Naruseuych Mykyta

Herman Geuvers: Introduction to type theory

Plan

- Introduction
- Types
- Terms
- Reductions
- Derivation rules
- Curry-Howard correspondence

Introduction
Types vs. sets

Set theory (tacitly = ZFC) plays foundational role.

Motto: Everything is a set (or, at least, can be thought of as a set).

Benefits: simple ontology.

Problems: allows to ask "nonsensical" questions, i.e.

$$\sqrt{2} \in \pi \ ?$$

Deeper problems: description of a set does not provide an easy way to decide, whether an element is it's member, i.e.

$$3 \in \{n \in \mathbb{N} : \forall x, y, z \in \mathbb{N}^{+} (x^n + y^n \neq z^n)\}.$$

Types vs. sets (cont.)

In general, the relation $\in$ is <u>undecidable</u> (by Turing machine)

Type theories use the relation $:$, where $a : B$ should be read as

<u>the object a is of type B</u>

There are many different type theories.

We will focus on the one(s) where $:$ is decidable (or even "effectively decidable").

Types can be thought of as collections of elements of similar intrinsic nature.

So then questions like $\sqrt{2} \in \pi$? are not allowed, since $\sqrt{2}$ and π are of the same type.

Simple type theory
$\lambda \to$

$a : B$

Terms

- for each type σ we have a countable set of typed variables: $x^{\sigma}_1, x^{\sigma}_2, \ldots$

- if $M : \sigma \to \tau$ and $N : \sigma$, then we can form a new term MN of type τ, which is called an application

- if $P : \tau$, then we can form $(\lambda x^{\sigma}. P)$ of type $\sigma \to \tau$, which is called abstraction

Types

- countable set of letters $\alpha, \beta, \gamma, \ldots$ - type variables

- if σ and τ are types, then we can form a type $\sigma \to \tau$, can be thought of as a type of functions with input of type σ and output of type τ

Currying

We don't have a way of constructing types like $\sigma \times T$, and consequently types like $\sigma \times T \to \Delta$ ... functions of several variables.

But we can still construct types like $\sigma \to (T \to \Delta)$.

Function $F : \sigma \to (T \to \Delta)$ can be thought of as a function of two variables, one of type σ, the other of type T.

Convention: $\sigma \to T \to \Delta$ is $\sigma \to (T \to \Delta)$.

Note that $(\sigma \to T) \to \Delta$ can be thought of as a functional

Examples

1. $(\alpha \to \beta) \to \alpha$ ... functional taking in functions of type $\alpha \to \beta$ and returning objects of type α.

2. $(\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma = (\alpha \to \beta) \to \big((\beta \to \gamma) \to (\alpha \to \gamma)\big)$ ...

functional with two inputs: first of type $\alpha \to \beta$, second of type $\beta \to \gamma$; which returns functions of type $\alpha \to \gamma$.

Or it can be thought of as a functional with three inputs.

Terms

Application $(M : \sigma \to \tau, N : \sigma, MN : \tau)$

Can be thought of as an application of function M to an object N to get an object MN of type τ.

Convention: $MNP = (MN)P$

Abstraction $(P : \tau, (\lambda x^{\sigma}.P) : \sigma \to \tau)$

Can be thought of as making a function which takes an object of type σ and plugs it into the term P on the places of x^{σ}, i.e. substitutes for x^{σ}.

Convention: type annotations on variables will be written only at the λ-abstraction, i.e.

write $\lambda x^{\sigma}.x$ instead of $\lambda x^{\sigma}.x^{\sigma}$

Examples

Identity combinator on σ: $I_\sigma := \lambda x^\sigma . x : \sigma \to \sigma$

K combinator on σ and τ: $K_{\sigma\tau} := \lambda x^\sigma . \lambda y^\tau . x : \sigma \to \tau \to \sigma$

$\lambda x^{\alpha\to\beta} . \lambda y^{\beta\to\gamma} . \lambda z^\alpha . y(xz) : (\alpha\to\beta) \to (\beta\to\gamma) \to \alpha \to \gamma$

$\lambda x^\alpha . \lambda y^{(\beta\to\alpha)\to\alpha} . y(\lambda z^\beta . x) : \alpha \to ((\beta\to\alpha)\to\alpha)\to\alpha$

Type checking

How to check the validity of $\lambda x^{\alpha\to\beta}.\lambda y^{\beta\to\gamma}.\lambda z^{\alpha}.\, y(xz)$ being of type $(\alpha\to\beta)\to(\beta\to\gamma)\to\alpha\to\gamma$?

