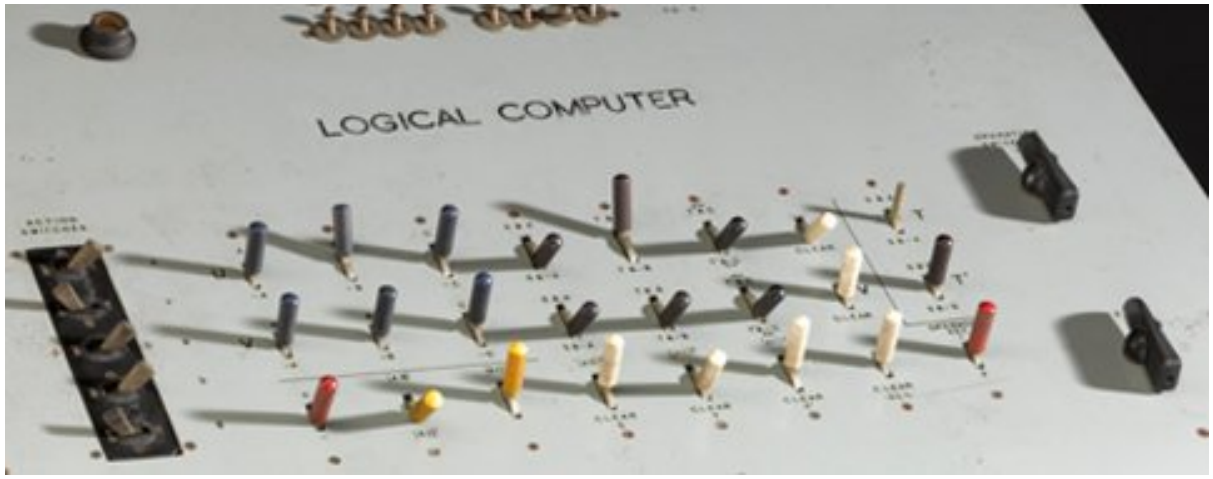
INF551

Computational Logic:

Artificial Intelligence in Mathematical Reasoning

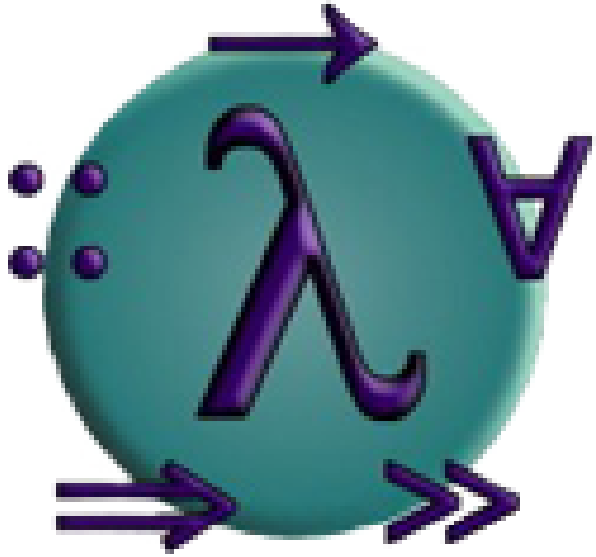


Stéphane Graham-Lengrand

Stephane.Lengrand@Polytechnique.edu

Lecture VII

The theory of functions



Where we stand today

Remember:

Last week we proposed *Set theory* as a universal framework (language+logic+theory) for representing any problem (of mathematical nature)

Aim: mechanics of solving problems in universal framework

$\Longrightarrow$ mechanics of solving any problem

Good points of Set theory:

- Hard to do without sets
- Rather minimalistic (although axiom schema of replacement))
- Nobody's found a contradiction yet (always a plus)
- Can express arithmetic and (we believe) all the mathematics we've written so far

Bad points of Set theory:

- Provability is undecidable (but semi-decidable)
- Doing *anything* in set theory is long and tedious

In any case, Church / Göedel's theorems $\Longrightarrow$

Ability to express arithmetic *incompatible* with decidability of provability/completeness

Formalising problems in *theory* or in *practice*

Why do we only *"believe"* that all the mathematics we've written so far can be done in Set theory?

Have you already seen books/articles/course notes expressed entirely in Set Theory?

