λ-Calculus

Simple Types and their Extensions

陳亮廷 Chen, Liang-Ting

Formosan Summer School on Logic, Language, and Computation 2024

Institute of Information Science
Academia Sinica

Simply Typed λ-Calculus: Introduction

While λ-calculus is expressive and computationally powerful, it is rather painful to write programs inside λ-calculus.

Function can be applied to an arbitrary term which can represent a Boolean value, a number, or even a function, so as a programming language it is not easy to see the intention of a program.

Therefore, we will consider a formal definition of a typing judgement

$$\Gamma \vdash t : A$$

which specifies the type A of a term t under a list of free (typed) variables, allowing us to *restrict the formation* of a valid term by typing.

Simply Typed λ-Calculus: Statics

Assume $\mathbb{V}$ is a set of type variables different from variables in untyped λ-terms. (And suppress its existence from now on.)

Definition 1

The judgement $A : \mathsf{Type}$ is defined inductively as follows.

$$\frac{}{X : \mathsf{Type}} \text{ if } X \in \mathbb{V} \qquad\qquad \frac{A : \mathsf{Type} \qquad B : \mathsf{Type}}{A \to B : \mathsf{Type}}$$

where $A \to B$ represents a function type from A to B.

We say that A is a type if $A : \mathsf{Type}$ is derivable.

The function type is higher-order, because

1. functions can be arguments of another function;
2. functions can be the result of a computation.

For example,

$(A_1 \to A_2) \to B$ a function type whose argument is of type $A_1 \to A_2$;

$A_1 \to (A_2 \to B)$ a function whose return type is $A_2 \to B$.

Following the convention of function application, we introduce the convention for the function type:

Convention

$$A_1 \to A_2 \to \ldots A_n \quad := \quad A_1 \to (A_2 \to (\ldots \to (A_{n-1} \to A_n) \ldots))$$

Definition 2

A *typing context* Γ is a sequence

$$\Gamma \equiv x_1 : A_1,\ x_2 : A_2,\ \ldots,\ x_n : A_n$$

of *distinct variables* x_i of type A_i.

Definition 3

The membership judgement $\Gamma \ni (x : A)$ is defined inductively:

$$\frac{}{\Gamma, x : A \ni x : A} \qquad \frac{\Gamma \ni x : A}{\Gamma, y : B \ni x : A}$$

We say that x of type A occurs in Γ if $\Gamma \ni (x : A)$ if derivable.

The implicit typing system for simply typed λ-calculus is defined by the following typing rules, i.e. inference rules with its conclusion a typing judgement:

$$\frac{}{\Gamma \vdash_i x : A} \text{(var)} \quad \text{if } \Gamma \ni (x : A)$$

$$\frac{\Gamma, x : A \vdash_i t : B}{\Gamma \vdash_i \lambda x.\, t : A \to B} \text{(abs)}$$

$$\frac{\Gamma \vdash_i t : A \to B \qquad \Gamma \vdash_i u : A}{\Gamma \vdash_i t\, u : B} \text{(app)}$$

We say that t is a closed term if $\vdash t : A$ is derivable.

N.B. Whether a term t has a typing derivation is a *property* of t.

A typing system is *syntax-directed* if it has *exactly* one typing rule for each term construct.

By being syntax-directed, every typing derivation can be inverted:

Lemma 4 (Typing inversion)

Suppose that $\Gamma \vdash_i t : A$ *is derivable. Then,*

$t \equiv x$ *implies* $x : A$ *occurs in* Γ.

$t \equiv \lambda x.\, t'$ *implies* $A = B \to C$ *and* $\Gamma, x : B \vdash_i u' : C$.

$t \equiv u\ v$ *implies there is some* A *such that* $\Gamma \vdash_i u : A \to B$ *and* $\Gamma \vdash_i v : B$.

This lemma is particularly useful when constructing a typing derivation by hand.

For any types A and B, the judgement $\vdash_i \lambda x\,y.\,x : A \to B \to A$ has a derivation

$$\dfrac{\dfrac{\dfrac{}{x : A, y : B \vdash_i x : A}\,\text{(var)}}{x : A \vdash_i \lambda y.\,x : B \to A}\,\text{(abs)}}{\vdash_i \lambda x\,y.\,x : A \to B \to A}\,\text{(abs)}$$

Therefore, $\lambda x\,y.\,x$ is a program of type $A \to B \to A$.

Derive the typing judgement

$$\vdash_i \lambda f\, g\, x.\, f\, x\, (g\, x) : (A \to B \to C) \to (A \to B) \to A \to C$$

for every types A, B and C.