First make the bracketing and then build up the type from the innermost term:

$$\left(\lambda x^{\alpha\to\beta}.\left(\lambda y^{\beta\to\gamma}.\left(\lambda z^{\alpha}.\, y(xz)\right)\right)\right)$$

- $x:\alpha\to\beta,\ y:\beta\to\gamma,\ z:\alpha$ implies $xz:\beta$ and $y(xz):\gamma$
- $\lambda z^{\alpha}.\, y(xz)$ is of type $\alpha\to\gamma$
- $\lambda y^{\beta\to\gamma}.(\lambda z^{\alpha}.\, y(xz))$ is of type $(\beta\to\gamma)\to\alpha\to\gamma$
- $\lambda x^{\alpha\to\beta}.(\lambda y^{\beta\to\gamma}.(\lambda z^{\alpha}.\, y(xz)))$ is of type $(\alpha\to\beta)\to(\beta\to\gamma)\to\alpha\to\gamma$

Convention: $\lambda x^{\sigma}.\lambda y^{\tau} M$ is the same as $\lambda x^{\sigma} y^{\tau}.M$

Free and bound variables, α-conversion

Variable x is <u>free</u>, if it does not appear in the scope of λx^{σ}.
Otherwise x is <u>bound</u> (by λx^{σ}).

<u>Ex</u>: $(\lambda x^{\sigma}.\, x)\, y$ ... x is bound, while y is free.

<u>α-conversion</u> is the renaming of the bound variable by a "fresh" one,
i.e. the one which does not appear in the term as
free or bound.

We write $M \equiv N$ or $M =_{\alpha} N$ if M can be made into N (or vice versa)
by a sequence of α-conversions.

Terms with all variables bound are called <u>combinators</u>

Examples

$\lambda x^{\sigma}.\ \lambda y^{\tau}.\ x \equiv \lambda x^{\sigma}.\ \lambda z^{\tau}.\ x \equiv \lambda y^{\sigma}.\ \lambda z^{\tau}.\ y \equiv \lambda y^{\sigma}.\ \lambda x^{\tau}.\ y$

Also note $\lambda x^{\sigma}.\ \lambda x^{\tau}.\ x \equiv \lambda x^{\sigma}.\ \lambda y^{\tau}.\ y$, since in first term x is tacitly assumed to be of type τ.

There are also situations, where we wish to rename not by a completely fresh variable, but at least by the one not in the scope.

So the concept of α-conversion (=renaming) should be taken in a slightly informal way.

We will also try to avoid situations as in the second example.

Barendregt convention: all free variables of the term are different from the bound ones and all bound variables are distinct.

β-reduction

We want to introduce rules for rewriting terms.
We have already seen α-conversion and, in fact, all terms should be taken up to $\equiv$-equivalence.
But we will mostly focus on particular terms rather than equivalence classes so that calculations will be simpler.

Denote $M[x := N]$ as a term constructed from M by substituting N for a free variable x.

Definition: <u>one-step β reduction</u> $\longrightarrow_\beta$ is defined as:

- $(\lambda x^\sigma . M)N \longrightarrow_\beta M[x := N]$
- $M \longrightarrow_\beta N$ implies $MP \longrightarrow_\beta NP$, also $PM \longrightarrow_\beta PN$
- $M \longrightarrow_\beta N$ implies $\lambda x^\sigma . M \longrightarrow_\beta \lambda x^\sigma . N$

β-reduction (cont.)

Multi-step β reduction $\twoheadrightarrow_\beta$ is a transitive closure of $\rightarrow_\beta$

β-equality $=_\beta$ is a transitive reflexive symmetric closure of $\rightarrow_\beta$

Convention: before applying one-step β reduction we rename variables inside functions so as to stick to Barendregt convention.

Ex: $(\lambda x^{\sigma}.\lambda y^{\tau}.x)y =_\alpha (\lambda x^{\sigma}.\lambda z^{\tau}.x)y \rightarrow_\beta \lambda z^{\tau}.y$

while without α-conversion

$(\lambda x^{\sigma}.\lambda y^{\tau}.x)y \rightarrow_\beta \lambda y^{\tau}.y$. where y is of type σ

Numerals

The type $(\sigma \to \sigma) \to \sigma \to \sigma$ is called the type of numerals over σ, $\mathrm{nat}\,\sigma$

Natural numbers are then represented as closed terms of type $\mathrm{nat}\,\sigma$

$$C_n := \lambda f^{\sigma \to \sigma}.\, \lambda x^{\sigma}.\, f^n(x),$$

$f^n(x)$ denotes $\underbrace{f(\ldots f(f\,x))}_{n\text{-times}}$

these are also called Church numerals.

