MTAT.05.105 Type Theory

Simply typed λ-calculus

Type systems

- Types are used to specify properties of a program.
  - Usually, types specify classes of normal forms.
  - If a term has a type Nat, and it has a normal form, the normal form is a natural number.
- Type systems contain:
  - A set of types.
  - A set of programs.
  - A type judgment.

Simple types

- Assume a countable set of type variables (base types).
- Simple types = no polymorhism.
- Syntax of simple types:

$$\begin{array}{rcll} \tau & ::= & \alpha & \text{variable} \\ & | & \tau_1 \to \tau_2 & \text{function type} \end{array}$$

- Type constructor $\to$ is right associative.

$$\tau_1 \to \tau_2 \to \tau_3 \equiv \tau_1 \to (\tau_2 \to \tau_3)$$

Explicit vs. implicit typing

- There are two main ways of typing λ-expressions.
- Church-style: give domain type explicitly as typed λ-term $\lambda x : \tau.\ e$.
  - Example: $\lambda x : \mathrm{Nat}.\ x + x : \mathrm{Nat} \to \mathrm{Nat}$.
  - Every term has a unique type.
- Curry-style: keep untyped syntax and use types as well-formedness predicate (type inference).
  - Example: $\lambda x.\ x + x : \mathrm{Nat} \to \mathrm{Nat}$.
  - Terms may have multiple types (eg. $\lambda x.\ x : \mathrm{Nat} \to \mathrm{Nat}$, but also $\lambda x.\ x : \mathrm{Bool} \to \mathrm{Bool}$).
- We will mostly use the first way.

Terms, notation, reduction

- Syntax of terms:

$$
\begin{array}{rcll}
e & ::= & x & \text{variable} \\
 & | & e_1\, e_2 & \text{application} \\
 & | & \lambda x : \tau.\, e & \text{abstraction}
\end{array}
$$

- Same syntactic conventions as in pure λ-calculus.
- Substitution, reduction, etc are also defined as in pure λ-calculus by ignoring typing annotations.

Typing relation

- To determine what terms have which types, we need to define a typing judgment.
- The basic judgment is a sequent $\Gamma \vdash e : \tau$
  - *"the term e has type τ in context Γ"*.
- Γ is a type assignement $\Gamma = \{x_1 : \tau_1, \ \ldots, \ x_n : \tau_n\}$.
  - Domain of Γ is denoted by $\mathrm{dom}(\Gamma) = \{x_1, \ \ldots, \ x_n\}$.
  - Must have: $\mathrm{FV}(e) \subseteq \mathrm{dom}(\Gamma)$.
- $\Gamma \leq \Delta$ if $x \in \mathrm{dom}(\Gamma)$ implies $x \in \mathrm{dom}(\Delta)$ and $\Gamma(x) = \Delta(x)$.
- A program (closed term) e has type τ if it has that type in the empty context $\vdash e : \tau$.

Typing relation

- Typing judgments are defined by inference rules.
- Typing rules for simply typed λ-calculus:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ var \qquad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x : \sigma.\, e : \sigma \to \tau}\ abs$$

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \quad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\, e_2 : \tau}\ app$$

  where $x \notin \mathrm{dom}(\Gamma)$.

- Further on we refer to the simply typed λ-calculus as $\lambda{\to}$.

Typing derivation

Typing derivation example (1):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau,\ y{:}\sigma\} \vdash x : \tau}\ var}{\{x{:}\tau\} \vdash \lambda y{:}\sigma.\, x : \sigma \to \tau}\ abs}{\vdash \lambda x{:}\tau,\, y{:}\sigma.\, x : \tau \to \sigma \to \tau}\ abs$$

Typing derivation

Typing derivation example (1):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau,\ y{:}\sigma\} \vdash x : \tau}\ var}{\{x{:}\tau\} \vdash \lambda y{:}\sigma.\, x : \sigma \to \tau}\ abs}{\vdash \lambda x{:}\tau,\, y{:}\sigma.\, x : \tau \to \sigma \to \tau}\ abs$$

Typing derivation

Typing derivation example (1):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau,\ y{:}\sigma\} \vdash x : \tau}\ var}{\{x{:}\tau\} \vdash \lambda y{:}\sigma.\, x : \sigma \to \tau}\ abs}{\vdash \lambda x{:}\tau,\, y{:}\sigma.\, x : \tau \to \sigma \to \tau}\ abs$$

Typing derivation

Typing derivation example (1):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau,\ y{:}\sigma\} \vdash x : \tau}\, var}{\{x{:}\tau\} \vdash \lambda y{:}\sigma.\, x : \sigma \to \tau}\, abs}{\vdash \lambda x{:}\tau,\, y{:}\sigma.\, x : \tau \to \sigma \to \tau}\, abs$$

