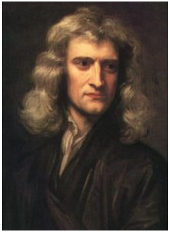
Is Intuitionistic Logic relevant for Computer Science?

Thorsten Altenkirch

School of Computer Science
University of Nottingham

March 3, 2008

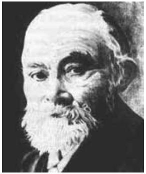
Birth of Modern Mathematics



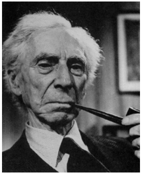
Isaac Newton (1642 - 1727)

1687: Philosophiae Naturalis Principia Mathematica

19/20th century: Foundations?



Frege (1848-1925)



Russell (1872-1970)

$\approx$ 1925: ZF set theory

Zermelo (1871-1953)

Fraenkel (1891-1965)

End of story ?

Overview

$A \wedge (B \vee C) \to (A \wedge B) \vee (A \wedge C)$, classically

A	B	C	$l = A \wedge (B \vee C)$	$r = A \wedge B \vee A \wedge C$	$l \to r$
F	F	F	F	F	T
F	F	T	F	F	T
F	T	F	F	F	T
F	T	T	F	F	T
T	F	F	F	F	T
T	F	T	T	T	T
T	T	F	T	T	T
T	T	T	T	T	T

- The same truth table shows that $A \wedge (B \vee C) \iff (A \wedge B) \vee (A \wedge C)$

BHK: Programs are evidence

Brouwer
(1881-1966)

Heyting
(1898-1980)

Kolmogorov
(1903-1987)

BHK in Haskell

- Evidence for $A \wedge B$ is given by pairs:
  type $A \wedge B = (A, B)$
- Evidence for $A \vee B$ is tagged evidence for A or B.
  data $A \vee B = \mathit{Inl}\ A \mid \mathit{Inr}\ B$
- Evidence for $A \rightarrow B$ is a program computing evidence for B from evidence for A.

$A \wedge (B \vee C) \to (A \wedge B) \vee (A \wedge C)$, constructively

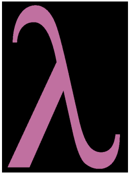
$$f \in A \wedge (B \vee C) \to (A \wedge B) \vee (A \wedge C)$$
$$f\ (a, \mathit{Inl}\ b) = \mathit{Inl}\ (a, b)$$
$$f\ (a, \mathit{Inr}\ c) = \mathit{Inr}\ (a, c)$$

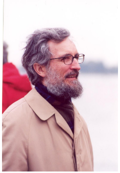
- The program is invertible, because the right hand sides are patterns.
- This shows that the propositions are not only logically equivalent but *isomorphic*.

Predicate logic

- Evidence for $\forall x \in S.P\,x$ is a function f which assigns to each $s \in S$ evidence for $P\,s$.
- Evidence for $\exists x \in S.P\,x$ is a pair (s, p) where $s \in S$ and $p \in P\,s$.
- We need *dependent types*!

Propositions = Types





Curry (1900-1982) Howard (1926-) Martin-Löf (1942-)

Implementations of Type Theory

NUPRL, Coq, Agda, Epigram . . .

$(\exists x \in S.P\,x \vee Q\,x) \to (\exists x \in S.P\,x) \vee (\exists x \in S.Q\,x)$

$$f \in (\exists x \in \mathrm{S}.((P\ x) \vee (Q\ x))) \to (\exists x \in \mathrm{S}.P\ x) \vee (\exists x \in \mathrm{S}.Q\ x)$$
$$f\ (s, \mathit{Inl}\ p) = \mathit{Inl}\ (s, p)$$
$$f\ (s, \mathit{Inr}\ q) = \mathit{Inr}\ (s, q)$$

- Finite explanation
- Logical equivalence, also isomorphism.
- Try to do the same for $(\forall x \in S.P\,x \wedge Q\,x) \to (\forall x : S.P\,x) \wedge (\forall x \in S.Q\,x)$.

$A \vee \neg A$

- We cannot prove $A \vee \neg A$, where $\neg A = A \to \emptyset$, for an undecided proposition A.
- $\forall n \in \mathbb{N}.\mathrm{Prime}\, n \vee \neg \mathrm{Prime}\, n$ is provable, i.e. Prime is *decidable*.
- Indeed, the proof is the program which decides Prime.
- $\forall n \in \mathbb{N}.\mathrm{Halt}\, n \vee \neg \mathrm{Halt}\, n$ is not provable, because Halt is *undecidable*.

