Type Theory

Sixth Summer School on Formal Techniques

May 2016

Dr Stéphane Graham-Lengrand

`Stephane.Lengrand@Polytechnique.edu`

In brief

Type theory can be viewed as an area of Logic. . .

. . . that is concerned not only with the semantics of formulae (e.g. in $\{\mathsf{true}, \mathsf{false}\}$)

. . . but also with the semantics of proofs

Semantics of programs perhaps more natural than semantics of proofs

Type theory exploits this, based on the proofs-as-programs paradigm

(a.k.a. Curry-Howard correspondence)

Type theory is based on types as in a (functional) programming language

More precisely: based on typed λ-calculus

Contents

I. The λ-calculus and simple types

Refreshing your memory: the λ-calculus

Three constructs:

- variables, e.g. x, y, z
- applications, e.g. $t\ u$
- λ-abstractions, e.g. $\lambda x.t$

In other words:

$$t, u, v, \ldots ::= x \mid t\ u \mid \lambda x.t$$

What is $\lambda x.x$? What is $\lambda y.y$?

What is $\lambda x.\lambda y.x$? What is $\lambda x.\lambda y.y\ x$?

β-reduction: $(\lambda x.t)\ u \longrightarrow \{{}^{u}\!/\!{}_{x}\}t$

How many ways to reduce

$(\lambda x.\lambda x'.x')\ ((\lambda y.y)\ z)\ ((\lambda y.y)\ z')$?

$$\frac{}{\Delta, x : A \vdash x : A}$$

$$\frac{\Delta \vdash t : A \rightarrow B \qquad \Delta \vdash u : A}{\Delta \vdash t\ u : B}$$

$$\frac{\Delta, x : A \vdash t : B}{\Delta \vdash \lambda x.t : A \rightarrow B}$$

Where Δ is a typing context, and A and B are types ranging over

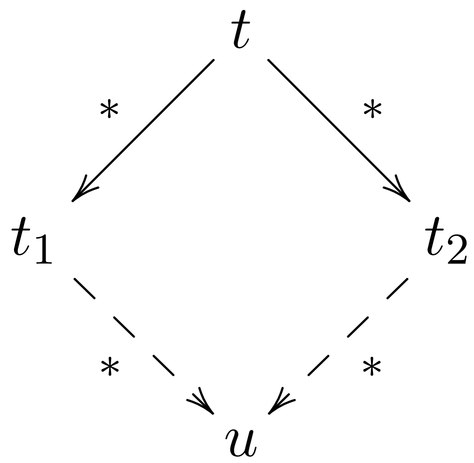
$$A, B, C, \ldots ::= a \mid A \rightarrow B$$

(a ranges over base types)

Confluence

Theorem: the relation $\longrightarrow$ is *confluent*, i.e.

If $t \longrightarrow^* t_1$ and $t \longrightarrow^* t_2$, then there exists u such that $t_1 \longrightarrow^* u$ and $t_2 \longrightarrow^* u$



$\longrightarrow^*$ is reflexive and transitive closure of $\longrightarrow$

$\longleftrightarrow^*$ is reflexive, transitive and symmetric closure of $\longrightarrow$

Proof: not today

Corollary: Irreducible forms are unique, i.e.

Given a λ-term t, there is at most one irreducible u such that $t \longrightarrow^* u$

Existence of u?

What about $\omega = (\lambda x.x\ x)\ (\lambda x.x\ x)$? What about $(\lambda x.y)\ \omega$?

Motivation for typing

Reason why some terms are non-normalising lies in the question of whether

applying a function to itself ($x\ x$) makes mathematical sense

(Issue very close to whether or not a predicate can apply to itself $P(P)$ or whether a set can belong to itself $y \in y$)

Has to do with controlling the **domain** of function x

In Set theory:

if $f : A \longrightarrow B$ and $x \in A$ then writing $f(x)$ "makes sense" and $f(x) \in B$

But we don't need Set theory for that:

Abstract away from sets to retain only the necessary ingredients for controlling domains and applications of functions to arguments

$$x \in A \qquad \text{becomes} \qquad x : A$$

This is the notion of *typing*

This is a purely syntactic notion

About typing

Are these λ-term typable?

$$\lambda x.\lambda y.y\ x$$

$$\lambda x.\lambda y.x\ (y\ x)$$

$$\lambda x.\lambda y.\lambda z.z\ (y\ x)\ (x\ y)$$

$$(\lambda x.x\ x)\ (\lambda x.x\ x)$$

Remark: If $\Delta \vdash t : A$ then FV(t) is included in the domain of Δ

Reduction preserves typing: If $\Delta \vdash t : A$ and $t \longrightarrow t'$ then $\Delta \vdash t' : A$

Proof: easy induction on the inductive property $t \longrightarrow t'$

Termination: If $\Delta \vdash t : A$ then all reduction paths starting from t are finite

Proof: not today

The λ-calculus

- models the core of functional programming languages
- is used in Higher-Order Logic (HOL) to construct predicates

(e.g. $\lambda x^a.\lambda y^a.\forall z^{a \to \mathsf{Prop}}(z\ x \Rightarrow z\ y)$)

In both cases, typing plays an important role.

