Title	An Introduction to Type Theoretical Ideas
Author(s)	Nordström, Bengt
Citation	
Issue Date	2007-03-06
Type	Presentation
Text version	publisher
URL	http://hdl.handle.net/10119/8293
Rights	
Description	4th VERITE : JAIST/TRUST-AIST/CVS joint workshop on VERIfication TEchnologyでの発表資料，開催：2007年3月6日～3月7日，開催場所：北陸先端科学技術大学院大学・知識講義棟２階中講義室

An Introduction to Type Theoretical Ideas

Bengt Nordström

Computing Science, Chalmers and University of Göteborg

Kanazawa, Japan, March 2007

What is type theory?

A Computer Science Perspective:

It is a precisely defined language to express important parts of programming.

- a programming language (to express programs)
- a specification language (to express the task of the program)
- a programming logic (to express correctness)

A Programmer's Perspective:

Type theory is a

- simple functional language
- with a rich type system (to express specifications)
- and a formal programming logic.

A Logic Perspective:

Type theory is a foundation for (constructive) mathematics.

Why is constructive mathematics relevant for programming?

- computation is fundamental
- function = computable function (= program)
- Proposition = Task / Problem

Classical logic, truth tables

Conjunction

A	B	$A \,\&\, B$
T	T	T
T	F	F
F	T	F
F	F	F

Implication

A	B	$A \supset B$
T	T	T
T	F	F
F	T	T
F	F	T

Disjunction

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

The meaning of proposition is an element in Bool. This assumes that a proposition is either true or false! The meaning of a mathematical statement refers to how things are in a mathematical world.

Example of a classical function

Goldbach's conjecture

Every even number greater than 3 is the sum of two primes.

Nobody knows if this conjecture holds.

A classical function

$$g(n) = \begin{cases} 1 & \text{if Goldbach's conjecture is true,} \\ 0 & \text{otherwise} \end{cases}$$

Is this function computable?

(Classical) example of a classical proof

There exist irrational numbers a and b such that a^b is rational.

We know that $\sqrt{2}^{\sqrt{2}}$ is either rational or irrational.

- In the first case we take $a = b = \sqrt{2}$.
- In the second case we take $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$.

(Classical) example of a classical proof

There exist irrational numbers a and b such that a^b is rational.

We know that $\sqrt{2}^{\sqrt{2}}$ is either rational or irrational.

- In the first case we take $a = b = \sqrt{2}$.
- In the second case we take $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$.

Brouwer

Brouwer rejected the idea that the meaning of a mathematical proposition is its truth value.
Mathematical propositions do not exist independently of us.
We cannot say that a proposition is true without having a proof of it.

Heyting

Heyting was a student of Brouwer. He gave the following explanation of the logical constants.

Heyting's explanation of the logical constants (1930)

A proof of:	consists of:
$A \& B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	a method which takes any proof of A to a proof of B
$\neg A$	a method which takes any proof of A to a proof of absurdity
$\perp$	has no proof
$\exists x \in A.B$	an element a in A and a proof of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof of $B[x := y]$

Heyting's explanation of the logical constants (1930)

A proof of:	consists of:
$A \& B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	a method which takes any proof of A to a proof of B
$\neg A$	a method which takes any proof of A to a proof of absurdity
$\perp$	has no proof
$\exists x \in A.B$	an element a in A and a proof of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof of $B[x := y]$

Heyting's explanation of the logical constants (1930)

A proof of:	consists of:
$A \,\&\, B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	a method which takes any proof of A to a proof of B
$\neg A$	a method which takes any proof of A to a proof of absurdity
$\perp$	has no proof
$\exists x \in A.B$	an element a in A and a proof of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof of $B[x := y]$

Heyting's explanation of the logical constants (1930)

A proof of:	consists of:
$A \,\&\, B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	a method which takes any proof of A to a proof of B
$\neg A$	a method which takes any proof of A to a proof of absurdity
$\perp$	has no proof
$\exists x \in A.B$	an element a in A and a proof of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof of $B[x := y]$

Heyting's explanation of the logical constants (1930)

A proof of:	consists of:
$A \,\&\, B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	a method which takes any proof of A to a proof of B
$\neg A$	a method which takes any proof of A to a proof of absurdity
$\perp$	has no proof
$\exists x \in A.B$	an element a in A and a proof of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof of $B[x := y]$

