An Introduction to Type Theory

Thorsten Altenkirch

University of Nottingham

Proposition

Proposition: There are two irrational numbers a, b s.t. a^b is rational.

Prove the proposition!

Hints:

- We know that $\sqrt{2}$ is irrational.
- $(a^b)^c = a^{bc}$

A proof ?!

$\sqrt{2}^{\sqrt{2}}$ **is rational.** We are done with $a = b = \sqrt{2}$.

$\sqrt{2}^{\sqrt{2}}$ **is irrational.** Now consider $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$.

$$\begin{aligned} a^b &= \left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} \\ &= \sqrt{2}^{(\sqrt{2}\sqrt{2})} \\ &= \sqrt{2}^{2} \\ &= 2 \end{aligned}$$

but . . .

Exercise:
Write down 2 irrational numbers a, b s.t. a^b is rational!

Constructive Reasoning

- The proof we have given is non-constructive.
- Even though we have proven a statement of the form $\exists x.P(x)$, we cannot name an individual a such that $P(a)$ holds.
- In *classical logic* the following equivalence holds

$$\exists x.P(x) \iff \neg\forall x.\neg P(x)$$

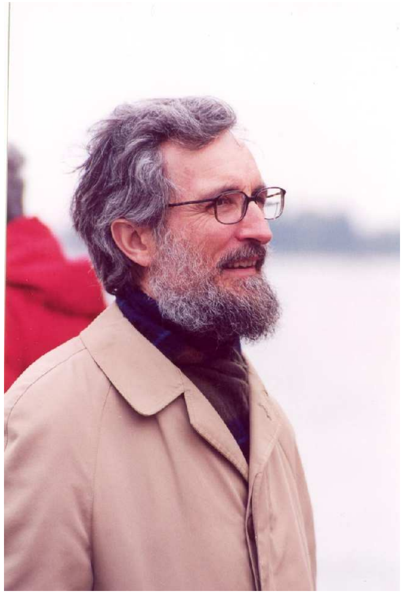
- Constructively, we want to make a difference between

  $\exists x.P(x)$ We are able to *calculate* an a such that $P(a)$ holds.

  $\neg\forall x.\neg P(x)$ We know that it is not the case that P is false everywhere.

Type Theory

- Developed by Per Martin-Löf since 1972 as a constructive foundation of Mathematics.



- At the same time a set theory and a programming language.
- Basis for a number of *Computer Aided Formal Reasonong* systems:
  NuPRL, LEGO, COQ, ALF, . . .
- Dependent types for programming

Plan of the course

1. Intuitionistic logic from a type theoretic perspective
   (I use *constructive* and *intuitionistic* synonymously.
2. Basic constructions of Type Theory
3. An extended example of a formal development:
   Normalisation by evaluation
4. Programming with dependent types

Today

Intuitionistic logic from a type theoretic perspective

- Proof by pattern matching and by elimination
- Propositional logic, constructively
- Predicate Logic, constructively
- Classical vs. intuitionistic logic

Propositions as types

We introduce a judgement

$$a \in A$$

meaning that a is a proof of the proposition A.
We also have a judgement

$$A \in \mathrm{Prop}$$

meaning that A is a proposition.

And (Conjunction)

How to form a conjunction?

$$\frac{A \in \mathrm{Prop} \qquad B \in \mathrm{Prop}}{A \wedge B \in \mathrm{Prop}}$$

How to prove a conjunction?

$$\frac{a \in A \qquad b \in B}{(a,b) \in A \wedge B}$$

(a,b) *is a canonical proof.*

Or (Disjunction)

How to form a disjunction?

$$\frac{A \in \mathrm{Prop} \qquad B \in \mathrm{Prop}}{A \wedge B \in \mathrm{Prop}}$$

How to prove a disjunction?

$$\frac{a \in A}{\mathrm{inl}\, a \in A \vee B} \qquad \frac{b \in B}{\mathrm{inr}\, b \in A \vee B}$$

$\mathrm{inl}\, a, \mathrm{inr}\, b$ *are canonical proofs.*

If, (Implication)

How to form an implication?

$$\frac{A \in \mathrm{Prop} \qquad B \in \mathrm{Prop}}{A \rightarrow B \in \mathrm{Prop}}$$

