Propositions as Types, by Philip Wadler

Víctor López Juan

The presenter

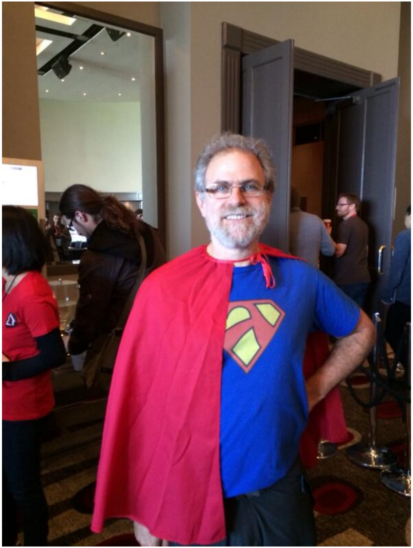
- Víctor López Juan.
- PhD student at Chalmers
- Programing with dependently types.

E-mail/XMPP	victor@lopezjuan.com
IRC	vlopez on `irc.freenode.net`
Website	lopezjuan.com

The author

Philip Wadler

- Haskell
- XQuery for XML
- Java generics
- *Theorems for free!*: parametricity
- Propositions as types



Propositions and Types (I)

Propositions: A statement, which may be true. If we have a proof for a proposition, then we know the proposition is *true*.

- "Six is an even number." That is, $6 = 2 \times n$ for some 'n'.
- "Five is an even number." That is, $5 = 2 \times n$ for some 'n'.
- "Given a list of numbers, there is a list that contains the same elements in ascending order."

Types: A specification that may or may not be possible to implement. If there is a program that implements a type, we say the type is *inhabited*.

- "The program must return a number 'n' such that $2 \times n = 6$."
- "The program must return a number 'n' such that $2 \times n = 5$."
- "The program must take as input a list, and return a sorted list that contains the same elements."

Propositions and Types (II)

Idea: Propositions are types. Proofs are programs.

This is ~~big~~ huge!

- We can write programs that prove theorems.
- We can take a proof and run it as a program.
- We can teach the computer to check our proofs!

But …

- What counts as a proof?
- What counts as a program?
- What language/system should propositions, types, proofs and programs written in?
- How do we avoid contradictions?

Paper structure

1. History
2. A system for proofs: Natural deduction
3. A system for programs: The simply typed λ-calculus
4. The Curry-Howard correspondence.
5. Aliens

Beginnings: Principia Mathematica

$$\ast 2{\cdot}08.\quad \vdash . \, p \supset p$$

Dem.

$$
\begin{array}{lll}
\left[\ast 2{\cdot}05 \dfrac{p \vee p, \; p}{q, \quad r}\right] & \vdash :: p \vee p \, . \supset . \, p : \supset :. \, p \, . \supset . \, p \vee p : \supset . \, p \supset p & (1) \\
[\text{Taut}] & \vdash : p \vee p \, . \supset . \, p & (2) \\
[(1).(2).\ast 1{\cdot}11] & \vdash :. \, p \, . \supset . \, p \vee p : \supset . \, p \supset p & (3) \\
[\ast 2{\cdot}07] & \vdash : p \, . \supset . \, p \vee p & (4) \\
[(3).(4).\ast 1{\cdot}11] & \vdash . \, p \supset p &
\end{array}
$$

Principia Mathematica, *Alfred Whitehead and Bertrand Russel*, 1910

Hilbert's plan

David Hilbert

Hilbert's Programme (1921)

- A formal *system* in which to write proofs.
- A proof that no *contradiction* can be proven in the system.
- A program that, given a proposition, produces a proof in the system.
  ⇒ A system to describe programs that can be run by a computer[a].

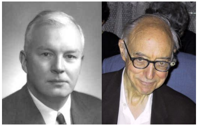
[a] A human computer with pen and paper, that is.