A slogan (Milner 78): “Well-typed programs cannot go wrong”

In Higher-Order Logic, it prevents us from applying a predicate to itself, e.g. $P(P)$. This was (essentially) allowed in Frege’s foundation for mathematics, which Russell’s paradox showed inconsistent.

In type theory: typed λ-calculus also used to represent **proofs**

Connection with proofs

Let's look at fragment of Natural Deduction for implication only:

$$\frac{}{\Gamma, P \vdash P}$$

$$\frac{\Gamma \vdash P \Rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q}$$

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q}$$

Now let's look again at the typing rules for λ-calculus:

$$\frac{}{\Delta, x : A \vdash x : A}$$

$$\frac{\Delta \vdash t : A \to B \qquad \Delta \vdash u : A}{\Delta \vdash t\, u : B}$$

$$\frac{\Delta, x : A \vdash t : B}{\Delta \vdash \lambda x.t : A \to B}$$

What can we say?

More precisely

Propositions are Types

Proofs are Programs

Every proof tree can be annotated to be the typing tree of some λ-term

(λ-calculus variables annotate hypotheses, a λ-term annotates the conclusion)

Conversely:

Every typing tree, for some λ-term t, can be turned into a proof tree,

simply by hiding variables and λ-term annotations

It is the **Curry-Howard isomorphism**

Proving a given statement = finding inhabitant of a given type (proving = programming)

Exercise

Give a proof of

$\vdash P \Rightarrow P$

What is the λ-term annotating the proof?

Give a proof of

$\vdash P_1 \Rightarrow (P_2 \Rightarrow P_1)$

What is the λ-term annotating the proof?

Give a proof of

$\vdash (P_1 \Rightarrow (P_2 \Rightarrow P_3)) \Rightarrow (P_1 \Rightarrow P_2) \Rightarrow P_1 \Rightarrow P_3$

What is the λ-term annotating the proof?

Give a proof of

$\vdash ((P_1 \Rightarrow P_2) \Rightarrow P_1) \Rightarrow P_1$

What is the truth table for this formula?

II. Intuitionistic logic

The drinker's theorem

"There is always someone such that, if he drinks, everybody drinks"

$$\exists x(\textsc{drinks}(x) \Rightarrow \forall y\ \textsc{drinks}(y))$$

Proof - Informal

Take the first guy you see, call it Bob.

Either Bob does not drink,

in which case he satisfies the predicate “if he drinks, everybody drinks”

. . . or Bob drinks, in which case we have to check that everybody else drinks

If this is the case, then again Bob is the person we are looking for

If we find someone who does not drink, call it Frank,

we change our mind and say that the guy we are looking for is Frank

We can turn this into a formal proof of the formula $\exists x\ (\textsc{drinks}(x) \Rightarrow \forall y\ \textsc{drinks}(y))$ in predicate logic. . .

. . . using the rule

$$\frac{}{\Gamma \vdash P \vee \neg P} \text{ Law of Excluded Middle}$$

Problem with this

We have proved the theorem

. . . but we are still incapable of identifying the person satisfying the property

(or rather, our choice depends on the context)

In other words, we fail to provide a witness of existence

In other words, the logic we use does not have the witness property

The logic we use lacks a certain dose of constructivism

Lack of witness, another example

Predicate P: assuming $P(0)$, $\neg P(2)$

Can we prove that there is an integer x such that $P(x) \wedge \neg P(x+1)$?

$$P(0), \neg P(2) \vdash \exists x\ (P(x) \wedge \neg P(x+1))$$

Is there an integer n such that we can prove $P(n) \wedge \neg P(n+1)$?

$$P(0), \neg P(2) \vdash P(n) \wedge \neg P(n+1)$$

Concrete example:

Let $u_0 := \sqrt{2}$, $u_{x+1} := u_x^{\sqrt{2}}$, and $P(x)$ be "u_x irrational"

$P(0)$, $P(2)$?