Heyting's explanation of the logical constants (1930)

A proof of:	consists of:
$A \& B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	a method which takes any proof of A to a proof of B
$\neg A$	a method which takes any proof of A to a proof of absurdity
$\perp$	has no proof
$\exists x \in A.B$	an element a in A and a proof of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof of $B[x := y]$

Heyting's explanation of the logical constants (1930)

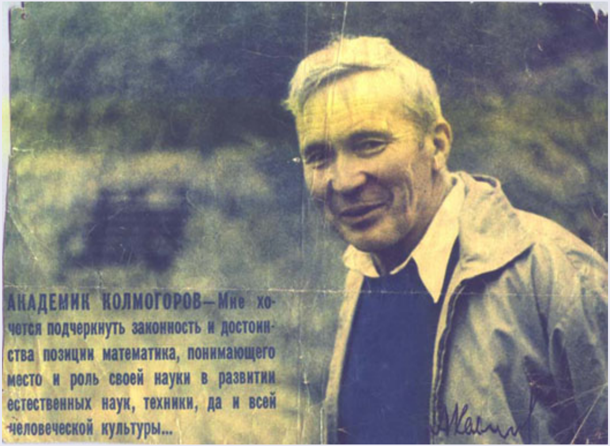
A proof of:	consists of:
$A \,\&\, B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	a method which takes any proof of A to a proof of B
$\neg A$	a method which takes any proof of A to a proof of absurdity
$\perp$	has no proof
$\exists x \in A.B$	an element a in A and a proof of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof of $B[x := y]$

Heyting's explanation of the logical constants (1930)

A proof of:	consists of:
$A \,\&\, B$	a proof of A and a proof of B
$A \vee B$	a proof of A or a proof of B
$A \supset B$	a method which takes any proof of A to a proof of B
$\neg A$	a method which takes any proof of A to a proof of absurdity
$\perp$	has no proof
$\exists x \in A.B$	an element a in A and a proof of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof of $B[x := y]$

Kolmogorov

Independently of Heyting, Kolmogorov interpreted propositions as problems.



Kolmogorov understood the logical constants as problems (1932)

The problem:	is solved if we can:
$A \& B$	solve A and solve B
$A \vee B$	solve A or solve B
$A \supset B$	reduce the solution of B to the solution of A
$\neg A$	show that there is no solution of A
$\perp$	has no solution

Kolmogorov understood the logical constants as problems (1932)

The problem:	is solved if we can:
$A \& B$	solve A and solve B
$A \vee B$	solve A or solve B
$A \supset B$	reduce the solution of B to the solution of A
$\neg A$	show that there is no solution of A
$\bot$	has no solution

Kolmogorov understood the logical constants as problems (1932)

The problem:	is solved if we can:
$A \& B$	solve A and solve B
$A \vee B$	solve A or solve B
$A \supset B$	reduce the solution of B to the solution of A
$\neg A$	show that there is no solution of A
$\perp$	has no solution

Kolmogorov understood the logical constants as problems (1932)

The problem:	is solved if we can:
$A \& B$	solve A and solve B
$A \vee B$	solve A or solve B
$A \supset B$	reduce the solution of B to the solution of A
$\neg A$	show that there is no solution of A
$\bot$	has no solution

Kolmogorov understood the logical constants as problems (1932)

The problem:	is solved if we can:
$A \,\&\, B$	solve A and solve B
$A \vee B$	solve A or solve B
$A \supset B$	reduce the solution of B to the solution of A
$\neg A$	show that there is no solution of A
$\perp$	has no solution

Kolmogorov understood the logical constants as problems (1932)

The problem:	is solved if we can:
$A \& B$	solve A and solve B
$A \vee B$	solve A or solve B
$A \supset B$	reduce the solution of B to the solution of A
$\neg A$	show that there is no solution of A
$\perp$	has no solution

Heyting's and Kolmogorov's explanation

A proof (solution) of:	consists of:
$A \,\&\, B$	a proof (solution) of A and a proof (solution) of B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$\neg A$	a method which takes any proof (solution) of A to a proof (solution) of absurdity
$\perp$	has no proof (solution)
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Question:

Is this correct? Could not a proof (solution) of $A \,\&\, B$ be obtained by induction, or modus ponens, or some other elmination rule?