Response

Gerhard Gentzen

Proofs: Natural deduction (1934)

Haskell Curry and William Howard



⇐ Curry-Howard isomorphism ⇒ (1969)

Alonzo Church

Programs: Simply typed lambda calculus (1940)

Natural deduction (6.)

$$\frac{A \qquad B}{A \,\&\, B}\,\&\text{-I} \qquad \frac{A \,\&\, B}{A}\,\&\text{-E}_1 \qquad \frac{A \,\&\, B}{B}\,\&\text{-E}_2$$

$$\frac{\begin{array}{c}[A]^x \\ \vdots \\ B\end{array}}{A \supset B}\,\supset\text{-I}^x \qquad \frac{A \supset B \qquad A}{B}\,\supset\text{-E}$$

Figure 1. Gerhard Gentzen (1935) — Natural Deduction

$$\dfrac{\dfrac{\dfrac{[B \,\&\, A]^z}{A}\,\&\text{-E}_2 \qquad \dfrac{[B \,\&\, A]^z}{B}\,\&\text{-E}_1}{A \,\&\, B}\,\&\text{-I}}{(B \,\&\, A) \supset (A \,\&\, B)}\,\supset\text{-I}^z$$

Figure 2. A proof

Natural deduction — Simplification (6.)

$$\dfrac{\dfrac{\begin{array}{c}\vdots \\ A\end{array} \qquad \begin{array}{c}\vdots \\ B\end{array}}{A \mathbin{\&} B}\;\&\text{-I}}{A}\;\&\text{-E}_1 \quad \Longrightarrow \quad \begin{array}{c}\vdots \\ A\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}[A]^x \\ \vdots \\ B\end{array}}{A \supset B}\;\supset\text{-I}^x \qquad \begin{array}{c}\vdots \\ A\end{array}}{B}\;\supset\text{-E} \quad \Longrightarrow \quad \begin{array}{c}\vdots \\ A \\ \vdots \\ B\end{array}$$

Figure 3. Simplifying proofs

Natural deduction — Example (6.)

$$\dfrac{\dfrac{\dfrac{\dfrac{[B \mathbin{\&} A]^z}{A}\,\&\text{-E}_2 \qquad \dfrac{[B \mathbin{\&} A]^z}{B}\,\&\text{-E}_1}{A \mathbin{\&} B}\,\&\text{-I}}{(B \mathbin{\&} A) \supset (A \mathbin{\&} B)}\,\supset\text{-I}^z \qquad \dfrac{B \quad A}{B \mathbin{\&} A}\,\&\text{-I}}{A \mathbin{\&} B}\,\supset\text{-E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{B \quad A}{B \mathbin{\&} A}\,\&\text{-I}}{A}\,\&\text{-E}_2 \qquad \dfrac{\dfrac{B \quad A}{B \mathbin{\&} A}\,\&\text{-I}}{B}\,\&\text{-E}_1}{A \mathbin{\&} B}\,\&\text{-I}$$

$$\Downarrow$$

$$\dfrac{A \quad B}{A \mathbin{\&} B}\,\&\text{-I}$$

Figure 4. Simplifying a proof

$$\frac{M : A \qquad N : B}{\langle M, N \rangle : A \times B}\ \times\text{-I} \qquad \frac{L : A \times B}{\pi_1\, L : A}\ \times\text{-E}_1 \qquad \frac{L : A \times B}{\pi_2\, L : B}\ \times\text{-E}_2$$

$$\frac{\begin{array}{c}[x : A]^x \\ \vdots \\ N : B\end{array}}{\lambda x.\, N : A \to B}\ \to\text{-I}^x \qquad \frac{L : A \to B \qquad M : A}{L\, M : B}\ \to\text{-E}$$

Figure 5. Alonzo Church (1935) — Lambda Calculus

$$\cfrac{\cfrac{\cfrac{[z : B \times A]^z}{\pi_2\, z : A}\ \times\text{-E}_2 \qquad \cfrac{[z : B \times A]^z}{\pi_1\, z : B}\ \times\text{-E}_1}{\langle \pi_2\, z, \pi_1\, z \rangle : A \times B}\ \times\text{-I}}{\lambda z.\, \langle \pi_2\, z, \pi_1\, z \rangle : (B \times A) \to (A \times B)}\ \to\text{-I}^z$$