How to prove an implication?

$$\frac{\begin{array}{c} x \in A \\ \vdots \\ b \in B \end{array}}{\lambda x \in A.b \in A \rightarrow B}$$

$\lambda x \in A.b$ *is a canonical proofs.*

Notation

We will omit type annotations when they are clear from the context,i.e. instead of

$$\lambda x \in A.b$$

we may just write

$$\lambda x.b$$

Non-canonical proofs

How do we prove

$$A \vee (B \wedge C) \to (A \vee B) \wedge (A \vee C)?$$

(given $A, B, C \in \mathrm{Prop}$)
We have to introduce a notion of *non-canonical proofs*:

elimination constants the traditional approach, easy to formalize, but hard to use.

pattern matching suggested by Thierry Coquand in 1992, more intuitive, but tricky metatheory.

Proof by pattern matching

$$f \in A\vee(B\wedge C)\to(A\vee B)\wedge(A\vee C)$$

where

$$\begin{aligned} f\,(\mathrm{inl}\,a) &= (\mathrm{inl}\,a,\mathrm{inl}\,a) \\ f\,(\mathrm{inr}\,(b,c)) &= (\mathrm{inr}\,b,\mathrm{inr}\,c) \end{aligned}$$

Pattern matching

- We start with a trivial pattern

$$f\ x_1 \ldots x_n = ?$$

- We may develop our pattern according to the following rules:
  - If a pattern variable x has type $A \wedge B$ we can replace it by (x_1, x_2) where $x1 \in A, x_2 \in B$ are fresh variables.
  - If a pattern variable x has type $A \vee B$ we can split the line into two, replacing x by $\mathrm{inl}\, x_1$ in the first line and by $\mathrm{inr}\, x_2$ in the second, where $x_1 \in A, x_2 \in B$ are fresh variables.
- Finally, we fill in all our right hand sides with canonical terms which may use variables introduced in the left hand side.
- We will not use recursion in the moment (but later).

Elimination for $\rightarrow$

We didn't introduce any pattern matching rules for $\rightarrow$. Instead we introduce application:

$$\frac{f \in A \rightarrow B \qquad a \in A}{f\,a \in B}$$

where

$$(\lambda x.b)\,a \;=\; b[x \leftarrow a]$$

Here $b[x \leftarrow a]$ means that all *free occurences* of x in b are replaced by a (capture avoiding).

Elimination constants for $\wedge$ and $\vee$

- Instead of using pattern matching, we may introduce *elimination constants*.
- These are special cases of pattern matching.
- Important principle: *Equivalence of pattern matching and elimination*
  All the proofs we can do with pattern matching can be done using the elimination constants.
- This way *pattern matching* can be reduced to elimination.
- While we move to more interesting systems it becomes more subtle to maintain this property.

Elimination constants $\wedge$

$$\frac{p \in A \wedge B}{\mathrm{fst}p \in A} \qquad \frac{p \in A \wedge B}{\mathrm{snd}p \in B}$$

where

$$\begin{aligned} \mathrm{fst}(a,b) &= a \\ \mathrm{snd}(a,b) &= b \end{aligned}$$

Elimination constants $\vee$

$$\frac{p \in A \vee B \qquad f \in A \to C \qquad g \in B \to C}{\mathrm{case}\, p\, f\, g \in C}$$

where

$$\begin{array}{rcl} \mathrm{case}\,(\mathrm{inl}\, a)\, f\, g & = & f\, a \\ \mathrm{case}\,(\mathrm{inr}\, b)\, f\, g & = & g\, b \end{array}$$

Proof using elimination constants

$$\lambda p.\mathrm{case}\, p\, (\lambda x.(\mathrm{inl}\, x, \mathrm{inl}\, x))\, (\lambda y\, (\mathrm{inr}\, (\mathrm{fst} y), \mathrm{inr}\, (\mathrm{snd} y)))$$
$$\in A \vee (B \wedge C) \to (A \vee B) \wedge (A \vee C)$$

Example: commutativity of $\vee$

$$\mathrm{orCom} \in A \vee B \rightarrow B \vee A$$

where

$$\begin{aligned} \mathrm{orCom}\,(\mathrm{inl}\,a) &= \mathrm{inr}\,a \\ \mathrm{orCom}\,(\mathrm{inr}\,b) &= \mathrm{inl}\,b \end{aligned}$$