Type inference and checking

Can we answer the following questions algorithmically?

Type inference Given a context Γ and a term t, is there a type $?$ such that the typing judgement $\Gamma \vdash t : ?$ is derivable?

Type checking Given a context Γ, a type A, and a term t, is the typing judgement $\Gamma \vdash t : A$ derivable?

Typability is reducible to type checking problem of

$$x_0 : A \vdash \texttt{fst}\, x_0\, t : A$$

Theorem 5

Type checking is decidable in simply typed λ-calculus.

Programming in Simply Typed λ-Calculus

The type of natural numbers is of the form

$$\mathtt{nat}_A := (A \to A) \to A \to A$$

for every type A.

Church numerals

$$\begin{aligned} \mathbf{c}_n &:= \lambda f\, x.\, f^n x \\ \vdash \mathbf{c}_n &: \mathtt{nat}_A \end{aligned}$$

Successor

$$\begin{aligned} \mathtt{suc} &:= \lambda n\, f\, x\, .\ f\ (n\ f\ x) \\ \vdash \mathtt{suc} &: \mathtt{nat}_A \to \mathtt{nat}_A \end{aligned}$$

Addition

$$\texttt{add} := \lambda n\, m\, f\, x.\, (m\ f)\ (n\ f\ x)$$
$$\vdash \texttt{add} : \texttt{nat}_A \to \texttt{nat}_A \to \texttt{nat}_A$$

Muliplication

$$\texttt{mul} := \lambda n\, m\, f\, x.\, (m\ (n\ f))\ x$$
$$\vdash \texttt{mul} : \texttt{nat}_A \to \texttt{nat}_A \to \texttt{nat}_A$$

Conditional

$$\texttt{ifz} := \lambda n\, x\, y.\, n\ (\lambda z.\, x)\ y$$
$$\vdash \texttt{ifz} : ?$$

The type of `ifz` may not be as obvious as you may expect. Try to find one and justify your guess.

We can also define the type of Boolean values for each type variable as

$$\mathtt{bool}_A := A \to A \to A$$

Boolean values

$$\mathtt{true} := \lambda x\, y.\, x \quad \text{and} \quad \mathtt{false} := \lambda x\, y.\, y$$

Conditional

$$\mathtt{cond} := \lambda b\, x\, y.\, b\, x\, y$$

$$\vdash \mathtt{cond} : \mathtt{bool}_A \to A \to A \to A$$

Exercise

1. Define conjunction `and`, disjunction `or`, and negation `not` in simply typed λ-calculus.
2. Prove that `and`, `or`, and `not` are well-typed.

Properties of Simply Typed λ-Calculus

Type safety = Preservation + Progress

> *"Well-typed programs cannot 'go wrong'."*
>
> —*(Milner, 1978)*

Preservation If $\Gamma \vdash t : A$ is derivable and $t \longrightarrow_\beta u$, then $\Gamma \vdash u : A$.

Progress If $\Gamma \vdash t : A$ is derivable, then either t is in *normal form* or there is u with $t \longrightarrow_\beta u$.

By combing the above two properties, we can extend the progress theorem to $\twoheadrightarrow_\beta$: if $\Gamma \vdash t : A$ then $t \twoheadrightarrow_\beta u$ for some $\Gamma \vdash u : A$ which is either reducible or in normal form.

The converse of preservation might not hold.

Lemma 6 (Typability of subterms)

Let t be a term with $\Gamma \vdash t : A$ derivable. Then, for every subterm t' of t there exists Γ' such that

$$\Gamma' \vdash t' : A'.$$

Recall that

1. $\mathbf{K}_1 = \lambda x\, y.\, x$
2. $\Omega = (\lambda x.\, x\, x)\, (\lambda x.\, x\, x)$

and $\mathbf{K}_1\ (\lambda x.\, x)\ \Omega \twoheadrightarrow_\beta \mathbf{I}$.

Ω is not typable, so $\mathbf{K}_1\, \mathbf{I}\, \Omega$ is not typable.

Weakening If $\Gamma \vdash t : A$ and $x \notin \Gamma$, then $\Gamma, x : B \vdash t : A$.

Substitution If $\Gamma, x : A \vdash t : B$ and $\Gamma \vdash u : A$ then $\Gamma \vdash t[u/x] : B$.

Corollary 7 (Variable renaming)

If $\Gamma, x : A \vdash t : B$ *and* $y \notin \mathrm{dom}(\Gamma)$, *then* $\Gamma, y : A \vdash t[y/x] : B$ *where* $\mathrm{dom}(\Gamma)$ *denotes the set of variables which occur in* Γ.