Impact of $\mathcal{ZF}$ on the world of mathematics:

Not huge revolution on nature of mathematical activity

At best: thought process at the back of each mathematician's mind raising the question at every reasoning step: *"Would I be able to justify this step within $\mathcal{ZF}$?"*

Conviction that formalising maths in $\mathcal{ZF}$ can be done *"in theory"*

Mathematicians did not formalise their maths in $\mathcal{ZF}$ *"in practice"*

. . . at least until computers were invented

Using computers

Proofs in $\mathcal{ZF}$ are too long, unreadable, cannot be processed by humans

For a start: lacking *definition mechanisms* and *convenient notations*

Could those problems be overcome by using computers?

Can do parsing and pretty-printing

$\Longrightarrow$ use for convenient interfacing between human and Set theory?

Set theory = low-level machine language, computer compiling human proofs into it?

Problem: Human wants to do big-step reasoning.

Challenges:

#1 Need mechanical way to convert big-step reasoning

into small-step reasoning (e.g., using predicate logic & set theory axioms)

#2 Need rigourous language to even express big-step reasoning

Challenge #1 is difficult to achieve with Set theory

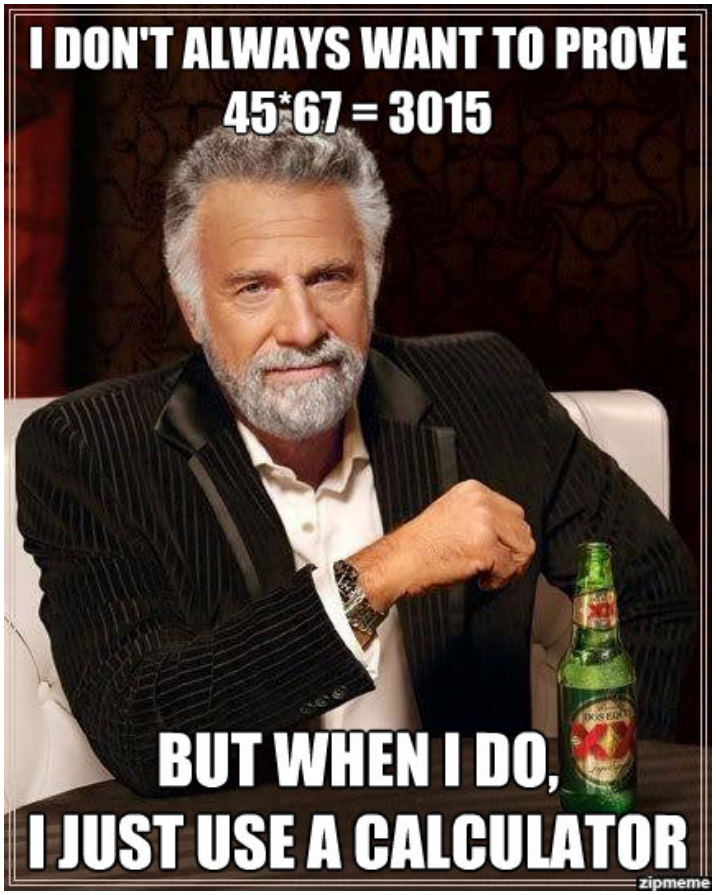
Big-step reasoning to small-step reasoning requires filling the "proof gaps" automatically

Can an algorithm be smart enough to do so, using the right axioms at the right time?

Axioms not very "computer-friendly"

Proving $45 * 67 = 3015$ requires a huge proof in $\mathcal{PA}$...

...and an even bigger proof in $\mathcal{ZF}$, with complex use of the axioms



As $45 * 67$ can be trivially computed,

using a computer to search for a proof of $45 * 67 = 3015$ in $\mathcal{PA}$ or $\mathcal{ZF}$...

...is using a computer the wrong way

In summary, computers have more processing power,

but no intuition

$\Longrightarrow$ so we need

an **alternative to $\mathcal{ZF}$ that is more computer-friendly**

Computer-friendly stuff

Set are not very computer-friendly:

one of the math objects least convenient to implement

Integers, lists, trees, on the other hand, are computer-friendly

(while not primary objects in Set theory)

So are Functions! (again, while not primary objects in Set theory)

. . . since they correspond to computer programs! (well, computable ones)

. . . but computer-friendly as long as one doesn't ask whether 2 functions are *extensionally equal* (whether for all possible inputs, they produce the same output)

Remember: whether 2 Turing machines produce the same output, given the same input, is an *undecidable problem!*

Today: theory of sets $\Longrightarrow$ **theory of functions**

(and we will drop extensionality)