Heyting's and Kolmogorov's explanation

A proof (solution) of:	consists of:
$A \& B$	a proof (solution) of A and a proof (solution) of B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$\neg A$	a method which takes any proof (solution) of A to a proof (solution) of absurdity
$\bot$	has no proof (solution)
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Question:

Is this correct? Could not a proof (solution) of $A \& B$ be obtained by induction, or modus ponens, or some other elmination rule?

Impredicativity in the definition of implication?

Dummett (and others) have pointed out that there is some kind of impredicativity in the definition of implication:

Heyting's and Kolmogorov's explanation

A proof (solution) of:	consists of:
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B

The method must take any proof of A, this is some kind of quantification over all proofs, including proofs involving implication.

Direct and indirect proofs

When we say that we have a proof of a proposition, then we mean that we have a method which when computed yields a direct proof of it.
Compare this with mathematics and programming: When we say that $2+4$ and $\mathrm{fst}(<45^2, -9>)$ are natural numbers, then we mean that they can be *computed* to a natural number.

Terminology:

	computed	not computed
object	value	expression
proof	direct	indirect
proof	canonical	non-canonical

Examples of indirect proofs

And-elimination

$$\frac{A \,\&\, B}{A}$$

If we have a proof of $A \,\&\, B$, then we can compute it to a direct proof. This always consists of a proof of A and a proof of B. Hence we may always obtain a proof of A from a proof of $A \,\&\, B$.

Mathematical induction

$$\frac{n \in \mathrm{N} \quad P(0) \quad (\forall n \in \mathrm{N}) P(n) \supset P(\mathrm{succ}(n))}{P(n)}$$

Examples of indirect proofs

And-elimination

$$\frac{A \,\&\, B}{A}$$

If we have a proof of $A \,\&\, B$, then we can compute it to a direct proof. This always consists of a proof of A and a proof of B. Hence we may always obtain a proof of A from a proof of $A \,\&\, B$.

Mathematical induction

$$\frac{n \in \mathrm{N} \quad P(0) \quad (\forall n \in \mathrm{N}) P(n) \supset P(\mathrm{succ}(n))}{P(n)}$$

Curry-Howard

To summarize Heyting's and Kolmogorov's explanations:

What does it mean to understand a proposition?

I understand a *proposition* when I understand what a *direct proof* of it is.

This looks very similar to:

What does it mean to understand a set?

I understand a *set* when I understand what a *canonical element* of it is.

Curry-Howard

To summarize Heyting's and Kolmogorov's explanations:

What does it mean to understand a proposition?

I understand a *proposition* when I understand what a *direct proof* of it is.

This looks very similar to:

What does it mean to understand a set?

I understand a *set* when I understand what a *canonical element* of it is.

Propositions and sets

A proof (element) of:	consists of:
$A \,\&\, B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \rightarrow B$	a method which takes any element in A to an element in B
$\perp$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \,\&\, B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \to B$	a method which takes any element in A to an element in B
$\bot$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \& B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \to B$	a method which takes any element in A to an element in B
$\perp$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \,\&\, B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \rightarrow B$	a method which takes any element in A to an element in B
$\perp$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \,\&\, B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \rightarrow B$	a method which takes any element in A to an element in B
$\perp$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \& B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \rightarrow B$	a method which takes any element in A to an element in B
$\bot$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \& B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \rightarrow B$	a method which takes any element in A to an element in B
$\bot$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \& B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \rightarrow B$	a method which takes any element in A to an element in B
$\perp$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \,\&\, B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \rightarrow B$	a method which takes any element in A to an element in B
$\perp$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

Propositions and sets

A proof (element) of:	consists of:
$A \& B$	a proof (solution) of A and a proof (solution) of B
$A \times B$	an element in A and an element in B
$A \vee B$	a proof (solution) of A or a proof (solution) of B
$A + B$	an element in A or an element in B
$A \supset B$	a method which takes any proof (solution) of A to a proof (solution) of B
$A \to B$	a method which takes any element in A to an element in B
$\perp$	has no proof (solution)
$\emptyset$	has no elements
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\forall x \in A.B$	a method, which takes any element y in A to a proof (solution) of $B[x := y]$

