An Introduction to Type Theory

Part 1

Tallinn, September 2003

http://www.cs.nott.ac.uk/~txa/tallinn/

Thorsten Altenkirch
University of Nottingham

Warming up

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

Prove it!

Hints:

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

Solution

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

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

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

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

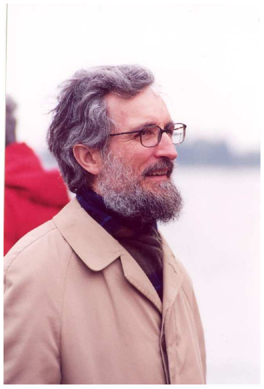
$$\exists x.P(x) \iff \neg\forall x.\neg P(x)$$

- 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 Reasoning* systems:
  NuPRL, LEGO, COQ, ALF, ...
- Dependent types are useful for programming

Plan of the course

1. Intuitionistic logic from a type theoretic perspective
2. Basic constructions of Type Theory
3. More Type Theory
4. An extended example of a formal development: Normalisation by evaluation

Today

Intuitionistic logic from a type theoretic perspective
(*constructive* $\approx$ *intuitionistic*)

- 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?

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

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 proof.*

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

$$A \lor (B \land C) \to (A \lor B) \land (A \lor C)?$$

(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

$$\mathrm{lem} \in A \vee (B \wedge C) \rightarrow (A \vee B) \wedge (A \vee C)$$

where

$$\begin{array}{lcl} \mathrm{lem}\,(\mathrm{inl}\,a) & = & (\mathrm{inl}\,a, \mathrm{inl}\,a) \\ \mathrm{lem}\,(\mathrm{inr}\,(b,c)) & = & (\mathrm{inr}\,b, \mathrm{inr}\,c) \end{array}$$

Pattern matching

- We start with a trivial pattern

$$\mathrm{f}\,x_1 \ldots x_n = ?$$

- 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 $x_1 \in A, x_2 \in B$ are fresh variables.
  - If a pattern variable x has type $A \vee B$ we can split the pattern 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.

Pattern matching

- Finally, we fill in all our right hand sides with terms which may use variables introduced in the left hand side.
- We will not use recursion in the moment (but later).

Elimination for $\rightarrow$

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

where

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

Syntax

- When writing proof terms we use the syntactic convention of λ-calculus.
- Application is left associative, e.g.

$$f\,x\,y = (f\,x)\,y$$

- The scope of binders (like λ) extends as far as possible.

$$\lambda x.f\,x\,y = \lambda x.(f\,x\,y)$$

- We use brackets to override (or emphasize) these conventions.
- We identify terms upto renaming of bound variables (α-conversions).

$$\lambda x.f\,x\,y = \lambda z.f\,z\,y$$

Substitution

- $b[x \leftarrow a]$ means that all *free occurences* of x in b are replaced by a.
- $(c\,x)[x \leftarrow a] = c\,a$
- $(\lambda x.x)[x \leftarrow a] = \lambda x.x$
- Capture avoiding
  $(\lambda y.x\,y)[x \leftarrow y] = \lambda y'.z\,y'$

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

- Instead of using pattern matching, we may introduce *elimination constants*.
- These are special cases of pattern matching.
- *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

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

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

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

lem using elimination constants

$$\mathrm{lem} \in A \vee (B \wedge C) \to (A \vee B) \wedge (A \vee C)$$

where

$$\mathrm{lem}\, p \;=\; \mathrm{case}\, p\, (\lambda x.(\mathrm{inl}\, x, \mathrm{inl}\, x))\, (\lambda y\, (\mathrm{inr}\, (\mathrm{fst}\, y), \mathrm{inr}\, (\mathrm{snd}\, y)))$$

Example: commutativity of $\vee$

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

where

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

False, True

How to form ?

$$\frac{}{\mathrm{False}, \mathrm{True} \in \mathrm{Prop}}$$

How to prove ?

$$\frac{}{\mathrm{triv} \in \mathrm{True}}$$

There is no canonical proof for False*!*

Pattern matching for $\mathrm{True}, \mathrm{False}$

- 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 $\mathrm{True}, \mathrm{False}$

There is no elimination constant for True

$$\frac{p \in \mathrm{False}}{\mathrm{caseF}\; p \in A}$$

where

empty pattern

Defined connective

$$\begin{array}{rcl} A \leftrightarrow B & = & (A \to B) \wedge (B \to A) \\ \neg A & = & A \to \mathrm{False} \end{array}$$