Applying the above: There is x such that $P(x)$ and $\neg P(x+1)$

Therefore: There is an irrational r ($:= u_x$) such that $r^{\sqrt{2}}$ rational

r is either $\sqrt{2}$ or $\sqrt{2}^{\sqrt{2}}$, depending on whether $\sqrt{2}^{\sqrt{2}}$ is rational or not

The mismatch

Remark:

$\vdash P \wedge Q$ if and only if both $\vdash P$ and $\vdash Q$

The object-level $\wedge$ matches the meta-level "and"

$\vdash \forall x\, P[x]$ if and only if for all terms t we have $\vdash P[t]$ (t not necessarily closed)

The object-level $\forall$ matches the meta-level "for all"

Clearly:
If either $\vdash P$ or $\vdash Q$ then $\vdash P \vee Q$
If there is a term t such that $\vdash P[t]$ then $\vdash \exists x\, P[x]$

But	If you have...	...you don't necessarily have
	$\vdash \exists x\, P[x]$	an n such that $\vdash P[n]$
Example	$\vdash \exists x\, (P(x) \vee \neg P(x+1))$	an n such that $\vdash P(n) \vee \neg P(n+1)$
	$\vdash P \vee Q$	either $\vdash P$ or $\vdash Q$
Example	$\vdash P \vee \neg P$	either $\vdash P$ or $\vdash \neg P$ (e.g. P Goedel formula)

For $\vee$ and $\exists$, there is a mismatch between the object-level and the meta-level

The culprit and how to fix the mismatch

In all our examples, mismatch entirely due to:

- the Law of Excluded Middle ($P \vee \neg P$)
- or, equivalently, the Elimination of Double Negation ($(\neg\neg P) \Rightarrow P$).

The fix is easy: **Disallow** those laws

You get what is called **Intuitionistic Logic**(s) -as opposed to Classical logic(s)

Distinction can be done for propositional logic, predicate logic, higher-order logic, etc.

The claim: we recover a full match between object-level and meta-level

- $\vdash P \wedge Q$ if and only if both $\vdash P$ and $\vdash Q$
- $\vdash \forall x\ P[x]$ if and only if for all terms t we have $\vdash P[t]$
- $\vdash P \vee Q$ if and only if either $\vdash P$ or $\vdash Q$
- $\vdash \exists x\ P[x]$ if and only if there is a term t such that $\vdash P[t]$

The above match works in the empty theory, not in any theory!

(imagine the theory $P \vee Q$ or the theory $\exists x\ P$)

III. Computing with intuitionistic proofs

Constructivism

Seeking the witness property is part of a wider approach to mathematics called "constructivism".

Objects that mathematics speak about must be "constructed".

(Typical bad example: the set of all sets, from Russell's paradox)

If we can always speak about an object we have shown to exist,
then while showing its existence we must have constructed it.

Similarly, if we claim one of two things holds, we should be able to compute which one of the two holds.

Constructivism brought about a very computational view of what it is to do mathematics.

A computational interpretation of intuitionistic logic

Following Brouwer–Heyting–Kolmogorov, Kleene interpreted formulae as sets of realisers:

$r \Vdash P_1 \wedge P_2$	if $r = (r_1, r_2)$ with $r_1 \Vdash P_1$ and $r_2 \Vdash P_2$
$r \Vdash P_1 \vee P_2$	if $r = \mathsf{inj}_i(t')$ with $r' \Vdash P_i$ for $i = 0$ or $i = 1$
$r \Vdash P_1 \to P_2$	if r is a computable function such that, whenever $r' \Vdash P_1$, $r(r') \Vdash P_2$
$r \Vdash \exists x P(x)$	if $r = (a, r')$ with a an element of the "model" and $r' \Vdash P(a)$
$r \Vdash \forall x P(x)$	if r is a computable function such that, for all elements a of the "model", $r(a) \Vdash P(a)$

Parameterised by a way to interpret atomic formulae

r ranges over mathematical objects such as pairs, computable functions, etc

can be implemented as a number

(ok for pairs, injections, & computable functions can be assigned their Gödel numbers)

can be implemented as an untyped λ-term (untyped λ-calculus being Turing-complete)

there comes the Curry-Howard correspondence

Through the C-H correspondence, intuitionistic proofs provide realisers

Let's use

- α, β, etc. for the variables annotating hypotheses

  (not to confuse with the variables x, y, etc. in the terms of predicate logic)

- M, N, etc. for the λ-terms annotating proof-trees

  (not to confuse with the terms of predicate logic t, u,...)

$$\frac{}{\Gamma, \alpha : P \vdash \alpha : P}$$

$$\frac{\Gamma, \alpha : P \vdash M : Q}{\Gamma \vdash \lambda\alpha.M : P \Rightarrow Q} \qquad \frac{\Gamma \vdash M : P \Rightarrow Q \qquad \Gamma \vdash N : P}{\Gamma \vdash M\ N : Q}$$

Theorem : If $\vdash M : P$ then $M \Vdash P$

What about the other connectives?