This similarity leads to the

Curry-Howard isomorphism

$$A \,\&\, B = A \times B$$

$$A \vee B = A + B$$

$$A \supset B = A \rightarrow B$$

$$\perp = \emptyset$$

$$\neg A = A \rightarrow \emptyset$$

Curry's contribution

Curry noticed the formal similarity between the axioms of positive implicational logic:

$$A \supset B \supset A$$
$$(A \supset B \supset C) \supset (A \supset B) \supset A \supset C$$

and the type of the basic combinators:

$$K \in A \to B \to A$$
$$S \in (A \to B \to C) \to (A \to B) \to A \to C$$

Modus ponens corresponds to the typing rule for application:

$$\frac{A \supset B \quad A}{A} \qquad \frac{f \in A \to B \quad a \in A}{f\ a \in B}$$

Proofs as Programs in a functional programming language

A direct proof of:	consists of:	As a type:
$A \vee B$	a proof of A or a proof of B	**data** Or A B $=$ Ori1 A $\vert$ Ori2 B;
$A \,\&\, B$	a proof of A and a proof of B	**data** And A B $=$ Andi A B;
$A \supset B$	a method taking a proof of A to a proof of B	**data** Implies A B $=$ Impi A $\longrightarrow$ B;
Falsity		**data** Falsity $=$;

Constructors are introduction rules

$$\frac{A}{A \vee B} \qquad \mathbf{Ori1} \in A \to A \vee B$$

$$\frac{B}{A \vee B} \qquad \mathbf{Ori2} \in B \to A \vee B$$

$$\frac{A \quad B}{A \,\&\, B} \qquad \mathbf{Andi} \in A \to B \to A \,\&\, B$$

$$\frac{\begin{matrix}[A] \\ B\end{matrix}}{A \supset B} \qquad \mathbf{Impli} \in (A \to B) \to A \supset B$$

Elimination rules can be defined

orel $\in A \vee B \to (A \to C) \to (B \to C) \to C$

$$\frac{A \vee B \quad \begin{matrix}[A]\\ C\end{matrix} \quad \begin{matrix}[B]\\ C\end{matrix}}{C}\ \textbf{orel}$$

orel $(\textbf{Ori1}\ a)\ f\ g = f\ a$

orel $(\textbf{Ori2}\ b)\ f\ g = g\ b$

andel $\in A \,\&\, B \to (A \to B \to C) \to C$

$$\frac{A \,\&\, B \quad \begin{matrix}[A, B]\\ C\end{matrix}}{C}\ \textbf{andel}$$

andel $(\textbf{Andi}\ a\ b)\ f = f\ a\ b$

implel $\in A \supset B \to A \to B$

$$\frac{A \supset B \quad A}{B}\ \textbf{implel}$$

implel $(\textbf{Impli}\ f)\ a = f\ a$

Elimination rules can be defined

orel $\in A \vee B \to (A \to C) \to (B \to C) \to C$

$$\frac{A \vee B \quad \begin{matrix}[A] \\ C\end{matrix} \quad \begin{matrix}[B] \\ C\end{matrix}}{C}\ \textbf{orel}$$

orel (**Ori1** a) $f\ g = f\ a$

orel (**Ori2** b) $f\ g = g\ b$

andel $\in A \,\&\, B \to (A \to B \to C) \to C$

$$\frac{A \,\&\, B \quad \begin{matrix}[A, B] \\ C\end{matrix}}{C}\ \textbf{andel}$$

andel (**Andi** $a\ b$) $f = f\ a\ b$

implel $\in A \supset B \to A \to B$

$$\frac{A \supset B \quad A}{B}\ \textbf{implel}$$

implel (**Impli** f) $a = f\ a$

Elimination rules can be defined

orel $\in A \vee B \to (A \to C) \to (B \to C) \to C$

orel $(\textbf{Ori1}\ a)\ f\ g = f\ a$

orel $(\textbf{Ori2}\ b)\ f\ g = g\ b$

$$\frac{A \vee B \quad \begin{matrix}[A] \\ C\end{matrix} \quad \begin{matrix}[B] \\ C\end{matrix}}{C}\ \textbf{orel}$$

andel $\in A \,\&\, B \to (A \to B \to C) \to C$

andel $(\textbf{Andi}\ a\ b)\ f = f\ a\ b$

$$\frac{A \,\&\, B \quad \begin{matrix}[A, B] \\ C\end{matrix}}{C}\ \textbf{andel}$$

implel $\in A \supset B \to A \to B$

implel $(\textbf{Impli}\ f)\ a = f\ a$

$$\frac{A \supset B \quad A}{B}\ \textbf{implel}$$

Proof checking = Type checking

In this way we can prove propositional formulas in a typed functional programming language. The problem of proving for instance

$$(A \,\&\, B) \supset (B \,\&\, A)$$

is then the problem of finding a program in this type. The type checker will check if the proof is correct. In this case, we can use the following program:

```
p = Impli (\x ->
              (andel x
                     (\ y -> \ z ->
                       Andi z y)))
```