They can be thought of as operators taking functions and returning iterations of given functions.

We will also omit denoting the dependency of C_n on type σ.

Examples

We want to show that applying numerals to identity does not change it.

Recall: $\overline{I}_\sigma = \lambda x^\sigma . x$

$$C_2\,\overline{I}_\sigma = \left(\lambda f^{\sigma\to\sigma} . \lambda x^\sigma . f(fx)\right)\left(\lambda x^\sigma . x\right) =_\alpha \dots$$

$$\dots =_\alpha \left(\lambda f^{\sigma\to\sigma} . \lambda z^\sigma . f(fz)\right)\left(\lambda x^\sigma . x\right) \longrightarrow_\beta \dots$$

$$\dots \longrightarrow_\beta \lambda z^\sigma . \left(\lambda x^\sigma . x\left(\lambda x^\sigma . x\ z\right)\right) \longrightarrow_\beta \lambda z^\sigma . \left(\lambda x^\sigma . x\ z\right) \longrightarrow_\beta \dots$$

$$\dots \longrightarrow_\beta \lambda z^\sigma . z =_\alpha \lambda x^\sigma . x = \overline{I}_\sigma$$

Examples

We call a term <u>redex</u>, if it can be β-reduced.

Note that, in general, reductions are non-deterministic, since we are able to choose a term to reduce at each step.

$$S := \lambda x^{\sigma\to\sigma\to\sigma}.\,\lambda y^{\sigma\to\sigma}.\,\lambda z^{\sigma}.\,xz(yz) : (\sigma\to\sigma\to\sigma)\to(\sigma\to\sigma)\to\sigma\to\sigma$$

Then $SK_{\sigma\sigma}\overline{I}_{\sigma} : \sigma\to\sigma$ and

$$SK_{\sigma\sigma}\overline{I}_{\sigma} \longrightarrow_{\beta} (\lambda y^{\sigma\to\sigma}.\,\lambda z^{\sigma}.\,K_{\sigma\sigma}z(yz))\,\overline{I}_{\sigma}$$

We can then proceed in several ways, either consider the whole term as a redex and push $\overline{I}_{\sigma}$ inside, or reduce the subterm $K_{\sigma\sigma}z$.

We will show both ways.

Call by Value vs. Call by Name

Consider $K_{\sigma\sigma} z$ as a redex. Then

$$(\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.K_{\sigma\sigma} z(yz))\,\underline{I}_{\sigma} \equiv (\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.(\lambda p^{\sigma} q^{\sigma}.p) z(yz))\,\underline{I}_{\sigma}$$

$$\downarrow_{\beta}$$

$$(\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.(\lambda q^{\sigma}.z)(yz))\,\underline{I}_{\sigma}$$

$$\downarrow_{\beta}$$

$$\lambda z^{\sigma}.(\lambda q^{\sigma}.z)(\underline{I}_{\sigma} z)$$

$$\downarrow_{\beta}$$

$$\lambda z^{\sigma}.(\lambda q^{\sigma}.z) z$$

$$\downarrow_{\beta}$$

$$\lambda z^{\sigma}.z \equiv \underline{I}_{\sigma}$$

Note that every β-reduction which has occured was of the form $(\lambda x.M)N$, where N was a value, i.e. an abstraction term or variable.

Such β-reductions are called Call by Value.

Call by Value vs. Call by Name (cont.)

Let us now consider the whole $(\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.K_{\sigma\sigma}z(yz))\underline{I}_{\sigma}$ as a redex.

$$(\lambda y^{\sigma\to\sigma}.\lambda z^{\sigma}.K_{\sigma\sigma}z(yz))\underline{I}_{\sigma} \longrightarrow_{\beta} \lambda z^{\sigma}.K_{\sigma\sigma}z(\underline{I}_{\sigma}z) \equiv \ldots$$

$$\ldots \equiv \lambda z^{\sigma}.(\lambda p^{\sigma}q^{\sigma}.p)(\underline{I}_{\sigma}z)$$

$$\downarrow_{\beta}$$

$$\lambda z^{\sigma}.(\lambda q^{\sigma}.z)(\underline{I}_{\sigma}z)$$

$$\downarrow_{\beta}$$

$$\lambda z^{\sigma}.\underline{I}_{\sigma}z$$

$$\downarrow$$

$$\lambda z^{\sigma}.z \equiv \underline{I}_{\sigma}$$

Note that this reduction is not call by value. Such reductions are called

call by name

(ordinary β-reductions)

Determinism

Allowing only call by value reductions we can reduce the number of guesses our reduction should make.
But it is still non-deterministic.