Contents

I. Back to Russell's paradox

Frege's system

In fact, Frege's Begriffschrift (1879) considers functions as **more primitive** than sets

They are part of his logic (rather than the purpose of an axiomatic theory)

Frege distinguishes *objects* (which include natural numbers) and *functions*

- Variables $x, y, z, \ldots$ represent objects, while variables $f, g, h, \ldots$ represent functions
- (The representation of) a function can be applied to (the representation of) an object,

  so $f(x)$ represents an object
- Both kinds of variables can be quantified over: $\forall x \ldots$ and $\forall f \ldots$
- *predicates*[1] are those functions mapping objects to either 0 or 1
- *formulae* are the various representations of 0 and 1
- Given a formula $A[x]$,

  Frege allows the representation, in his syntax, of the function $c \mapsto [\![A[x]]\!]_{x \mapsto c}$

  In modern notations (Church's), such a function is denoted

$$\lambda x.A[x]$$

- His system allows to prove the *simplification property*: $\forall y\ (((\lambda x.A[x])(y)) \Leftrightarrow A[y])$

[1] Frege called them "concepts"

Frege's system

Notice that we are already outside the syntax of (what you know to be) predicate logic, since

- there are 2 kinds of variables and terms (instead of 1): for objects and for functions
- formulae are particular object terms & $\lambda x.A[x]$ builds a function term from a formula
- moreover $\lambda x.A[x]$ and $\lambda y.A[y]$ are identified as the same term

$\Longrightarrow$ notion of binder in the syntax of terms

But so far so good, no one has found any contradiction in this system

But in 1893, Frege adds a tool for creating objects from functions

(to speak about equality of functions)

In the case of a predicate F,

ϵF denotes a particular object called the *extension of concept* F,

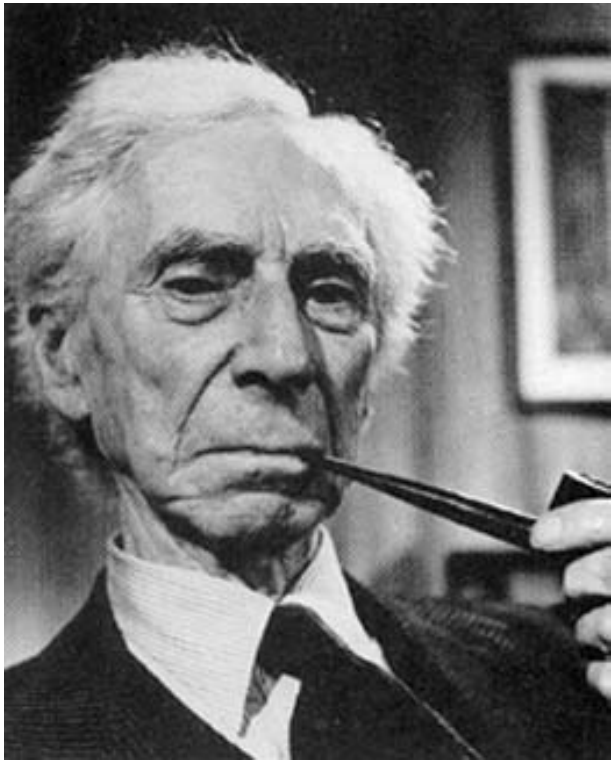
satisfying *"Basic Law V"* (Axiom 5 of Frege's system):

$$\forall F \forall G \left((\epsilon F = \epsilon G) \Leftrightarrow \forall x \left(Fx \Leftrightarrow Gx \right) \right)$$

Russell's paradox (1902)

Russell expresses his paradox both in terms of set theory and in Frege's logical system

(with functions)



Let P be the following predicate term:

$$\lambda x.\exists F\,(x = \epsilon F \wedge \neg F(x))$$

Clearly, $P(\epsilon P)$ means

$$(\lambda x.\exists F\,(x = \epsilon F \wedge \neg F(x)))(\epsilon P)$$

and by the simplification property this is equivalent to

$$\exists F\,(\epsilon P = \epsilon F \wedge \neg F(\epsilon P))$$

Then by Basic Law V we know that

$$\epsilon P = \epsilon F \text{ is equivalent to } \forall x\,(Px \Leftrightarrow Fx)$$

so the above is equivalent to

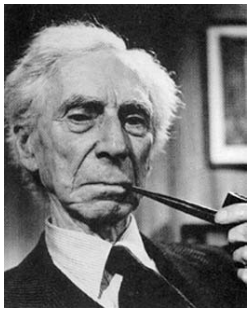
$$\neg P(\epsilon P)$$

In Frege's system, there is a formula $P(\epsilon P)$ equivalent to its negation

$\Longrightarrow$ in Frege's system **there is a proof of** $\bot$

No set involved, but of course in substance ϵP is "the set of all objects satisfying P"