Proof checking = Type checking

In this way we can prove propositional formulas in a typed functional programming language. The problem of proving for instance

$$(A \,\&\, B) \supset (B \,\&\, A)$$

is then the problem of finding a program in this type. The type checker will check if the proof is correct. In this case, we can use the following program:

```
p = Impli (\x ->
             (andel x
                   (\ y -> \ z ->
                     Andi z y)))
```

What about the quantifiers?

Propositions and sets

A proof (element) of:	consists of:
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\Sigma x \in A.B$	an element a in A and an element in $B[x := a]$
$\forall x \in A.B$	a method, which takes any element x in A to a proof (solution) of $B[x := a]$
$\Pi x \in A.B$	a method, which takes any element y in A to an element in $B[x := y]$

What about the quantifiers?

Propositions and sets

A proof (element) of:	consists of:
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\Sigma x \in A.B$	an element a in A and an element in $B[x := a]$
$\forall x \in A.B$	a method, which takes any element x in A to a proof (solution) of $B[x := a]$
$\Pi x \in A.B$	a method, which takes any element y in A to an element in $B[x := y]$

What about the quantifiers?

Propositions and sets

A proof (element) of:	consists of:
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\Sigma x \in A.B$	an element a in A and an element in $B[x := a]$
$\forall x \in A.B$	a method, which takes any element x in A to a proof (solution) of $B[x := a]$
$\Pi x \in A.B$	a method, which takes any element y in A to an element in $B[x := y]$

What about the quantifiers?

Propositions and sets

A proof (element) of:	consists of:
$\exists x \in A.B$	an element a in A and a proof (solution) of $B[x := a]$
$\Sigma x \in A.B$	an element a in A and an element in $B[x := a]$
$\forall x \in A.B$	a method, which takes any element x in A to a proof (solution) of $B[x := a]$
$\Pi x \in A.B$	a method, which takes any element y in A to an element in $B[x := y]$

Overview of Martin Löf´s type theory

- Type theory is a small typed functional language with one basic type and two type forming operation.
- It is a **framework** for defining logics.
- A logic is introduced by declarations of new constants.

What types are there?

- Set is a type
- $El(A)$ is a type, if $A \in \mathrm{Set}$.
- $(x \in A) \to B$ is a type, if A is a type and B a family of types for $x \in A$.

What programs are there?

Programs are formed from variables and constants using abstraction and application:

- Application

$$\frac{c \in (x \in A) \to B \quad a \in A}{c\ a \in B[x := a]}$$

- Abstraction

$$\frac{b \in B \quad [x \in A]}{[x]b \in (x \in A) \to B}$$

- constants are either primitive or defined

Constants

There are two kinds of constants:

primitive: (not defined) have a type but no definiens (RHS):

$$\text{identifier} \in \text{Type}$$

defined: have a type and a definiens:

$$\text{identifier} = \text{expr} \in \text{Type}$$

There are two kinds of defined constants:

- explicitly defined
- implicitly defined

Primitive constants

- computes to themselves (i.e. are values).
- constructors in functional languages.
- introduction rules and formation rules in logic
- postulates

Examples:

$$
\begin{array}{rcl}
\mathsf{N} & \in & \mathsf{Set} \\
0 & \in & \mathsf{N} \\
\mathsf{s} & \in & \mathsf{N} \rightarrow \mathsf{N} \\
\& & \in & \mathsf{Set} \rightarrow \mathsf{Set} \rightarrow \mathsf{Set} \\
\&\mathbf{I} & \in & (A \in \mathsf{Set}) \rightarrow (B \in \mathsf{Set}) \rightarrow A \rightarrow B \rightarrow A \,\&\, B \\
\Pi & \in & (A \in \mathsf{Set}) \rightarrow (A \rightarrow \mathsf{Set}) \rightarrow \mathsf{Set} \\
\lambda & \in & (A \in \mathsf{Set}) \rightarrow (B \in A \rightarrow \mathsf{Set}) \rightarrow ((x \in A) \rightarrow B(x)) \rightarrow \\
 & & \Pi(A, B)
\end{array}
$$

Explicitly defined constants

- have a type and a definiens (RHS).
- the definiens is a welltyped expression
- abbreviation
- derived rule in logic.
- names for proofs and theorems in math.