Typing derivation

Typing derivation example (2):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash x : \sigma \to \tau \qquad \dfrac{\Gamma \vdash y : \rho \to \sigma \quad \Gamma \vdash z : \rho}{\Gamma \vdash yz : \sigma}}{\Gamma \vdash x(yz) : \tau}}{\Gamma_2 \vdash \lambda z{:}\rho.\, x(yz) : \rho \to \tau}}{\Gamma_1 \vdash \lambda y{:}\rho \to \sigma,\, z{:}\rho.\, x(yz) : (\rho \to \sigma) \to \rho \to \tau}}{\vdash \lambda x{:}\sigma \to \tau,\, y{:}\rho \to \sigma,\, z{:}\rho.\, x(yz) : (\sigma \to \tau) \to (\rho \to \sigma) \to \rho \to \tau}$$

where

$$\begin{aligned} \Gamma_1 &= \{x{:}\sigma \to \tau\} \\ \Gamma_2 &= \{x{:}\sigma \to \tau,\ y{:}\rho \to \sigma\} \\ \Gamma &= \{x{:}\sigma \to \tau,\ y{:}\rho \to \sigma,\ z{:}\rho\} \end{aligned}$$

Typing derivation

Typing derivation example (3):

$$\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x : \tau \to \rho}\ var \qquad \dfrac{}{\{x{:}\tau\} \vdash x : \tau}\ var}{\dfrac{\{x{:}\tau\} \vdash x\,x : \rho}{\vdash \lambda x{:}\tau.\, x\,x : \tau \to \rho}\ abs}\ app$$

Typing derivation

Typing derivation example (3):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x : \tau \to \rho}\ var \qquad \dfrac{}{\{x{:}\tau\} \vdash x : \tau}\ var}{\{x{:}\tau\} \vdash x\, x : \rho}\ app}{\vdash \lambda x{:}\tau.\, x\, x : \tau \to \rho}\ abs$$

Typing derivation

Typing derivation example (3):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x : \sigma \to \rho}\ var \qquad \dfrac{}{\{x{:}\tau\} \vdash x : \sigma}\ var}{\{x{:}\tau\} \vdash x\, x : \rho}\ app}{\vdash \lambda x{:}\tau.\, x\, x : \tau \to \rho}\ abs$$

Typing derivation

Typing derivation example (3):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x : \tau \to \rho}\ var \qquad \dfrac{}{\{x{:}\tau\} \vdash x : \tau}\ var}{\{x{:}\tau\} \vdash x\,x : \rho}\ app}{\vdash \lambda x{:}\tau.\, x\,x : \tau \to \rho}\ abs$$

Typing derivation

Typing derivation example (3):

$$\dfrac{\dfrac{\dfrac{}{\{x{:}\tau\} \vdash x : \tau \to \rho}\, var \qquad \dfrac{}{\{x{:}\tau\} \vdash x : \tau}\, var}{\{x{:}\tau\} \vdash x\, x : \rho}\, app}{\vdash \lambda x{:}\tau.\, x\, x : \tau \to \rho}\, abs$$

Typing derivation fails!

Type safety

- Type safety (soundness):

  Well typed programs do not "go wrong".

- The price:
  - Some sensible programs are rejected.
- Consists of two parts:
  - Progress: A well-typed term is not stuck (either it's a value or it can take a step according to the evaluation rules).
  - Preservation: If a well-typed term takes a step of evaluation, then the resulting term is also well-typed.

Uniqueness of typing

- Lemma (Generation lemma):

$$\begin{array}{lcl} \Gamma \vdash x : \tau & \Rightarrow & \Gamma(x) = \tau \\ \Gamma \vdash e_1\, e_2 : \tau & \Rightarrow & \exists \sigma\, [\Gamma \vdash e_1 : \sigma \to \tau \ \wedge \ \Gamma \vdash e_2 : \sigma] \\ \Gamma \vdash \lambda x{:}\sigma.e : \tau & \Rightarrow & \exists \rho\, [\Gamma, x : \sigma \vdash e : \rho \ \wedge \ \tau = \sigma \to \rho] \end{array}$$

- Theorem (Unique typing): If $\Gamma \vdash e : \tau_1$ and $\Gamma \vdash e : \tau_2$, then $\tau_1 = \tau_2$.
  - Proof: By induction with respect to the structure of e and using Generation lemma.
- Note: Unique typing fails for many richer languages.