We can extend the λ-calculus

to account for the introduction and elimination rules of the other connectives

$\wedge$, $\vee$

$$\frac{\Gamma \vdash M : P_1 \quad \Gamma \vdash N : P_2}{\Gamma \vdash (M, N) : P_1 \wedge P_2} \qquad \frac{\Gamma \vdash M : P_1 \wedge P_2}{\Gamma \vdash \pi_i(M) : P_i} i \in \{1, 2\}$$

$P_1 \wedge P_2$ is a product type "$P_1 * P_2$"

Can you give a proof-term for the valid formula $(P_1 \wedge P_2) \Rightarrow (P_2 \wedge P_1)$?

$$\frac{\Gamma \vdash M : P_i}{\Gamma \vdash \mathsf{inj}_i(M) : P_1 \vee P_2} i \in \{1, 2\}$$

$$\frac{\Gamma \vdash M : P_1 \vee P_2 \quad \Gamma, \alpha_1 : P_1 \vdash N_1 : Q \quad \Gamma, \alpha_2 : P_2 \vdash N_2 : Q}{\Gamma \vdash \mathsf{match}\ M\ \mathsf{with}\ \mathsf{inj}_1(\alpha_1) \mapsto N_1, \mathsf{inj}_2(\alpha_2) \mapsto N_2 : Q}$$

$P_1 \vee P_2$ is a sum type "$P_1 + P_2$"

Can you give a proof-term for the valid formula $(P_1 \vee P_2) \Rightarrow (P_2 \vee P_1)$?

$\forall$, $\exists$, $\bot$

$$\frac{\Gamma \vdash M : P}{\Gamma \vdash \lambda x.M : \forall x P} \qquad \frac{\Gamma \vdash M : \forall x P}{\Gamma \vdash M\, t : \{t/x\}\, P}$$

Can you give a proof-term for the valid formula

$(\forall x (p\, x)) \Rightarrow (\forall y (p\, y \Rightarrow q\, y)) \Rightarrow \forall z (q\, z)$?

$$\frac{\Gamma \vdash M : \{t/x\}\, P}{\Gamma \vdash \langle t, M \rangle : \exists x P} \qquad \frac{\Gamma \vdash M : \exists x P \quad \Gamma, \alpha : P \vdash N : Q}{\Gamma \vdash \text{let } \langle x, \alpha \rangle = M \text{ in } N : Q}$$

Can you give a proof-term for the valid formula

$(\exists x (p\, x)) \Rightarrow (\exists y (p\, y \Rightarrow q\, y)) \Rightarrow \exists z (q\, z)$?

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \text{abort}(M) : P}$$

$\neg P$ defined as $P \Rightarrow \bot$, and $\top$ defined as $\neg \bot$ (i.e. $\bot \Rightarrow \bot$)

Summing up the syntax

		Intro-constructs	Elim-constructs	
$M, N, \ldots ::=$		α		axiom
	$\mid$	$\lambda\alpha.M$	$\mid M\ N$	$\Rightarrow$
	$\mid$	(M, N)	$\mid \pi_i(M)$	$\wedge$
	$\mid$	$\mathsf{inj}_i(M)$	$\mid$ match M with $\mathsf{inj}_1(\alpha_1) \mapsto N_1, \mathsf{inj}_2(\alpha_2) \mapsto N_2$	$\vee$
	$\mid$	$\lambda x.M$	$\mid M\ t$	$\forall$
	$\mid$	$\langle t, M \rangle$	$\mid$ let $\langle x, \alpha \rangle = M$ in N	$\exists$
			$\mid$ abort(M)	$\bot$

Reductions

$$
\begin{array}{ll}
(\lambda\alpha.M)\ N & \longrightarrow \left\{{}^{N}\!/_{\alpha}\right\} M \\
\pi_i((M_1, M_2)) & \longrightarrow M_i \\
\text{match inj}_i(M)\text{ with inj}_1(\alpha_1) \mapsto N_1, \text{inj}_2(\alpha_2) \mapsto N_2 & \longrightarrow \left\{{}^{M}\!/_{\alpha_i}\right\} N_i \\
(\lambda x.M)\ t & \longrightarrow \left\{{}^{t}\!/_{x}\right\} M \\
\text{let } \langle x, \alpha \rangle = \langle t, M \rangle \text{ in } N & \longrightarrow \left\{{}^{t,M}\!/_{x,\alpha}\right\} N
\end{array}
$$

+ some permutation rules such as

$$(\text{match } M \text{ with inj}_1(\alpha_1) \mapsto N_1, \text{inj}_2(\alpha_2) \mapsto N_2)\ N$$

$$\longrightarrow\ \text{match } M \text{ with inj}_1(\alpha_1) \mapsto (N_1\ N), \text{inj}_2(\alpha_2) \mapsto (N_2\ N)$$

$$\pi_i(\text{match } M \text{ with inj}_1(\alpha_1) \mapsto N_1, \text{inj}_2(\alpha_2) \mapsto N_2)$$

$$\longrightarrow\ \text{match } M \text{ with inj}_1(\alpha_1) \mapsto \pi_i(N_1), \text{inj}_2(\alpha_2) \mapsto \pi_i(N_2)$$