Example: associativity of $\wedge$

$$\begin{array}{ll} \mathrm{andAss} & \in (A \wedge B) \wedge C \leftrightarrow A \wedge (B \wedge C) \\ \mathrm{andAssL} & \in (A \wedge B) \wedge C \to A \wedge (B \wedge C) \\ \mathrm{andAssR} & \in A \wedge (B \wedge C) \to (A \wedge B) \wedge C \end{array}$$

where

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

Predicate logic

- Typed predicate logic!
- We introduce the judgements

  $S \in \mathrm{Type}$ $\quad$ S is a type

  $s \in S$ $\quad$ s is an element of $S \in \mathrm{Type}$

  $P \in S_1 \to S_2 \to \ldots S_n \to \mathrm{Prop}$

  P is a predicate over $S_1, \ldots, S_n \in \mathrm{Type}$

Predicate logic

- $\mathrm{Type} \approx \mathrm{Set}$
- Defining predicates

$$\frac{\begin{array}{c} x_1 \in S_1 \ldots x_n \in S_n \\ \vdots \\ A \in \mathrm{Prop} \end{array}}{\lambda x_1.\lambda x_2.\ldots.\lambda x_n.A \in S_1 \to S_2 \to \ldots S_n \to \mathrm{Prop}}$$

- Using predicates

$$\frac{P \in S_1 \to S_2 \to \ldots S_n \to \mathrm{Prop} \quad s_1 \in S_1 \ldots s_n \in S_n}{P\, s_1\, \ldots\, s_n \in \mathrm{Prop}}$$

Universal quantification: $\forall$

How to form?

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

How to prove?

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

Elimination $\forall$

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

where

$$(\lambda x.p)\ s \quad = \quad p[x \leftarrow s]$$

allAnd

$$\text{allAnd} \in (\forall x \in S.P\ x \wedge Q\ x) \rightarrow (\forall x \in (S.P\ x) \wedge (\forall x \in S.Q\ x)$$

where

$$\text{allAnd}\ f \quad = \quad (\lambda x.\text{fst}\,(f\ x), \lambda x.\text{snd}\,(f\ x))$$

Existential quantification: $\exists$

How to form?

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

How to prove?

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

Pattern matching for $\exists$

- If a pattern variable x has type $\exists y \in S.P$ we can replace it by (x_1,x_2) where $x_1 \in S, x_2 \in P[y \leftarrow x_1]$ are fresh variables.

exOr

$$\mathrm{exOr} \in (\exists x \in S.P\, x \vee Q\, x) \to (\exists x \in S.(P\, x)) \vee (\exists x \in S.Q\, x)$$

$$\begin{array}{rcl} \mathrm{exOr}\,(x,\mathrm{inl}\, p) & = & \mathrm{inl}\,(x,p) \\ \mathrm{exOr}\,(x,\mathrm{inr}\, q) & = & \mathrm{inr}\,(x,q) \end{array}$$

Elimination for $\exists$

$$\frac{\begin{array}{c} x \in S \\ \vdots \\ P \in \mathrm{Prop} \end{array} \qquad A \in \mathrm{Prop} \qquad \begin{array}{c} x \in S \quad y \in P \\ \vdots \\ a \in A \end{array} \qquad p \in \exists x \in S.P}{\mathrm{exElim}\,(\lambda x.\lambda y.a)\,p \in A}$$

where

$$\mathrm{exElim}\,(\lambda x.\lambda y.a)\,(s,p) \quad = \quad a[x \leftarrow s][y \leftarrow p]$$

exOr

$$\mathrm{exOr} \in (\exists x \in S.P\,x \vee Q\,x) \rightarrow (\exists x \in S.(P\,x)) \vee (\exists x \in S.Q\,x)$$

$$\mathrm{exOr}\,y \quad = \quad \mathrm{exElim}\,(\lambda x.\lambda r.\mathrm{case}\,r\,(\lambda p.\mathrm{inl}\,(x,p))\,(\lambda q.\mathrm{inr}\,(x,q))\,y$$

Normalisation

Every closed proof (no free term variables) can be reduced to a canonical proof by systematically applying the equations.

Consequences

Consistency There is no closed proof of False.

Disjunction property If we have a closed proof $p \in A \vee B$ then we can construct a proof $a \in A$ or $b \in B$.

Existence property If we have a closed proof $p \in \exists x \in S.P\,x$ then we can construct an element $s \in S$ and a proof $p \in P\,s$.

Can we prove…

Excluded middle *Tertium non datur* (TND)

$$A \vee \neg A$$

Proof by contradiction *Reductio ad absurdo* (RAA)

$$\neg\neg A \rightarrow A$$

?

$A \vee \neg A$

- 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*: *You cannot create information.*

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.

Intuitionistic vs classical logic

<table>
<tr><th>Classical</th><th>Intuitionistic</th></tr>
<tr><td>$A \vee \neg A$</td><td>$\neg\neg(A \vee \neg A)$</td></tr>
<tr><td>$A \leftrightarrow \neg\neg A$</td><td>$A \rightarrow \neg\neg A$</td></tr>
<tr><td colspan="2">$\neg A \wedge \neg B \leftrightarrow \neg(A \vee B)$</td></tr>
<tr><td>$\neg A \vee \neg B \leftrightarrow \neg(A \wedge B)$</td><td>$\neg A \vee \neg B \rightarrow \neg(A \wedge B)$</td></tr>
<tr><td>$(\exists x \in S. \neg P\, x) \leftrightarrow \neg(\forall x \in S. P\, x)$</td><td>$(\exists x \in S. \neg P\, x) \rightarrow \neg(\forall x \in S. P\, x)$</td></tr>
</table>

Discussion

Is classical logic *better*, because we can prove more?