To make it deterministic, we can explicitly define reduction strategies, i.e. left-most outermost or right-most innermost.

It is also possible to reduce redexes simultaneously, which is called complete development.

It should be noted, that the term we'll get in the end does not depend on a strategy of reducing.

Important properties

We finally list important properties of the simple type theory. The proofs are rather technical but fairly obvious and so we omit them.

Theorem $\lambda\!\to$ enjoys the following properties:

- **Subject Reduction**

  If $M : \sigma$ and $M \longrightarrow_\beta P$, then $P : \sigma$

- **Church-Rosser**

  If M is a well-typed term and $M \longrightarrow\!\!\!\!\rightarrow_\beta P$ and $M \longrightarrow\!\!\!\!\rightarrow_\beta N$, then there is a well-typed term Q, s.t. $P \longrightarrow\!\!\!\!\rightarrow_\beta Q$ and $N \longrightarrow\!\!\!\!\rightarrow_\beta Q$

- **Strong Normalization**

  If M is a well-typed term, then there is no infinite β-reduction path starting from M.

Derivation rules

Our inductive definition of terms is given in the standard Church style.

It is also possible to present the inductive definition in rule form:

$$\frac{}{x^{\sigma} : \sigma} \qquad \frac{M : \sigma \to \tau \quad N : \sigma}{MN : \tau} \qquad \frac{P : \tau}{\lambda x^{\sigma}.\, P : \sigma \to \tau}$$

So now we have a <u>derivation tree</u>, which can be considered as a proof of the fact that the term has that type.

Derivation rules (cont.)

In the previous presentation we do not specify the set of free variables.
We can, in principle, compute them recursively, but this is considered to be imprecise.

So we will actually present rules with explicitly stated set of free variables, which is known as <u>contexts</u>.

Context Γ is defined as a set consisting of statements $x_1 : \sigma_1, x_2 : \sigma_2, \ldots, x_n : \sigma_n$, where x_i are distinct and σ_i are types.

We write $x \in \Gamma$ if it is among $x_1, \ldots, x_n$.

Derivations à la Church

Definition: derivation rules **à la Church** are as follows:

$$\frac{x:\sigma \in \Gamma}{\Gamma \vdash x:\sigma} \qquad \frac{\Gamma \vdash M:\sigma \to \tau \quad \Gamma \vdash N:\sigma}{\Gamma \vdash MN:\tau} \qquad \frac{\Gamma, x:\sigma \vdash P:\tau}{\Gamma \vdash \lambda x:\sigma.P:\sigma \to \tau}$$

We write $\Gamma \vdash_{\lambda\to} M:\sigma$ if there is a derivation using these rules with conclusion $\Gamma \vdash M:\sigma$.

Note that we write $\lambda x:\sigma.x$ instead of $\lambda x^{\sigma}.x$, which can be interpreted as assigning a type to an untyped variable.

Comparing definitions

Thm: if $\Gamma \vdash M : \sigma$ then $M : \sigma$ (in the original definition) and also free variables of M are among Γ.

If $M : \sigma$ (in the original definition), then $\Gamma \vdash M : \sigma$, where Γ consists exactly of declarations of all the free variables of M to their corresponding types.

Pf: obvious induction, also note that translating term à la Church to the original form utilizes an obvious "isomorphism" that lifts the types in the λ-abstraction to a superscript ∎

Example

Derivation of $\vdash K_{\sigma\tau} : \sigma \to \tau \to \sigma$

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

In general, derivations may be quite big, since we are constructing trees where many duplicates can occur.

There is a <u>Fitch style representation</u> of typing derivations, which we will omit.

Curry-Howard correspondence

Using the presentation of $\lambda\!\to$ with rules, we can state the Curry-Howard formulas-as-types correspondence.

The idea is that there are two reading of a judgement $M : \sigma$

1. term as algorithm/program, type as specification:

   M is a function of type σ

2. type as a proposition, term as its proof:

   M is a proof of the proposition σ

More precisely: there is a natural 1–1 correspondence between typable terms in $\lambda\!\to$ and derivations in minimal propositional logic.