Define $\leftrightarrow$

$$A \leftrightarrow B = (A \rightarrow B) \wedge (B \rightarrow A)$$

Example: associativity of $\wedge$

$$
\begin{aligned}
\mathrm{andAss} &\in (A{\vee}B){\vee}C{\leftrightarrow}A{\vee}(B{\vee}C)\\
\mathrm{andAssL} &\in (A{\vee}B){\vee}C{\rightarrow}A{\vee}(B{\vee}C)\\
\mathrm{andAssR} &\in A{\vee}(B{\vee}C) \rightarrow (A{\vee}B){\vee}C
\end{aligned}
$$

where

$$
\begin{aligned}
\mathrm{andAss} &= (\mathrm{andAssL},\mathrm{andAssR})\\
\mathrm{andAssL}\,((a,b),c) &= (a,(b,c))\\
\mathrm{andAssR}\,(a,(b,c)) &= ((a,b),c)
\end{aligned}
$$

False, True

- We haven't yet introduced False, True.
- Canonical proof $\mathrm{triv} \in \mathrm{True}$
- There is no canonical proof for False!

Pattern matching for True, False

- A pattern variable of type True can be replaced by triv.
  Yes, this is useless!
- If we have a pattern variable of type False we can delete the line.

Elimination constants for True, False

There is no elimination constant for True

$$\frac{p \in \mathrm{False}}{\mathrm{caseF}\, p \in a}$$

Define $\neg$

$$\neg A = A \rightarrow \mathrm{False}$$

Predicate logic

- Since we are doing things type-theoretically, we introduce typed predicate logic.
- We introduce the judgements

  $$S \in \text{Type}$$

  for s is a type, and

  $$s \in S$$

  for s us an element of type S.
- Given types $S_1, \ldots, S_n$ we write

  $$P \in S_1 \rightarrow S_2 \ldots S_n \rightarrow \text{Prop}$$

  for $n - ary$ predicates.
  We use λ abstraction to defi ne and application to apply predicates.

Universal quantification: $\forall$

How to form?

$$\dfrac{S \in \mathrm{Type} \qquad \begin{array}{c} x \in S \\ \vdots \\ P \in \mathrm{Prop} \end{array}}{\forall x \in S.P \in \mathrm{Prop}}$$

How to prove?

$$\dfrac{\begin{array}{c} x \in S \\ \vdots \\ p \in P \end{array}}{\lambda x \in A.p \in \forall x \in S.P}$$

How to use?

$$\dfrac{f \in \forall x \in S.P \qquad s \in S}{f\ s \in P[x \leftarrow s]}$$

Example

$$\mathrm{allAnd} \in (\forall x \in S.P\,x \wedge Q\,x) \to (\forall x \in S.(P\,x)) \wedge (\forall x \in S.Q\,x)$$

Existential quantification: $\exists$

How to form?

$$\frac{S \in \mathrm{Type} \qquad \begin{array}{c} x \in S \\ \vdots \\ P \in \mathrm{Prop} \end{array}}{\exists x \in S.P \in \mathrm{Prop}}$$

How to prove?

$$\frac{s \in S \qquad p \in P}{(s,p) \in \exists x \in S.P}$$

Can we prove. . .

Excluded middle *Tertium non datur* (TND)

$$A \vee \neg A$$

Proof by contradiction *Reductio ad absurdo* (RAA)

$$\neg\neg A \to A$$

?

$A \vee \neg A$

- Since $A \vee \neg A$ is not an implication we have to provide a canonical proof.
- Hence we have to use inl or inr.
- But which one?

$A \to \neg\neg A$

- We can prove $A \to \neg\neg A$
- To prove $\neg\neg A \to A$ we have to prove a *positive* formula A from a *negative* formula $\ldots \to \mathrm{False}$.
- This is not possible by the principle of *entropy*:

  positive formula contain information

  negative formula contain no information

Classical principles

- Both principle (TND),(RAA) are not provable constructively.
- **Exercise:** Show that both are equivalent.
- Constructive logic + TND (or RAA) = classical logic.