Figure 6. A program

The simply-typed lambda calculus — Evaluation (7.)

$$\dfrac{\dfrac{\begin{array}{cc}\vdots & \vdots\\ M : A & N : B\end{array}}{\langle M, N\rangle : A \times B}\times\text{-I}}{\pi_1\,\langle M, N\rangle : A}\times\text{-E}_1 \quad\Longrightarrow\quad \begin{array}{c}\vdots\\ M : A\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}[x : A]^x\\ \vdots\\ N : B\end{array}}{\lambda x.\,N : A \to B}\to\text{-I}^x \qquad \begin{array}{c}\vdots\\ M : A\end{array}}{(\lambda x.\,N)\,M : B}\to\text{-E} \quad\Longrightarrow\quad \begin{array}{c}\vdots\\ M : A\\ \vdots\\ N[M/x] : B\end{array}$$

Figure 7. Evaluating programs

$$\dfrac{\dfrac{\dfrac{\dfrac{[z : B \times A]^z}{\pi_2\, z : A}\times\text{-E}_2 \qquad \dfrac{[z : B \times A]^z}{\pi_1\, z : B}\times\text{-E}_1}{\langle \pi_2\, z, \pi_1\, z \rangle : A \times B}\times\text{-I}}{\lambda z.\, \langle \pi_2\, z, \pi_1\, z \rangle : (B \times A) \to (A \times B)}\to\text{-I}^z \qquad \dfrac{y : B \qquad x : A}{\langle y, x \rangle : B \times A}\times\text{-I}}{(\lambda z.\, \langle \pi_2\, z, \pi_1\, z \rangle)\, \langle y, x \rangle : A \times B}\to\text{-E}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{y : B \qquad x : A}{\langle y, x \rangle : B \times A}\times\text{-I}}{\pi_2\, \langle y, x \rangle : A}\times\text{-E}_2 \qquad \dfrac{\dfrac{y : B \qquad x : A}{\langle y, x \rangle : B \times A}\times\text{-I}}{\pi_1\, \langle y, x \rangle : B}\times\text{-E}_1}{\langle \pi_2\, \langle y, x \rangle, \pi_1\, \langle y, x \rangle \rangle : A \times B}\times\text{-I}$$

$$\Downarrow$$

$$\dfrac{x : A \qquad y : B}{\langle x, y \rangle : A \times B}\times\text{-I}$$

Figure 8. Evaluating a program

The Curry-Howard isomorphism (3. Propositions as types)

Type	Values	$\cong$	Proposition	Canonical proofs
Empty			$\bot$	
Unit	()		$\top$	∎
$A \times B$	(M,N)		$A \mathbin{\&} B$	pf. A and pf. B
$A \to B$	$\lambda\, x \mapsto N$		$A \supset B$	from pf. A, build pf. B
$A + B$	inl M , inr N		$A \vee B$	pf. A or pf. B (and which one)

Normalization and consistency (2. Gentzen, and the theory of proof)

Proposition

(Strong normalization of the simply typed λ-calculus) All programs in the simply-typed lambda calculus ***normalize*** *to a value.*

Proof.

Types and Programming languages, Benjamin C. Pierce, Chapter 12. □

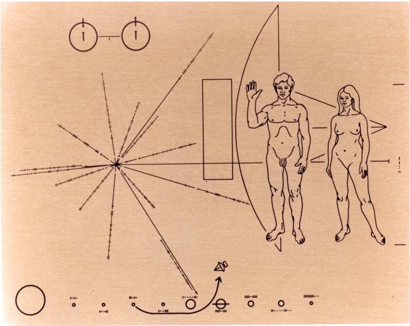
Observation

There are no programs of type Empty in the STλC.

Corollary

(Consistency of natural deduction) There are no proofs of $\perp$ in natural deduction.

Universality of logic (8. Conclusion)



"Scientists imagine that in different universes one might encounter different fundamental constants. […] But […] it is difficult to conceive a universe where the fundamental rules of logic fail to apply.
So we may conclude it would be a mistake to characterise the λ-calculus as a universal language, because calling it universal would be *too limiting*."
— Philip Wadler, Propositions as Types