Decidability of equality of natural numbers

$$
\begin{array}{llll}
\multicolumn{4}{l}{eq \in \forall m, n \in \mathbb{N}.(m = n) \vee (m \neq n)} \\
eq\ 0 & 0 & = & Inl\ Refl \\
eq\ 0 & (n+1) & = & Inr\ (\lambda p \to \mathbf{case}\ p) \\
eq\ (m+1) & 0 & = & Inr\ (\lambda p \to \mathbf{case}\ p) \\
eq\ (m+1) & (n+1) & = & \mathbf{case}\ eq\ m\ n\ \mathbf{of} \\
 & & & \quad Inl\ Refl \to Inl\ Refl \\
 & & & \quad Inr\ h \quad\ \to Inr\ (\lambda q \to h\ Refl)
\end{array}
$$

- Idealized Agda/Epigram.
- Equality is given by

  $\mathbf{data}\ \_ = \_ \in \mathbb{N} \to \mathbb{N} \to \mathbf{Type}$

  $\mathbf{where}\ Refl \in \forall_{n \in \mathbb{N}} n = n$
- Compare this to

  $eq \in \mathbb{N} \to \mathbb{N} \to \mathbf{Bool}$

The classical Babelfish 

Classical reasoner says:	Babelfish translates to:
$A \vee B$	$\neg(\neg A \wedge \neg B)$
$\exists x : S.Px$	$\neg \forall x : S.\neg Px$

- *Negative translation*
- $A \vee \neg A$ is translated to $\neg(\neg A \wedge \neg\neg A)$ which is constructively provable.
- A classical reasoner is somebody who is unable to say anything positive.

The axiom of choice ?

- Source of non-constructive reasoning ?
- $$\frac{g \in \forall x \in S.\exists y \in T.R\,x\,y}{\mathrm{ac}\,g \in \exists f \in S \to T.\forall x \in S.R\,x\,(f\,x)}\;\mathrm{AC}$$
- Definable in Type Theory:

$$\mathrm{ac}\,g \quad = \quad (\pi_1 \circ g, \pi_2 \circ g)$$

The classical axiom of choice

- $$\frac{\forall x \in S.\exists y \in T.R\,x\,y}{\exists f \in S \to T.\forall x \in S.R\,x\,(f\,x)}\,\mathrm{AC}$$

  $$\frac{\forall x \in S.\neg\forall y \in T.\neg R\,x\,y}{\neg\forall f \in S \to T.\neg\forall x \in S.R\,x\,(f\,x)}\,\mathrm{CAC}$$

- Apply negative translation.
- Not provable constructively:

  $$\begin{aligned} R &\subseteq \mathbb{N} \times \mathrm{Bool} \\ R\,m\,b &= \mathrm{Halts}\,m \iff (b = \mathrm{T}) \end{aligned}$$

- Incompatible with *Church's thesis*: *All functions are computable*

Partial Type Theory ?

- Partial function: a function which may fail to return a result.
- Funtions returning an infinite result (e.g. a stream) are not partial.
- Partial Type Theory is logically inconsistent. $\perp \in \emptyset$.
- Do we actually need partial functions?

A genuinely partial function

$$\textbf{data SK} = \mathrm{S} \mid \mathrm{K} \mid \textbf{SK} :@\ \textbf{SK}$$

$$
\begin{array}{lcl}
nf \in \textbf{SK} \to \textbf{SK} & & \\
nf\ \mathrm{S} & = & \mathrm{S} \\
nf\ \mathrm{K} & = & \mathrm{K} \\
nf\ (t :@\ u) & = & (nf\ t)@(nf\ u)
\end{array}
$$

$$
\begin{array}{lll}
(@) \in \textbf{SK} \to \textbf{SK} \to \textbf{SK} & & \\
\mathrm{K} & @t & = \mathrm{K} :@\ t \\
(\mathrm{K} :@\ t) & @u & = t \\
\mathrm{S} & @t & = \mathrm{S} :@\ t \\
(\mathrm{S} :@\ t) & @u & = (\mathrm{S} :@\ t) :@\ u \\
((\mathrm{S} :@\ t) :@\ u)@v & & = (t@v)@(u@v)
\end{array}
$$

A monad for partiality

- Haskell (a pure functional languages)
  models effects using a monad (the IO monad).
- A monad $\mathrm{M} \in \textbf{Type} \rightarrow \textbf{Type}$ is given by
  $$\mathit{return} \in A \rightarrow \mathrm{M}\ A$$
  $$(\gg\!=) \in (\mathrm{M}\ A) \rightarrow (A \rightarrow \mathrm{M}\ B) \rightarrow \mathrm{M}\ B$$
  subject to some equations.
- We introduce a monad P for partiality.
  (based on joint but yet unpublished work
  with Venanzio Capretta and Tarmo Uustalu).
- Unlike Haskell where IO is opaque, we define P explicitely.