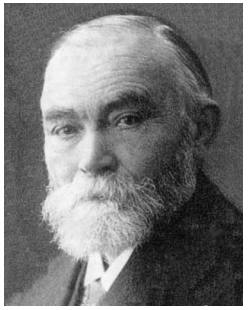
In their own words. . .



Original letter to Frege:

“There is just one point where I have encountered a difficulty. You state that a function too, can act as the indeterminate element. This I formerly believed, but now this view seems doubtful to me because of the following contradiction. Let w be the predicate: to be a predicate that cannot be predicated of itself. Can w be predicated of itself? From each answer its opposite follows. Therefore we must conclude that w is not a predicate.”

Frege:



“A scientist can hardly meet with anything more undesirable than to have the foundation give way just as the work is finished. In this position I was put by a letter from Mr Bertrand Russell as the work was nearly through the press.”

Vicious circle

It became quite clear that Russel's paradox was about a *"vicious circle"*:

the liar saying "I am a liar"

In Frege's system, the simplification property

$$\forall y\ (((\lambda x.A[x])(y)) \Leftrightarrow A[y])$$

can be seen as a computational rule, oriented left-to-right

Basic Law 5 allows us to forget about ϵ

and basically "simplify"

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

to

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

The paradox comes from the non-termination of the "simplification" process

II. The λ-calculus

Seeing computation in the Begriffschrift

Church, following previous work by Curry, made a theory of this simplification process:

The λ-calculus

Retaining only the necessary ingredients from Frege.

We need:

- variables $x, y, z, \ldots$
- λ-abstractions
- applications
- and that's it

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

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

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 $\ldots$ as possible must be understood to be under the λ-abstraction,

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

We have seen all this before

Free variables: as you expect,

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

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:

$$(\lambda x.t)\, u \longrightarrow_{\text{root}} \{^{u}\!/_{x}\}t$$

t, u ranging over terms, x over variables

Redex: term that can be reduced by $\longrightarrow_{\text{root}}$, i.e., an instance of $(\lambda x.t)\, u$

But we do not only want to reduce at the root of terms, also inside them:

$\longrightarrow$ inductively defined by

- if $t \longrightarrow_{\text{root}} t'$, then $t \longrightarrow t'$
- if $t \longrightarrow t'$, then $(t\, u) \longrightarrow (t'\, u)$
- if $u \longrightarrow u'$, then $(t\, u) \longrightarrow (t\, u')$
- if $t \longrightarrow t'$, then $(\lambda x.t) \longrightarrow (\lambda x.t')$

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

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

Currification

No need to model functions with several arguments:

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 $e[x, y]$

$(x, y) \mapsto e[x, y]$ 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 redexes? 3

Several ways in which we can reduce

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

(t irreducible if it cannot be reduced by $\longrightarrow$, i.e., none of its sub-terms is a redex)

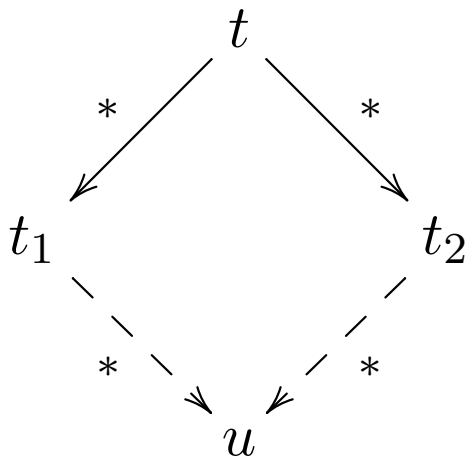
Confluence

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 $t_2 \longrightarrow^* u$

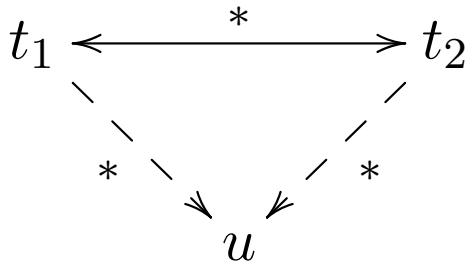


Proof: not today

Corollaries

Corollary 1: the relation $\longrightarrow$ is *Church-Rosser*, i.e.,

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



Proof: by induction on the number of "peaks"

Corollary 2: Irreducible forms are unique, i.e.,

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

Proof: trivial

Existence of u?