$\ldots$

Recovering the match with the meta-level

Reduction still preserves typing: If $\Gamma \vdash M : P$ and $M \longrightarrow M'$ then $\Gamma \vdash M' : P$

Through the Curry-Howard isomorphism,

this is describing a proof transformation process

Termination (proof: not today): We still have termination of typed terms

Corollaries (proof: not today):

Consistency	There is no closed proof of $\bot$
Witness property	If $\vdash \exists x\, P[x]$ then there is a term t such that $\vdash P[t]$
Disjunction property	If $\vdash P_1 \vee P_2$ then either $\vdash P_1$ or $\vdash P_2$

Conclusion: In the empty theory, we recover a full match between object-level and meta-level (in the sense discussed before)

Remark: Law of Excluded Middle would break all of the above approach

In non-empty theories

Axioms labelled by variables α, β, etc.

without computational role

Theorem about shape of irreducible typed λ-terms no longer holds if not closed

In some theories (e.g. $\mathcal{PA}$), a computational role can be given to axioms

Theorem holds again, and its corollaries:

Consistency, Witness property, Disjunction property

Programming by proving

In arithmetic, does $\forall x\, \exists y\, (x = 2 \times y \vee x = 2 \times y + 1)$

have a proof in intuitionistic logic? by induction on x!

What about $\exists y\, (25 = 2 \times y \vee 25 = 2 \times y + 1)$?

What is the witness? How do you compute it?

An intuitionistic proof of

$$\forall x\, \exists y\, (x = 2 \times y \vee x = 2 \times y + 1)$$

is a program that computes the half here by recursion on x!

Its execution mechanism is the proof-transformation process described before

The program is *correct* with respect to the *specification*

$$x = 2 \times y \vee x = 2 \times y + 1$$

IV. Putting it all together: HOL with proof-terms, Dependent types

Using λ-calculus for both propositions and proofs

So far in logic, λ-calculus used

- at the level of propositions in Higher-Order Logic
- at the level of proofs in the Curry-Howard correspondence

(so far in intuitionistic first-order logic)

Can we have combine the two in one system,

with the λ-calculus operating at both levels?

This means

- equip (the intuitionistic version of) HOL with a notion of proof-terms based on λ-calculus
- equivalently, extend the Curry-Howard correspondence so that the types are the propositions of HOL

This is called System F_ω

System F_ω, equipping (intuitionistic) HOL with proof-terms

HOL types $\quad A, B, \ldots ::= \ \mathsf{Prop} \mid a \mid A \to B$

HOL terms $\quad t, u, \ldots ::= \ x \mid t \Rightarrow u \mid \forall x^A t \mid \lambda x.t \mid t\ u$

HOL proof-terms $\quad M, N, \ldots ::= \ \alpha \mid \lambda \alpha.M \mid M\ N \mid \lambda x.M \mid Mt$

We can write the following well-formed HOL propositions:

$$\forall x^{\mathsf{Prop}}((\forall y^{\mathsf{Prop}} y) \Rightarrow x)$$

$$\forall x^{\mathsf{Prop}} \forall y^{\mathsf{Prop}}(((x \Rightarrow y) \Rightarrow x) \Rightarrow x)$$

$$\forall x^a \forall y^a ((\forall z^{a \to \mathsf{Prop}}(z\ x \Rightarrow z\ y)) \Rightarrow (\forall z^{a \to \mathsf{Prop}}(z\ y \Rightarrow z\ x)))$$

Can you find proof-terms for them?

System F_ω, extending the C-H correspondence for propositional logic

Kinds $\quad A, B, \ldots ::= \ \mathsf{Prop} \mid a \mid A \rightarrow B$

Types $\quad t, u, \ldots ::= \ x \mid t \Rightarrow u \mid \forall x^A t \mid \lambda x.t \mid t\ u$

Terms $\quad M, N, \ldots ::= \ \alpha \mid \lambda\alpha.M \mid M\ N \mid \lambda x.M \mid Mt$

We can write the following well-formed HOL propositions:

$$\forall x^{\mathsf{Prop}}((\forall y^{\mathsf{Prop}} y) \Rightarrow x)$$

$$\forall x^{\mathsf{Prop}} \forall y^{\mathsf{Prop}}(((x \Rightarrow y) \Rightarrow x) \Rightarrow x)$$

$$\forall x^a \forall y^a ((\forall z^{a \rightarrow \mathsf{Prop}}(z\ x \Rightarrow z\ y)) \Rightarrow (\forall z^{a \rightarrow \mathsf{Prop}}(z\ y \Rightarrow z\ x)))$$

Can you find proof-terms for them?

