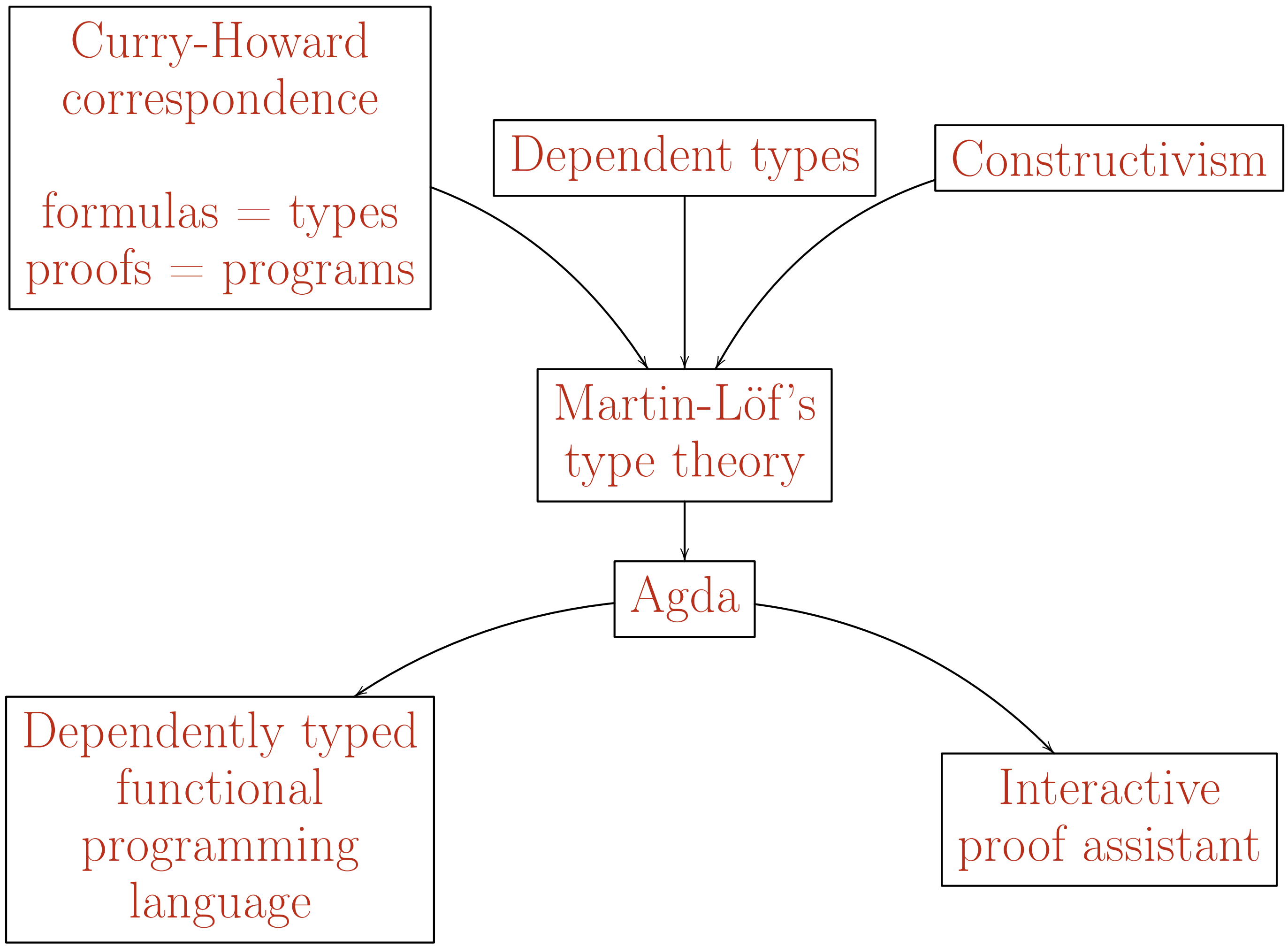
Proofs = Programs

Andrés Sicard-Ramírez
Grupo de Lógica y Computación
Universidad EAFIT

Días de la Ciencia Aplicada
Universidad EAFIT
2009-09-29

Overview


Curry-Howard correspondence
formulas = types
proofs = programs
Dependent types
Constructivism
Martin-Löf's type theory
Agda
Dependently typed functional programming language
Interactive proof assistant

Constructive Interpretation of the Logical Constants

- Proofs are constructions (programs)
- Reject of the principle of the excluded third

The Brouwer-Heyting-Kolmogorov (BHK) interpretation:

A construction of:	Consists of:
$\sigma_1 \wedge \sigma_2$	A construction of σ_1 and a construction of σ_2.
$\sigma_1 \vee \sigma_2$	An indicator $i \in \{1, 2\}$ and a construction of σ_i.
$\sigma_1 \to \sigma_2$	A method (function) which takes any construction of σ_1 to a construction of σ_2.
$\perp$	There is not construction.

Intuitionistic Logic: Fragment $\mathrm{NJ}(\rightarrow)$

Definition (Judgement).

Γ: finite set of formulas

$\Gamma \vdash \sigma$: Γ proves σ

Definition (Rules).

$$\Gamma, \sigma \vdash \sigma \quad (\mathrm{Ax})$$

$$\frac{\Gamma, \sigma \vdash \tau}{\Gamma \vdash \sigma \rightarrow \tau} \ (\rightarrow I) \qquad\qquad \frac{\Gamma \vdash \sigma \rightarrow \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau} \ (\rightarrow E)$$

Example.

$$\frac{\sigma \vdash \sigma}{\vdash \sigma \rightarrow \sigma} \ (\rightarrow I)$$

Intuitionistic Logic: Fragment NJ($\rightarrow$) (cont.)