Preservation of typing

- Lemma (Weakening): If $\Gamma \vdash e : \tau$ and $\Gamma \leq \Delta$, then $\Delta \vdash e : \tau$.
  - Proof: By induction on the typing derivation.
- Lemma (Substitution): If $\Gamma, x{:}\sigma \vdash e_1 : \tau$ and $\Gamma \vdash e_2 : \sigma$, then $\Gamma \vdash e_1[x \mapsto e_2] : \tau$.
  - Proof: By induction with respect to the structure of e_1 and using Weakening lemma.
- Theorem (Subject reduction): If $\Gamma \vdash e : \tau$ and $e \twoheadrightarrow_\beta e'$, then $\Gamma \vdash e' : \tau$.
  - Proof: By induction with respect to the definition of $\twoheadrightarrow_\beta$ and using Substitution lemma.

Progress

- Lemma (Progress): If $\Gamma \vdash e : \sigma$, then $\exists e' : \sigma . e \rightarrow_\beta e'$ or $e \in \mathrm{Val}$, where

$$v \in \mathrm{Val} \ ::= \ x\, v \ldots v \mid \lambda x{:}\tau . v$$

  - Proof: By induction with respect to the structure of e.

Strong normalization

- Theorem (Strong normalization): In $\lambda\!\rightarrow$, every β-reduction sequence is finite.
- Some simple corollaries:
  - The question of equality of terms in $\lambda\!\rightarrow$ is solvable.
  - Fixpoint operators are not definable in $\lambda\!\rightarrow$.

Extensions: booleans

- Types: $\tau ::= \ldots \mid \mathsf{Bool}$
- Terms: $e ::= \ldots \mid \mathtt{true} \mid \mathtt{false} \mid \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2$
- Values: $v ::= \ldots \mid \mathtt{true} \mid \mathtt{false}$
- Typing rules:

$$\frac{}{\Gamma \vdash \mathtt{true} : \mathsf{Bool}} \qquad \frac{}{\Gamma \vdash \mathtt{false} : \mathsf{Bool}}$$

$$\frac{\Gamma \vdash e_0 : \mathsf{Bool} \quad \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \mathbf{if}\, e_0\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 : \tau}$$

- Evaluation rules:

$$\begin{array}{lcl} \mathbf{if}\, \mathtt{true}\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 & \rightarrow & e_1 \\ \mathbf{if}\, \mathtt{false}\, \mathbf{then}\, e_1\, \mathbf{else}\, e_2 & \rightarrow & e_2 \end{array}$$

Extensions: unit type

- Types: $\tau ::= \ldots \mid \mathsf{Unit}$
- Terms: $e ::= \ldots \mid ()$
- Values: $v ::= \ldots \mid ()$
- Typing rules:

$$\frac{}{\Gamma \vdash () : \mathsf{Unit}}$$

- No evaluation rules!

Extensions: products

- Types: $\tau ::= \ldots \mid \tau_1 \times \tau_2$
- Terms: $e ::= \ldots \mid (e_1, e_2) \mid \mathtt{fst} \mid \mathtt{snd}$
- Values: $v ::= \ldots \mid (v_1, v_2)$
- Typing rules:

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{fst}\, e : \tau_1} \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathtt{snd}\, e : \tau_2}$$

- Evaluation rules:

$$\begin{array}{lcl} \mathtt{fst}\,(e_1, e_2) & \to & e_1 \\ \mathtt{snd}\,(e_1, e_2) & \to & e_2 \end{array}$$

Extensions: sums

- Types: $\tau ::= \ldots \mid \tau_1 + \tau_2$
- Terms: $e ::= \ldots \mid \mathtt{inl} \mid \mathtt{inr} \mid \mathbf{case}\,(e_0;\, x_1.e_1;\, x_2.e_2)$
- Values: $v ::= \ldots \mid \mathtt{inl}\, v_1 \mid \mathtt{inl}\, v_2$
- Typing rules:

$$\frac{\Gamma \vdash e : \tau_1}{\Gamma \vdash \mathtt{inl}\, e : \tau_1 + \tau_2} \qquad \frac{\Gamma \vdash e : \tau_2}{\Gamma \vdash \mathtt{inr}\, e : \tau_1 + \tau_2}$$

$$\frac{\Gamma \vdash e_0 : \tau_1 + \tau_2 \quad \Gamma, x_1 : \tau_1 \vdash e_1 : \sigma \quad \Gamma, x_2 : \tau_2 \vdash e_2 : \sigma}{\Gamma \vdash \mathbf{case}\,(e_0;\, x_1.e_1;\, x_2.e_2) : \sigma}$$

- Evaluation rules:

$$\begin{aligned} \mathbf{case}\,(\mathtt{inl}\, e_0;\, x_1.e_1;\, x_2.e_2) &\rightarrow e_1[x_1 \mapsto e_0] \\ \mathbf{case}\,(\mathtt{inr}\, e_0;\, x_1.e_1;\, x_2.e_2) &\rightarrow e_2[x_2 \mapsto e_0] \end{aligned}$$

- Note: In the case of sums unique typing fails!