Back to Russell's vicious circle

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

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

Definition

t weakly normalising if there exists irreducible u such that $t \longrightarrow^* u$

t strongly normalising if there is no infinite reduction sequence starting from t

strongly normalising clearly implies weakly normalising (König's lemma)

Examples above show that it is not the same

Russell's paradox is based on absence of normalisation

(using the negation connective):

$$\begin{aligned}
(\lambda x.\neg(x\ x))\ (\lambda x.\neg(x\ x)) &\longrightarrow \neg((\lambda x.\neg(x\ x))\ (\lambda x.\neg(x\ x))) \\
&\longrightarrow \neg\neg((\lambda x.\neg(x\ x))\ (\lambda x.\neg(x\ x))) \\
&\longrightarrow \neg\neg\neg((\lambda x.\neg(x\ x))\ (\lambda x.\neg(x\ x))) \\
&\ldots
\end{aligned}$$

III. Ensuring termination of λ-calculus with a typing system

Intuition

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

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

Paradox in Frege's system lies in the question of whether

applying a predicate to itself ($P(P)$ or even $P(\epsilon P)$) makes mathematical sense

Paradox in naïve set theory lies in the question of whether

it makes mathematical sense for a set to belong to itself ($x \in x$)

At least in the first two cases, this has to do with the

domain of definition of function x or predicate P

We need to control it to avoid paradoxes

Controlling domains and applications

Easy 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

Idea:

Abstract away from sets to retain only the necessary ingredients for

controlling domains of definition 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

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:

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

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

Properties of the typing system

Remark: If $\Delta \Vdash t : A$ then $\mathrm{FV}(t)$ is included in the domain of Δ

Substitution: If $\Delta, x : A \Vdash t : B$ and $\Delta \Vdash u : A$, then $\Delta \Vdash \{u/x\}t : B$

Proof: easy induction on t

Subject Reduction: If $\Delta \Vdash t : A$ and $t \longrightarrow t'$ then $\Delta \Vdash t' : A$

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

(base case = root reduction; uses above lemma)

Strong Normalisation: If $\Delta \Vdash t : A$ then t is strongly normalising

Proof: not today

Easy extension

Easy to add typed constants to the simply-typed λ-calculus

(Higher-order) signature: set of constants equipped with fixed types (instead of arities)

Syntax of the λ-calculus with constants:

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

where c ranges over the signature

Constants play no role in computation. Same definition of reduction relation $\longrightarrow$

Add 1 typing rule for constants:

$$\frac{}{\Delta \Vdash c : A}$$

if $c : A$ is in the signature

Properties on previous slide still hold

(constants behave like free variables that can never be bound)

IV. Higher-Order Logic: using the typed λ-calculus to fix Frege's logic

Setting up the scene

2 base types: i (individuals), Prop (propositions)

As opposed to predicate logic, terms and formulae are made of the same syntax:

that of λ-terms with constants

Signature of constants:

$$\top : \text{Prop} \qquad \wedge : \text{Prop} \to \text{Prop} \to \text{Prop}$$

$$\bot : \text{Prop} \qquad \vee : \text{Prop} \to \text{Prop} \to \text{Prop}$$

$$\neg : \text{Prop} \to \text{Prop} \qquad \Rightarrow : \text{Prop} \to \text{Prop} \to \text{Prop}$$

$$\forall_A : (A \to \text{Prop}) \to \text{Prop}$$

$$\exists_A : (A \to \text{Prop}) \to \text{Prop}$$

$P \wedge Q$ abbreviation for $\wedge\ P\ Q$,...

$\forall_A x\ P$ abbreviation for $\forall_A\ \lambda x.P$, ...

Philosophy of the proof system

We get a typing system deriving typing judgements of the form $\Delta \Vdash t : A$

A "formula" P is now just a λ-term of type Prop (depends on context Δ!)

