_n) \to \tau \quad \Gamma \vdash e_1 : \sigma_1 \quad \ldots \quad \Gamma \vdash e_n : \sigma_n}{\Gamma \vdash e\ (e_1, \ldots, e_n) : \tau}\ \text{app'}$$

Type System-6

$$\frac{\Gamma,\ f : (\ldots, \sigma_i, \ldots) \to \tau,\ \ldots, x_i : \sigma_i, \ldots \vdash e : \tau}{\Gamma \vdash \texttt{fix}\ (f : (\ldots, \sigma_i, \ldots) \to \tau)(\ldots, x_i : \sigma_i, \ldots).\ e\ :\ (\ldots, \sigma_i, \ldots) \to \tau}\ \text{fix}$$

Note. $\texttt{fix}\ (f : \sigma)(\ldots, x_i : \tau_i, \ldots).\ e$ is a function such that $f(\ldots, x_i, \ldots) = e$.

Exercises

- Define the typing rules for $e - f$ and $e = f$ appropriately. (For $e = f$, you have many choices.)
- Write a type derivation for $x : \texttt{int} \to \texttt{int}, y : \texttt{int} \vdash x(x\ y)) : \texttt{int}$.
- Find a term e such that the following judgment is derivable for *any* σ,τ and ρ:

$$\vdash e\ :\ (\sigma \to \tau) \to ((\tau \to \rho) \to (\sigma \to \rho))$$

- Write a type derivation for the following function:

$$\texttt{fix}\ (f : \sigma \to \tau)(x : \sigma).\ \texttt{if}\ \texttt{gt}(x, 100)\ \texttt{then}\ x - 10\ \texttt{else}\ f(f(x + 11))$$

- (optional) Is there any type derivation for $x : \sigma \vdash x\ x : \tau$ for some σ and τ ?

型システムの健全性 (Type Soundness)

Strong type soundness means subject reduction AND progress.

Theorem (Subject Reduction)

If $\Gamma \vdash e : \sigma$ *is derivable, and* $e \rightsquigarrow^* e'$, *then* $\Gamma \vdash e' : \sigma$ *is derivable, where* $\rightsquigarrow^*$ *means computation.*

Theorem (Progress)

If $\Gamma \vdash e : \sigma$ *is derivable, and* e *is not (yet) a value, then there exists a* e' *such that* $e \rightsquigarrow e'$.

Both need rigid (mathematical) proofs.

Optional Homework: Try to write a proof of subject reduction for this language. Figure out what kind of auxiliary lemma you need.

Benefits of Type System

If strong type soundness holds, then well-typed programs do not go wrong [Hindley and Milner].

- Once a program typechecks (before execution), then
- it does not cause a type error during its computation, and
- it does not get stuck.

Note. Type soundness does not guarantee the program will terminate, or the program does not cause a run-time error such as "division by zero".

Type Checking and Type Inference

Type checking (e.g. in C and Java):

- Input: A typing context Γ, a term e, and a type τ.
- Output: Yes or No ($\Gamma \vdash e : \tau$ is derivable or not)

Type inference (e.g. in ML family and Haskell):

- Input: e
- Output:
  - 1: Γ and τ (such that $\Gamma \vdash e : \tau$ is derivable)
  - 2: Fail
- Question: but how to output infinitely many answers?

Principal Typing

Problem: Infer the type of $\lambda(.\ x : \sigma).x$.
Solution: $\texttt{int} \to \texttt{int}$, $\texttt{bool} \to \texttt{bool}$, $(\texttt{int} \to \texttt{bool}) \to (\texttt{int} \to \texttt{bool})$, $(\texttt{int} \to \texttt{int}) \to (\texttt{int} \to \texttt{int}), \cdots$

We must represent an infinitely many solutions in a finitistic way.

- A typing of a term e is (Γ, σ) such that $\Gamma \vdash e : \sigma$ is derivable.
- Principal typing of e is a typing of e such that any other typing of e can be obtained by it (using substitution and weakening).

Theorem (Principal Typing)

If e is typable, its pricipal typing exists.

Summary

Types play an essential role in most modern programming languages.

- Type system guarantees a certain static property,
- by ruling out syntactically bad (ill-typed) programs.
- The "type soundness" property is the key.

Type systems in programming languages.

- Core: simply typed lambda calculus (as in this lecture).
- Widely used: record, variant, polymorphic type, types for objects
- Research: dependent types, higher-order types