Different dependencies

types depend on types

$A \to B$ allows $\lambda x.t$ and $t\ u$

e.g. $z\ y$ on the previous slide

(proof-)terms depend on (proof-)terms

$t \Rightarrow u$ allows $\lambda \alpha.M$ and $M\ N$

e.g. to prove $z\ y \Rightarrow z\ x$, we must provide a proof-term for $z\ x$ that can depend on a proof-term for $z\ y$

(proof-)terms depend on types

$\forall x^A t$ allows $\lambda x.M$ and $M\ t$

e.g. to prove $\forall z^{\text{Prop}}(f\,z)$, we must provide a proof-term for $f\,z$ that can depend on the type z

Dependent types

Types depending on terms? e.g. the type of lists of length n:

$$\text{list } n$$

Currently, impossible to write $\forall n^{\mathsf{nat}} \forall l^{\mathsf{list}\ n} P(n, l)$

if $(\text{list } n)$ is a type, and therefore l is a proof-term variable, we do not have the quantifier $\forall l^{\mathsf{list}\ n}$ or the possibility to make $P(n, l)$ really depend on l.

Dependent types add that possibility

Extending F_ω with this gives the Calculus of Constructions (CoC)

Alternative is to make kinds depend on types (i.e. "**Dependent kinds**"), so that $(\text{list } n)$ is a valid kind.

Once everything depends on everything

One realises that $A \to B$ and $t \Rightarrow u$ are really

particular cases of $\forall x^A B$ and $\forall \alpha^t u$ when $x \notin$ FV(B) and $\alpha \notin$ FV(u)

Instead of defining

- first, what well-formed (=well-typed) terms and formulae are
- second, what the notion of provability on formulae is (via e.g. an inference system)

. . . one defines them both at the same time via a single judgement

$$\Gamma \vdash M : A$$

"In environment Γ, M has type A / M is a proof of A"

A logic talking about the functions described by typed λ-terms . . . is a logic that talks about its own proofs (careful there: remember a logic is either inconsistent or cannot prove its own consistency)

Calculus of Constructions

Let $\text{Prop} = \text{Type}_0$

$$\frac{}{[]\ \text{wf}} \qquad \frac{\Gamma \vdash A : \text{Type}_i}{\Gamma, x : A\ \text{wf}}\ x \text{ fresh}$$

$$\frac{\Gamma\ \text{wf}}{\Gamma \vdash \text{Type}_0 : \text{Type}_1} \qquad \frac{\Gamma \vdash A : \text{Type}_i \quad \Gamma, x : A \vdash B : \text{Type}_j}{\Gamma \vdash \forall x^A B : \text{Type}_j}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A . M : \forall x^A B} \qquad \frac{\Gamma \vdash M : \forall x^A B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : \{^N/_x\} B}$$

$$\frac{\Gamma\ \text{wf}}{\Gamma \vdash x : A}\ (x : A) \in \Gamma \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash B : \text{Type}_i}{\Gamma \vdash M : B}\ A \longleftrightarrow^* B$$

Careful: "everything depends on everything" could suggest there is just one Type, with $\text{Type} : \text{Type}$. This is inconsistent

Instead, consistency relies on $\text{Type}_0 : \text{Type}_1$, and can generalise to an infinite hierarchy of universes $\text{Type}_0 : \text{Type}_1 : \cdots : \text{Type}_i : \cdots$

Inductive types

Many proof assistants (Coq, Matita, Lean, Agda, Epigram, Twelf, Lego, etc) are developed for variants of this logic, with some features removed or added.

Coq, Matita, etc add to the Calculus of Constructions (with infinitely many universes) inductive types, which generalise in that logic the algebraic datatypes of ML languages, and are used to represent

- enumerated types, e.g. booleans $\{\mathsf{true}, \mathsf{false}\}$ (different from Prop!)
- tuples, records
- natural numbers
- lists
- trees
- other logical connectives $\wedge$, $\vee$, etc
- existential quantifier
- equality

V. Special treatment of equality: interpretation of its proofs

Equality

In pure first-order logic: well-understood, both semantically and syntactically

reflexivity + Leibniz $$\frac{}{\Gamma \vdash t = t} \qquad \frac{}{\Gamma \vdash t = u \Rightarrow P(t) \Rightarrow P(u)}$$

If logic / theory talks about specific kinds of individuals (sets, functions, proofs)

important choices need to be made

(by tuning your first-order axioms / tuning your logical system)

- **Sets** are usually taken to be extensional: $\forall xy, (\forall z, z \in x \Leftrightarrow z \in y) \Rightarrow x = y$
- **Functions**? it depends.

  In set theory, a function is represented as a set (its graph) $\Rightarrow$ extensional

  Seen as programs, it is not unnatural to consider some intentionality
- Equality of **formulae**, equality of **proofs**?

