Instructors: Kazutaka Matsuda and Eijiro Sumii

Typed λ-Calculus

Definition (λ-terms with Sums and Products)**.** The set of *terms* is defined by the following BNF.

$$\begin{array}{rcl} M, N & ::= & x \mid M\ N \mid \lambda x.M \\ & \mid & (M, N) \mid \pi_1 M \mid \pi_2 M \\ & \mid & \mathsf{InL}\ M \mid \mathsf{InR}\ M \mid \mathbf{case}\ M\ \mathbf{of}\ (x.N_1)\ (y.N_2) \end{array}$$ □

Intuitively, (M, N) makes the pair of M and N, $\pi_1 M$ extracts the first component of the pair M, and $\pi_2 M$ extracts the second component. Expressions $\mathsf{InL}\ M$ and $\mathsf{InR}\ N$ are injections: $\mathsf{InL}\ M$ assign the tag InL to M and $\mathsf{InR}\ M$ assign the tag InR to M. These tags are used in the case-analysis performed by $\mathbf{case}\ M\ \mathbf{of}\ (x.N_1)\ (y.N_2)$: if M is tagged left as $\mathsf{InL}\ M'$, then it is reduced to $N_1[M'/x]$, and if M is tagged right as $\mathsf{InR}\ M'$, the it is reduced to $N_2[M'/x]$.

Formally, we have additional reduction rules

$$\frac{}{\pi_1\ (M, N) \longrightarrow M} \quad \frac{}{\pi_2\ (M, N) \longrightarrow N}$$

$$\frac{}{\mathbf{case}\ (\mathsf{InL}\ M)\ \mathbf{of}\ (x.N_1)\ (y.N_2) \longrightarrow N_1[M/x]} \quad \frac{}{\mathbf{case}\ (\mathsf{InR}\ M)\ \mathbf{of}\ (x.N_1)\ (y.N_2) \longrightarrow N_2[M/y]}$$

along with the rules to reduce subterms.

$$\frac{M \longrightarrow M'}{(M, N) \longrightarrow (M', N)} \quad \frac{N \longrightarrow N'}{(M, N) \longrightarrow (M, N')} \quad \frac{M \longrightarrow M'}{\pi_1 M \longrightarrow \pi_1 M'} \quad \frac{M \longrightarrow M'}{\pi_2 M \longrightarrow \pi_2 M'}$$

$$\frac{M \longrightarrow M'}{\mathsf{InL}\ M \longrightarrow \mathsf{InL}\ M'} \quad \frac{M \longrightarrow M'}{\mathsf{InR}\ M \longrightarrow \mathsf{InR}\ M'} \quad \frac{M \longrightarrow M'}{\mathbf{case}\ M\ \mathbf{of}\ (x.N_1)\ (y.N_2) \longrightarrow \mathbf{case}\ M'\ \mathbf{of}\ (x.N_1)\ (y.N_2)}$$

$$\frac{N_1 \longrightarrow N_1'}{\mathbf{case}\ M\ \mathbf{of}\ (x.N_1)\ (y.N_2) \longrightarrow \mathbf{case}\ M\ \mathbf{of}\ (x.N_1')\ (y.N_2)}$$

$$\frac{N_2 \longrightarrow N_2'}{\mathbf{case}\ M\ \mathbf{of}\ (x.N_1)\ (y.N_2) \longrightarrow \mathbf{case}\ M\ \mathbf{of}\ (x.N_1)\ (y.N_2')}$$

There are terms, such as $\pi_1\ (\lambda x.x)$ and $((\lambda x.x), (\lambda y.y))\ (\lambda z.z)$, that are in normal form but appear intuitively meaningless. We formalize "meaningful" normal form as *values* below (mutually defined with the set of *neutral terms*).

$$\begin{array}{l} V ::= \lambda x.V \mid (V_1, V_2) \mid \mathsf{InL}\ V \mid \mathsf{InR}\ V \mid W \\ W ::= x \mid W\ V \mid \pi_1 W \mid \pi_2 W \mid \mathbf{case}\ W\ \mathbf{of}\ (x.V_1)\ (y.V_2) \end{array}$$

We call a term *stuck* if it is in normal form but not a value. Accordingly, we say that a term M *gets stuck* if $M \longrightarrow^* M'$ for some stuck term M'.

Goal

Find a way to tell that a term will not get stuck before trying to reduce it.

Why we have pairs and sums explicitly? One reason is to introduce clearly-meaningless terms like $\pi_1\ (\lambda x.x)$ with no "meaningful" way to evaluate them. Recall that everything is a function in the untyped λ-calculus. The other reason is that simple types discussed below are not powerful enough to type Church-encoded data.

Simple Types

The idea is to classify terms by which kind of values they evaluates to. For example, if we know that $\lambda x.x$ evaluates to a function, we know that $\pi_1\ (\lambda x.x)$ is meaningless because it tries to extract the first component of a function (this is clearly impossible).

Definition. The set of *(simple) types* is defined as follows.

$$\tau ::= B \mid \tau_1 \times \tau_2 \mid \tau_1 + \tau_2 \mid \tau_1 \to \tau_2 \qquad \square$$

Here, B represents a *base type* such as *Int* or *Bool*, $\tau_1 \times \tau_2$ represents the *product type* of τ_1 and τ_2, $\tau_1 + \tau_2$ represents the *sum type* of τ_1 and τ_2, and $\tau_1 \to \tau_2$ represents the *function type* from τ_1 to τ_2. Very roughly speaking, a term belongs to the type $\tau_1 \times \tau_2$ will be reduced to a pair whose first and second components belong to τ_1 and τ_2 respectively, and a term belongs to the type $\tau_1 + \tau_2$ will be reduced to a term that is either injected left from a term in τ_1 or injected right from a term in τ_2.