Theorem 8

For any t *and* u *if* $\Gamma \vdash t : A$ *is derivable and* $t \longrightarrow_\beta u$, *then* $\Gamma \vdash u : A$.

Proof sketch.

By induction on both the derivation of $\Gamma \vdash t : A$ and $t \longrightarrow_\beta u$. □

N.B. The only non-trivial case is $\Gamma \vdash (\lambda x.\, t)\; u : B$ which needs the above results.

Proof.

By induction on both the derivation of $\Gamma \vdash t : A$ and $t \longrightarrow_\beta u$.

1. Suppose $\Gamma \vdash x : A$. However, $x \not\longrightarrow_\beta u$ for any u. Therefore, it is vacuously true that $\Gamma \vdash u : A$.
2. Suppose $\Gamma \vdash \lambda x.\, t : A \to B$ and $\lambda x.\, t \longrightarrow_\beta u$. Then, u must be $\lambda x.\, u'$ for some u'; $\Gamma, x : A \vdash t : B$ and $t \longrightarrow_\beta u'$ must be derivable. By induction hypothesis, $\Gamma, x : A \vdash u'$ is derivable, so is $\Gamma \vdash \lambda x.\, u' : A \to B$.
3. Suppose $\Gamma \vdash t\ u$. Then ...
4. ...

$\square$

Theorem 9

If $\Gamma \vdash t : A$ is derivable, then t is in normal form or there is u with $t \longrightarrow_\beta u$.

To prove the theorem, we would like to use induction on $\Gamma \vdash t : A$ again.

However, the fact that t is in normal form does not tell us much what t is. Can we characterise t syntactically?

The notion of normal form can be characterised syntactically:

Definition 10

Define judgements $\mathtt{Neutral}\ t$ and $\mathtt{Normal}\ u$ mutually by

$$\frac{}{\mathtt{Neutral}\ x} \qquad \frac{\mathtt{Neutral}\ t}{\mathtt{Normal}\ t}$$

$$\frac{\mathtt{Neutral}\ t \qquad \mathtt{Normal}\ u}{\mathtt{Neutral}\ t\ u} \qquad \frac{\mathtt{Normal}\ u}{\mathtt{Normal}\ \lambda x.\, u}$$

Idea. $\mathtt{Neutral}\ u$ and $\mathtt{Normal}\ t$ are derivable iff

$$t \equiv x\ u_1 \cdots u_n \quad \text{and} \quad u \equiv \lambda x_1 \cdots x_n .\, x\ u_1 \cdots u_m .$$

That is, β-redex cannot exist in u if u is normal.

A term t has no β-reduction if and only if t is normal:

Lemma 11

Soundness *If* `Normal` t *(resp.* `Neutral` t*) is derivable, then* t *is in normal form.*

Completeness *If* t *is in normal form, then* `Normal` t *is derivable.*

Proof sketch.

Soundness By mutual induction on the derivation of `Normal` t and `Neutral` t.

Completeness By induction on the formation of t.

□

Theorem 12

If $\Gamma \vdash t : A$ is derivable, then $\texttt{Normal}\ t$ or there is u with $t \longrightarrow_\beta u$.

Proof sketch.

By induction on the derivation of $\Gamma \vdash t : A$. $\square$

The statement is trivial in classical logic, as a direct consequence of the Law of Excluded Middle.

Yet, the progress theorem can be proved constructively without LEM. What is the *computational meaning* of this theorem?

Definition 13

t is *weakly normalising* denoted by $t \downarrow$ if

$$\frac{\texttt{Normal}\ t}{t \downarrow} \qquad\qquad \frac{t \longrightarrow_\beta u \qquad u \downarrow}{t \downarrow}$$

That is, t is weakly normalising if there is a sequence

$$t \longrightarrow_\beta t_1 \longrightarrow_\beta t_2 \longrightarrow_\beta \ldots u \not\longrightarrow_\beta$$

Theorem 14 (Weak normalisation)

Every term t with $\Gamma \vdash t : A$ is weakly normalising.

Definition 15

t is *strongly normalising* denoted by $t \Downarrow$ if

$$\frac{\forall u.\, (t \longrightarrow_\beta u \implies u \Downarrow)}{t \Downarrow}$$

Intuitively, *strong normalisation* says every sequence

$$t \longrightarrow_\beta t_1 \longrightarrow_\beta t_2 \cdots$$

terminates, but the definition builds the sequence backwards.

Theorem 16

Every term t with $\Gamma \vdash t : A$ is strongly normalising.