  In Frege's tradition, where predicates are seen as functions to $\{\mathsf{true}, \mathsf{false}\}$, they are usually extensional.

  A logic that talks about its own proofs can state an equality between two proofs.

  When is the statement provable?

Equality in type theory

…is an inductive type with one constructor for reflexivity $\text{eq\_refl} : t = t$

Question: how to computationally interpret equality and proofs of equality (as we interpreted other formulae as sets of realisers, and proofs as realisers)?

Typing rules for eliminating inductive types are sufficient to get

- Leibniz principle
- disequality of terms of an inductive type with different constructors at their head
- constructors of inductive types are injective

e.g. the following axioms of arithmetic are provable (in-built, no need for axioms)

- $\forall n (0 \neq S\ n)$
- $\forall nm ((S\ n = S\ m) \Rightarrow (n = m))$

But type theory does not impose that the interpretation of $t = u$ has a realiser if and only if the interpretation of t is equal to the interpretation of u

A modern idea is to interpret types as topological spaces, and let the realisers of $t = u$ be the (continuous) paths from the interpretation of t to that of u

This is **Homotopy Type Theory** (HoTT)

A few points about Homotopy Type Theory 1/2

- There is an equality between t and u if they belong to the same connected component in the interpretation of their type

- There can be different proofs of an equality $t = u$, interpreted as different paths

- We can formalise a notion of equality between two proofs of equality π and π' as the possibility to continuously deform one path into the other (this is how paths of a topological space form a topological space)

- We can state the equality between two proofs of equality between proofs of equality, and so on and so forth...

A few points about Homotopy Type Theory 1/2

- Homotopy theorists like to impose the univalence axiom,

  which entails that isomorphic types are equal

  (this does not come for free in standard type theory)

- A characterisation of usual notions comes out of the following hierarchy:
  - "Propositions" are types that are either empty or entirely connected,

    with the types of equalities entirely connected,

    the types of equalities between equalities entirely connected, etc
  - "Sets" are types whose elements may be equal "in at most one way",

    i.e. equalities between its elements are propositions
  - etc

Main thing to take away

Via the Curry-Howard correspondence

Programming = Proving

Proving proposition A = Inhabiting type A

VI. Appendix

More detailed slides on λ-calculus

Three constructs:

- variables, e.g. x, y, z
- applications, e.g. $t\ u$
- λ-abstractions, e.g. $\lambda x.t$

In other words, λ-terms are defined by the following syntax:

$$t, u, v, \ldots ::= \ x \mid t\ u \mid \lambda x.t$$

Notational conventions:

Implicit parentheses: the concrete syntax $t_0\ t_1\ \ldots\ t_n$ means $(\ldots(t_0\ t_1)\ \ldots\ t_n)$

Scope of λ-abstractions: when writing the concrete syntax $\lambda x.\ldots$,

as much of ... as possible must be understood to be under the λ-abstraction,

e.g. $\lambda x.xy$ means $\lambda x.(xy)$, not $(\lambda x.x)y$

Free variables:

$$\begin{aligned} \mathsf{FV}(x) &:= x \\ \mathsf{FV}(\lambda x.t) &:= \mathsf{FV}(t) \setminus \{x\} \\ \mathsf{FV}(t\ u) &:= \mathsf{FV}(t) \cup \mathsf{FV}(u) \end{aligned}$$

x is *bound* in $\lambda x.t$

The syntax is quotiented by α-equivalence, e.g. $\lambda x.x$ and $\lambda y.y$ are the same λ-term

Substitution $\{{}^{u}\!/\!_{x}\}t$ defined by induction on t in a way that avoids variable capture

Reduction

One reduction rule, called β-reduction: $(\lambda x.t)\ u \longrightarrow \ \{{}^{u}\!/\!_{x}\}t$

We do not only reduce at the root of terms, but also deep inside them.

$\longrightarrow^*$ is reflexive and transitive closure of $\longrightarrow$

$\longleftrightarrow^*$ is reflexive, transitive and symmetric closure of $\longrightarrow$

Currification: No need to explicitly model functions with several arguments, since a function f with 2 arguments $(x,y) \mapsto e[x,y]$ can be seen as a function g mapping one argument x to: the function that maps y to $e[x,y]$

$$(x,y) \mapsto e[x,y] \text{ equivalent to } x \mapsto (y \mapsto e[x,y])$$

In λ-calculus syntax: $\lambda x.\lambda y.e[x,y]$

To apply it, instead of writing $f(x,y)$, write $((g\ x)\ y)$

Example: $(\lambda x.\lambda x'.x')\ ((\lambda y.y)\ z)\ ((\lambda y.y)\ z')$

How many ways to reduce this term? Several.