Next slide: a proof system deriving judgements of the form $\Gamma \vdash_{\Delta} P$

where Δ is a typing context,

P (the goal) is a formula according to Δ,

and Γ (the hypotheses) a set of formulae according to Δ

The inference rules -part 1

You already know the rules for propositional logic (see INF551 course notes):

$$\frac{\Gamma, P \vdash_\Delta Q}{\Gamma \vdash_\Delta P \Rightarrow Q} \qquad \frac{\Gamma \vdash_\Delta P \Rightarrow Q \quad \Gamma \vdash_\Delta P}{\Gamma \vdash_\Delta Q}$$

$$\frac{\Gamma \vdash_\Delta P_1 \quad \Gamma \vdash_\Delta P_2}{\Gamma \vdash_\Delta P_1 \wedge P_2} \qquad \frac{\Gamma \vdash_\Delta P_1 \wedge P_2}{\Gamma \vdash_\Delta P_i}\, i \in \{1,2\}$$

$$\frac{\Gamma \vdash_\Delta P_i \quad \Delta \Vdash P_j : \text{Prop}}{\Gamma \vdash_\Delta P_1 \vee P_2}\, \{i,j\} = \{1,2\} \qquad \frac{\Gamma \vdash_\Delta P_1 \vee P_2 \quad \Gamma, P_1 \vdash_\Delta Q \quad \Gamma, P_2 \vdash_\Delta Q}{\Gamma \vdash_\Delta Q}$$

$$\frac{}{\Gamma \vdash_\Delta \top} \qquad \frac{\Gamma \vdash_\Delta \bot \quad \Delta \Vdash P : \text{Prop}}{\Gamma \vdash_\Delta P}$$

$$\frac{\Gamma, P \vdash_\Delta \bot}{\Gamma \vdash_\Delta \neg P} \qquad \frac{\Gamma \vdash_\Delta \neg\neg P}{\Gamma \vdash_\Delta P}$$

$$\frac{\Delta \Vdash P : \text{Prop}}{\Gamma, P \vdash_\Delta P}$$

The inference rules -part2

We adapt the rules for quantifiers to the higher-order case

$$\frac{\Gamma \vdash_{\Delta, xA} P}{\Gamma \vdash_{\Delta} \forall_A x P} \qquad \frac{\Gamma \vdash_{\Delta} \forall_A x P \quad \Delta \Vdash t : A}{\Gamma \vdash_{\Delta} \left\{ {}^{t}\!/_{x} \right\} P}$$

$$\frac{\Gamma \vdash_{\Delta} \left\{ {}^{t}\!/_{x} \right\} P \quad \Delta, x : A \Vdash P : \mathsf{Prop}}{\Gamma \vdash_{\Delta} \exists_A x P} \qquad \frac{\Gamma \vdash_{\Delta} \exists_A x P \quad \Gamma, P \vdash_{\Delta, xA} Q}{\Gamma \vdash_{\Delta} Q}$$

Finally, we include computation within the reasoning:

$$\frac{\Gamma \vdash_{\Delta} Q \quad \Delta \Vdash P : \mathsf{Prop}}{\Gamma \vdash_{\Delta} P} \; P \longleftrightarrow^{*} Q$$

Property and equality

Property: Whenever $\Gamma \vdash_\Delta P$ we have $\Delta \Vdash P : \mathsf{Prop}$

(invariant by each of the inference rules)

As usual, $P \Leftrightarrow Q$ is an abbreviation for $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$

Equality can be defined by Leibniz:

$$=_A \quad \text{is} \quad \lambda x.\lambda y.\forall_{A \to \mathsf{Prop}} P\, (P\, x \Leftrightarrow P\, y)$$

We can use the usual infix notation $t =_A u$ for $((=_A\ t)\, u)$

Exercises (practical next week):

Prove $\vdash \forall_A x, x =_A x$

Prove that if $\Gamma \vdash_\Delta P[t]$ and $\Gamma \vdash_\Delta t =_A u$ then $\Gamma \vdash_\Delta P[u]$

(assuming $\Delta, y_1 : A \Vdash P : \mathsf{Prop}$, $\Delta \Vdash t : A$ and $\Delta \Vdash u : A$)

HOL computer-friendly?

HOL includes computation within reasoning.

Our hope:

encode arithmetic so that $45 * 67 = 3015$ can be proved in 1 reasoning step.

$$\dfrac{\dfrac{\dfrac{}{\vdash 3015 =_{\mathsf{nat}} 3015}}{\vdash 45 * 67 =_{\mathsf{nat}} 3015}\; 45 * 67 \longleftrightarrow^{*} 3015}$$

We do this next week

You will have noticed that moving from predicate logic to HOL gets us much closer to the Coq system. . .

Questions?