Examples:

$$
\begin{aligned}
2 &\in \mathsf{N} \\
&= \mathsf{succ}(\mathsf{succ}\ 0) \\
\forall(A \in \mathsf{Set})(B \in A \to \mathsf{Set}) &\in \mathsf{Set} \\
&= \Pi\ A\ B \\
+(x \in \mathsf{N})(y \in \mathsf{N}) &\in \mathsf{N} \\
&= \mathsf{natrec}\ [x]\mathsf{N}\ x\ y\ [u, v](\mathsf{succ}\ v) \\
\supset (A \in \mathsf{Set})(B \in \mathsf{Set}) &\in \mathsf{Set} \\
&= \Pi\ A\ [x]B
\end{aligned}
$$

Implicitly defined constants

The definiens (RHS) may contain pattern matching and may contain occurrences of the constant itself. The correctness of the definition must in general be decided outside the system

- Recursively defined programs
- Elimination rules (the step from the definiendum to the definiens is the contraction rule).

Examples:

$$\mathbf{add}(x \in \mathrm{N})(y \in \mathrm{N}) \in \mathrm{N}$$
$$\mathbf{add}\ 0\ y = y$$
$$\mathbf{add}\ (\mathrm{succ}\ u)\ y = \mathrm{succ}\ (\mathbf{add}\ u\ y)$$

$$\begin{aligned}&\&\mathbf{E}(A \in \mathrm{Set})(B \in \mathrm{Set})(C \in A \to B \to \mathrm{Set})\\ &\quad (f \in (x \in A) \to (y \in B) \to C(\&\mathbf{I}\ x\ y))\\ &\quad (z \in A \,\&\, B)\\ &\quad \in C(z)\end{aligned}$$

$$\&\mathbf{E}\ A\ B\ C\ f\ (\&\mathbf{I}\ a\ b) = f\ a\ b$$

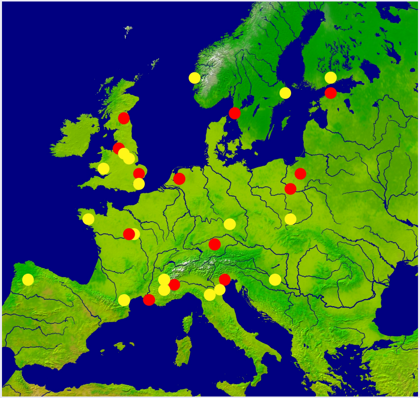
Type theory in Europe

- We had a couple of informal workshops on the Swedish west coast in the '80s.
- The EU funded Types project started in 1989
- The annual Types conference has around 100 participants.

Sites

Main sites:

- Tallinn
- Göteborg
- Edinburgh
- Manchester
- Nijmegen
- London
- Bialystok
- Warsaw
- Paris 7
- Paris Sud
- Munich TUM
- Munich LMU
- Udine
- Torino
- INRIA



Subsites:

- Helsinki
- Bergen
- Stockholm
- Sheffield
- Nottingham
- Birmingham
- Kent
- Swansea
- Krakow
- France Telecom
- Inria Futurs
- Bamberg
- Dassault Aviation
- Novi Sad
- Padova
- Savoie
- Bologna
- Grenoble
- Toulouse
- Minho

Proof editor

A proof editor is a program which lets the user edit a proof of a proposition.

- The user enters a type (a problem)
- The computer checks if it is a propositon
- The user interactively builds an object (proof) of it.

The computer checks all the time that the object is of the given type, i.e. that it proves the given problem.

Important proof editors in the Types project:

- Coq (Paris)
- Lego (Edinburgh)
- Isabelle (Cambridge, Munich)
- Alf, Agda (Göteborg)
- Epigram (Nottingham)

Correctness of the proof editor

An interactive proof checker is a rather complicated program. It contains a lot of complicated code to deal with the interaction with the user. Do we have to trust the entire computer system? An important idea is the idea of *independent checking*:

> We should have a small type checker which checks a complete proof. This type checker will be so small and simple that it is "obviously" correct.

Then we can even use external tools to find proofs, if these tools also produces proof objects in type theory.