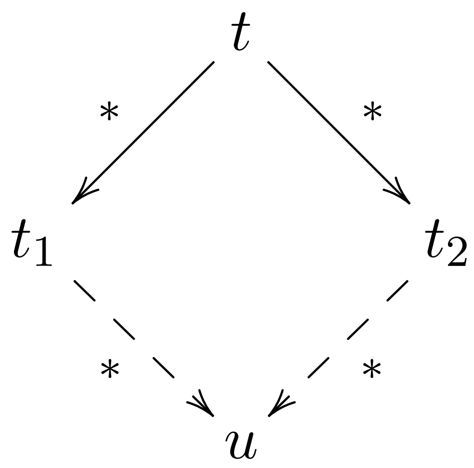
In this case, they all end up with the same (irreducible) λ-term z'

Confluence

Is this a general property? Yes!

Theorem: the relation $\longrightarrow$ is *confluent*, i.e.

If $t \longrightarrow^* t_1$ and $t \longrightarrow^* t_2$, then there exists u such that $t_1 \longrightarrow^* u$ and $t_2 \longrightarrow^* u$



Proof: not today

Corollary: Irreducible forms are unique, i.e.

Given a λ-term t, there is at most one irreducible u such that $t \longrightarrow^* u$

Existence of u?

What about $\omega = (\lambda x.x\ x)\ (\lambda x.x\ x)$?

What about $(\lambda x.y)\ \omega$?

The simply-typed λ-calculus

We consider some *base types*: a, b, etc

Syntax of types

$$A, B, C, \ldots ::= \; a \mid A \to B$$

Notational conventions on implicit parentheses:

the concrete syntax $A_1 \to \cdots \to A_n \to A_0$ means $A_1 \to (\cdots \to (A_n \to A_0) \cdots)$

Typing context $\Delta, \ldots$: finite map from λ-calculus variables to types

Notation: Δ can be for instance $x_1 : A_1, \ldots, x_n : A_n$ We can write $\Delta, x : A$

Typing is a relation between 3 things: a context, a λ-term and a type

defined inductively by typing rules:

$$\frac{}{\Delta, x : A \vdash x : A}$$

$$\frac{\Delta \vdash t : A \to B \qquad \Delta \vdash u : A}{\Delta \vdash t\, u : B} \qquad \frac{\Delta, x : A \vdash t : B}{\Delta \vdash \lambda x.t : A \to B}$$

Properties of the typing system

Remark: If $\Delta \vdash t : A$ then FV(t) is included in the domain of Δ

Reduction preserves typing: If $\Delta \vdash t : A$ and $t \longrightarrow t'$ then $\Delta \vdash t' : A$

Proof: easy induction on the inductive property $t \longrightarrow t'$

Termination: If $\Delta \vdash t : A$ then all reduction paths starting from t are finite

Proof: not today

After extending the λ-calculus to treat $\wedge, \vee, \forall, \exists, \bot$

Remark: If $\Gamma \vdash M : P$ then FV(M) is included in the domain of Γ

Substitution: If $\Gamma, \alpha : P \vdash M : Q$ and $\Gamma \vdash N : P$, then $\Gamma \vdash \left\{{}^{N}\!/\!_{\alpha}\right\} M : Q$

Reduction still preserves typing: If $\Gamma \vdash M : P$ and $M \longrightarrow M'$ then $\Gamma \vdash M' : P$

Through the Curry-Howard isomorphism,

this is describing a proof transformation process

Termination: If $\Gamma \vdash M : P$ then all reduction paths starting from M are finite

(careful with the permutation rules, though)

The process of transforming proofs terminates, producing proofs of a particular shape:

the typing trees of irreducible λ-terms

Corollary:

Every theorem P that has a proof in a theory $\mathcal{T}$, also has a proof of that shape

Shape of those proofs in the empty theory

Theorem:

1. Any closed, irreducible and typed λ-term, is an intro-construct
2. There is no closed, irreducible λ-term of type $\perp$

Proof: by simultaneous induction on the size of λ-terms

(and 2. easy consequence of 1.)

Corollary (stilll in the empty theory)

Consistency: intuitionistic predicate logic without axioms is consistent

Proof: If $\perp$ has a proof, it also has a proof whose λ-term is an intro-construct. Impossible.

Witness property: if $\vdash \exists x\ P[x]$ then there is a term t such that $\vdash P[t]$

Proof: The proof can be transformed into a proof annotated by an intro-construct, necessarily $\langle t, M \rangle$, which provides t and the proof of $\vdash P[t]$

Disjunction property: if $\vdash P_1 \vee P_2$ then either $\vdash P_1$ or $\vdash P_2$

Proof: The proof can be transformed into a proof annotated by an intro-construct, necessarily $\text{inj}_i(M)$, where M annotates a proof of $\vdash P_i$

Conclusion: In the empty theory, we recover a full match between object-level and meta-level (in the sense discussed before)

Remark: Law of Excluded Middle would break all of the above approach