Extensions to Simply Typed λ-Calculus

Self-applicative term cannot be typed in simply typed λ-calculus. E.g.,

$$\lambda x.\, x\, x$$

cannot be typed, since $A \to A$ is not equal to A. Hence, the Y-combinator in untyped λ-calculus cannot be typed.

A construct is introduced explicitly for general recursion:

Let $\Lambda_{\mathtt{fix}}(V)$ be the set of terms defined with an additional construct:

fixpoint $\mathtt{fix}\, f.\, t$ is a term in $\Lambda_{\mathtt{fix}}(V)$, if $t \in \Lambda_{\mathtt{fix}}(V)$ and $f \in V$

An additional typing rule is added to simply typed λ-calculus:

$$\frac{\Gamma, f : A \vdash_i t : A}{\Gamma \vdash_i \mathtt{fix}\, f.\, t : A}$$

β-reduction for the general recursion $\texttt{fix}$ is extended with the relation

$$\texttt{fix}\, x.\, t \longrightarrow_\beta t[\texttt{fix}\, x.\, t/x]$$

A term which never terminates can be defined easily.

$$\begin{aligned} &\texttt{fix}\, x.\, x && \longrightarrow_\beta x[\texttt{fix}\, x.\, x/x] \\ \equiv\ &\texttt{fix}\, x.\, x && \longrightarrow_\beta x[\texttt{fix}\, x.\, x/x] \\ \equiv\ &\texttt{fix}\, x.\, x && \longrightarrow_\beta x[\texttt{fix}\, x.\, x/x] \\ \equiv\ &\ldots \end{aligned}$$

Other notions such as $=_\alpha, \twoheadrightarrow_\beta$, and $\mathbf{FV}$ are extended similarly.

While Church numerals can have multiple types $\mathtt{nat}_A$, for any A, we extend the calculus with a single type of natural numbers instead.

Let $\Lambda_{\mathtt{fix},\mathsf{N}}(V)$ be the set of terms defined with additional constructs:

- $\mathtt{zero}$ is a term in $\Lambda_{\mathtt{fix},\mathsf{N}}(V)$
- $\mathtt{suc}(t)$ is a term in $\Lambda_{\mathtt{fix},\mathsf{N}}(V)$ if t is
- $\mathtt{ifz}(t; x.\, u)$ is a term in $\Lambda_{\mathtt{fix},\mathsf{N}}(V)$ if $t, u \in \Lambda_{\mathtt{fix},\mathsf{N}}(V)$ and $x \in V$

with additional typing rules

$$\frac{}{\Gamma \vdash \mathtt{zero} : \mathbb{N}} \qquad \frac{\Gamma \vdash t : \mathbb{N}}{\Gamma \vdash \mathtt{suc}(t) : \mathbb{N}}$$

$$\frac{\Gamma \vdash v : \mathbb{N} \qquad \Gamma \vdash t : A \qquad \Gamma, x : \mathbb{N} \vdash u : A}{\Gamma \vdash \mathtt{ifz}(t; x.\, u)\; v : A}$$

The third rule is akin to pattern matching on natural numbers.

β-reduction for natural numbers is extended with two rules:

$$\mathtt{ifz}(t; x.\, u)\ \mathtt{zero} \longrightarrow_\beta t$$
$$\mathtt{ifz}(t; x.\, u)\ \mathtt{suc}(n) \longrightarrow_\beta u[n/x]$$

Define the predecessor of natural numbers as a program

$$\texttt{pred} : \mathbb{N} \to \mathbb{N}.$$

Evaluate the following terms to their normal forms.

1. $\texttt{pred}\ \texttt{zero}$
2. $\texttt{pred}\ (\texttt{suc}\ (\texttt{suc}\ (\texttt{suc}\ \texttt{zero})))$

Boolean values: Exercise

Extend simply typed λ-calculus $\Lambda_{\mathtt{fix},\mathsf{N}}(V)$ further with a type of Boolean values.

1. What term constructs are needed?
2. What typing rules should be added?
3. How β-reduction should be updated?
4. Define Boolean operations, i.e. conjunction, disjunction, and negation, in this extension.

1. (2.5%) Complete the proof of the Preservation Theorem.
2. (5%) Show the Progress Theorem.
3. (2.5%) Show that if t is in normal form then $\texttt{Normal}\ t$ is derivable.
4. (5%) Extend $\Lambda_{\texttt{fix},\mathsf{N}}(V)$ further with product types $A \times B$, for any A and B where additional constructs should include pairs (t, u) and a construct to pattern match on a pair.