The Delay monad

$$\textbf{codata D } a = \mathrm{Now}\ a \mid \mathrm{Later}\ (\mathbf{D}\ a)$$

instance *Monad* **D where**

$$
\begin{aligned}
&\mathit{return} = \mathrm{Now} \\
&\mathrm{Now}\ a \gg\!= k \ = k\ a \\
&\mathrm{Later}\ d \gg\!= k = \mathrm{Later}\ (d \gg\!= k)
\end{aligned}
$$

$$
\begin{aligned}
&\bot \in \mathbf{D}\ A \\
&\bot = \mathrm{Later}\ \bot
\end{aligned}
$$

Recursion with Delay

$$rec \in ((A \to \mathbf{D}\ B) \to (A \to \mathbf{D}\ B)) \to A \to \mathbf{D}\ B$$

$$\begin{aligned}
& rec\ \phi\ a = aux\ (\lambda _ \to \bot) \\
& \quad \textbf{where}\ aux \in (A \to \mathbf{D}\ B) \to \mathbf{D}\ B \\
& \qquad aux\ k = race\ (k\ a)\ (\mathrm{Later}\ (aux\ (\phi\ k)))
\end{aligned}$$

$$race \in (\mathbf{D}\ A) \to (\mathbf{D}\ A) \to (\mathbf{D}\ A)$$

$$\begin{aligned}
& race\ (\mathrm{Now}\ a)\ \ _ && = \mathrm{Now}\ a \\
& race\ (\mathrm{Later}\ _)\ (\mathrm{Now}\ a) && = \mathrm{Now}\ a \\
& race\ (\mathrm{Later}\ d)\ (\mathrm{Later}\ d') && = \mathrm{Later}\ (race\ d\ d')
\end{aligned}$$

From Delay to Partial

- $\mathbf{D}$ is too intensional...
- We can observe how fast a function terminates.
- Hence $\mathit{rec}\ f \neq f\ (\mathit{rec}\ f)$
- We define
$$\mathbf{P}\ A = \mathbf{D}\ A / \sim$$
where $d \sim d' = \forall a \in A. d \downarrow a \iff d' \downarrow a$
- We have to show that $\gg=$ preserves $\sim$.

Continuity

- $\mathbf{P}\ A$ and hence also $A \to \mathbf{P}\ B$ are ω-CPOs.
- To show that *rec* preserves $\sim$ and that $rec\ f \neq f\ (rec\ f)$ we need that f is ω-continuous.
- All f we can construct have this property!
- Reminiscient of Brouwer's continuity principle:
  All (constructive) functions on $\mathbb{R}$ are continuous.

Type Theory with continuity

- Consider $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$.
- Functions in this type can be given by games:
  $$\textbf{data}\ G = \mathit{Put}\ \mathbb{N} \mid \mathit{Get}\ (\mathbb{N} \to G)$$
- Assign a function to a game:
  $$\mathit{eval} \in G \to (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$$
  $$\begin{aligned}\mathit{eval}\ (\mathit{Put}\ n)\ f &= n\\ \mathit{eval}\ (\mathit{Get}\ h)\ f &= \mathit{eval}\ (h\ (f\ 0))\ (f \circ (+1))\end{aligned}$$
- Identify extensionally equivalent games:
  $g \sim g' \iff \mathit{eval}\ g = \mathit{eval}\ g'$
- Continuity = eval has an inverse:
  $$\mathit{quote} \in ((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \to G/\sim$$

Type Theory with continuity . . .

- Can we interpret all types by games? E.g. $((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \to \mathbb{N}$
- Can we construct a *non-trivial* type **D** such that $D \simeq D \to D$?
- Here *non-trivial* means that there is an injection: $\mathbf{Bool} \to D$.
- Not, that there is a surjection: $D \to \mathbf{Bool}$.

Last slide

- Type Theory is at the same time:
  - A logic
  - A programming language
  - A set theory
- Overcome the *ASCII* - greek dichotomy in Computer Science.
- Applications in natural sciences?