Now we define how to give a term a type . A *type environment* is a mapping from variables to types, which is used to assign types to free variables in a term. A *typing judgment* $\Gamma \vdash M : \tau$, which is read that under typing environment Γ term M has type τ, is defined by the following typing rules.

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}\ \textsc{T-Var} \qquad \frac{\Gamma \vdash M : \tau' \to \tau \quad \Gamma \vdash N : \tau'}{\Gamma \vdash M\ N : \tau}\ \textsc{T-App} \qquad \frac{\Gamma \uplus \{x \mapsto \tau_1\} \vdash M : \tau_2}{\Gamma \vdash \lambda x.M : \tau_1 \to \tau_2}\ \textsc{T-Abs}$$

$$\frac{\Gamma \vdash M : \tau_1 \quad \Gamma \vdash N : \tau_2}{\Gamma \vdash (M, N) : \tau_1 \times \tau_2}\ \textsc{T-Pair} \qquad \frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \pi_1 M : \tau_1}\ \textsc{T-Fst} \qquad \frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \pi_2 M : \tau_2}\ \textsc{T-Snd}$$

$$\frac{\Gamma \vdash M : \tau_1}{\Gamma \vdash \mathsf{InL}\ M : \tau_1 + \tau_2}\ \textsc{T-Left} \qquad \frac{\Gamma \vdash M : \tau_2}{\Gamma \vdash \mathsf{InR}\ M : \tau_1 + \tau_2}\ \textsc{T-Right}$$

$$\frac{\Gamma \vdash M : \tau_1 + \tau_2 \quad \Gamma \uplus \{x \mapsto \tau_1\} \vdash N_1 : \tau' \quad \Gamma \uplus \{y \mapsto \tau_2\} \vdash N_2 : \tau'}{\Gamma \vdash \textbf{case}\ M\ \textbf{of}\ (x.N_1)\ (y.N_2) : \tau'}\ \textsc{T-Case}$$

Above, we name each inference rule for convenience. Here, $\uplus$ represents disjoint union. We assumed that a term M of $\Gamma \vdash M : \tau$ is appropriately α-renamed so that every $\Gamma \uplus \{\dots\}$ above is defined. A term M is called *well-typed* (under Γ) if $\Gamma \vdash M : \tau$ holds for some τ, and otherwise it is called *ill-typed*. Notice that $\emptyset \vdash M : \tau$ implies that M is closed. For this set of the inference rules, which rule should be applied to a term M is uniquely determined by the form of M. The set of rules satisfying this condition is sometimes called *syntax-directed*. An example of a well-typed term is $\lambda x.(x, x)$, which has the following derivation tree for any type τ.

$$\cfrac{\cfrac{\cfrac{}{\{x \mapsto \tau\} \vdash x : \tau} \quad \cfrac{}{\{x \mapsto \tau\} \vdash x : \tau}}{\{x \mapsto \tau\} \vdash (x, x) : \tau \times \tau}}{\emptyset \vdash \lambda x.(x.x) : \tau \to \tau \times \tau}$$

An example of an ill-typed term is $\pi_1\ (\lambda x.x)$.

We state that well-typed closed normal forms are values.

Theorem ((An Equivalent form of) Progress)**.** For a term M, if $\emptyset \vdash M : \tau$ for some τ and M is in a normal form, M is a value.

Proof. Induction on the typing derivation of $\emptyset \vdash M : \tau$. □

Type Safety

Type safety is a statement something like "well-typed programs do not go wrong". Here, since we are interested in whether a term will get stuck or not, the type safety for our case is that "well-typed programs do not get stuck". This property is usually proved by proving the two properties:

- *Subject reduction (or, preservation)* is a statement that reductions preserve types. Thus, well-typed terms are reduced to well-typed terms.
- *Progress* is a statement that a well-typed term is not stuck, i.e., either a value or reducible. In other words, well-typed normal forms are values, which already we have proved.

Having the two properties, we can prove the type safety by a simple induction.

In advance to stating the subject reduction property, we introduce an important lemma below.

Lemma (Substitution Lemma)**.** Let M and N be terms. If $\Gamma \uplus \{x \mapsto \tau\} \vdash M : \tau'$ and $\Gamma \vdash N : \tau$ for some Γ, τ and τ' then, $\Gamma \vdash M[N/x] : \tau'$ holds.

Proof. Induction on the derivation of $\Gamma \uplus \{x \mapsto \tau\} \vdash M : \tau'$. □

We are now ready to prove the subject reduction.

Theorem (Subject Reduction)**.** Let M be a term such that $\Gamma \vdash M : \tau$ for some Γ and τ. If $M \longrightarrow M'$, then $\Gamma \vdash M' : \tau$ holds.

Proof. Induction on the derivation of $M \longrightarrow M'$. We use the substitution lemma when substitution occurs. □

Theorem (Type Safety)**.** For a term M such that $\emptyset \vdash M : \tau$ for some τ, if $M \longrightarrow^* M'$ for some M', M' is not stuck.

Proof. By the subject reduction property and by the induction on $M \longrightarrow^* M'$, we can prove that $\Gamma \vdash M' : \tau$ holds. Then, by the progress property, M' is not stuck. □

Other Important Properties

Theorem (Decidability of Type Checking)**.** Given a type environment Γ, a term M and a type τ, checking whether $\Gamma \vdash M : \tau$ holds or not is decidable. □

Theorem (Strong Normalization)**.** For a well-typed term M, there is no infinite sequence of $M \longrightarrow M' \longrightarrow M'' \longrightarrow \cdots$. □

In other words, every well-typed term has a normal form. This also means that the simply-typed λ-calculus is not Turing complete.