Convention: The implication is right associative

e.g. $\sigma \rightarrow (\tau \rightarrow \sigma) \equiv \sigma \rightarrow \tau \rightarrow \sigma$

Example.

$$\dfrac{\dfrac{\sigma, \tau \vdash \sigma}{\sigma \vdash \tau \rightarrow \sigma}\,(\rightarrow I)}{\vdash \sigma \rightarrow \tau \rightarrow \sigma}\,(\rightarrow I)$$

Intuitionistic Logic: Fragment $\mathrm{NJ}(\to)$ (cont.)

Example. $\Gamma = \{\sigma \to \tau \to \rho, \sigma \to \tau, \sigma\}$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash \sigma \to \tau \to \rho \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau \to \rho}\,(\to E) \quad \dfrac{\Gamma \vdash \sigma \to \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau}\,(\to E)}{\Gamma \vdash \rho}\,(\to E)}{\sigma \to \tau \to \rho, \sigma \to \tau \vdash \sigma \to \rho}\,(\to I)}{\sigma \to \tau \to \rho \vdash (\sigma \to \tau) \to \sigma \to \rho}\,(\to I)}{\vdash (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho}\,(\to I)$$

Lambda-Calculus

Invented by the American mathematician and logician Alonzo Church (around 1930s).

Intended for studying functions and recursion.

Informally

Element	Example	Denotes
Abstraction	$\lambda x.x^2 + 1$	Function $x \mapsto x^2 + 1$
Application	$(\lambda x.x^2 + 1)\ 3$	Function $x \mapsto x^2 + 1$ applied to 3
β-reduction	$(\lambda x.x^2 + 1)\ 3 \to_\beta 3^2 + 1$	The value of function $x \mapsto x^2 + 1$ applied to 3

Lambda-Calculus (cont.)

Definition (λ-terms).

$$\begin{aligned} \Lambda ::= \;& x && \text{(variable)} \\ \mid\;& \Lambda\Lambda && \text{(application)} \\ \mid\;& \lambda x.\Lambda && \text{(abstraction)} \end{aligned}$$

Definition (β-conversion).

$$(\lambda x.M)N \rightarrow_{1\beta} M[x/N].$$

Definition (β-reduction).

$\rightarrow_{\beta}$: Closure reflexive and transitive of $\rightarrow_{1\beta}$.

Lambda-Calculus (cont.)

Example.

$$
\begin{aligned}
\boldsymbol{I} &\equiv \lambda x.x && \text{(The identity operator)} \\
\boldsymbol{K} &\equiv \lambda xy.x && \text{(The first coordinate projection operator)} \\
\boldsymbol{S} &\equiv \lambda xyz.xz(yz) && \text{(A stronger composition operator)}
\end{aligned}
$$

For all $M, N, O \in \Lambda$,

$$
\begin{aligned}
\boldsymbol{I}M &\to_\beta M \\
\boldsymbol{K}MN &\to_\beta M \\
\boldsymbol{S}MNO &\to_\beta MO(NO)
\end{aligned}
$$

Theorem. In the λ-calculus every (Turing machine)-computable function can be represented by a λ-term (combinator).

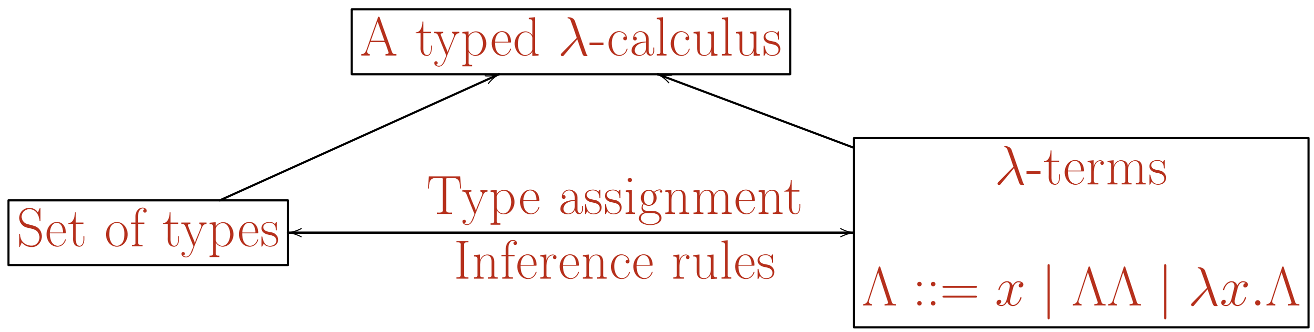
Martin-Löf's Type Theory: Types and Terms

Per Martin-Löf. Swedish logician, philosopher, and mathematician.

Type A	Term $a : A$	
A is a set	a is an element of the set A	$A \neq \emptyset$
A is a proposition	a is a proof (construction) of the proposition A	A is true
A is a problem	a is a method of solving the problem A	A is solvable
A is a specification	a is a program than meets the specification A	A is satisfiable

The Curry-Howard Correspondence: The Simply Typed Lambda-Calculus

The general picture



The Curry-Howard Correspondence: The Simply Typed Lambda-Calculus (cont.)

Definition (Types).

$$
\begin{aligned}
T ::= \ & \sigma & \text{(type variables)} \\
 \mid \ & T \rightarrow T & \text{(function space)}
\end{aligned}
$$

Definition (Context).

Γ: Finite set of pairs of the form $\{x_1 : \tau_1, \ldots, x_n : \tau_n\}$.

$rg(\Gamma) : \{\tau \in T \mid (x : \tau) \in \Gamma, \text{ for some } x\ \}$.

The Curry-Howard Correspondence: The Simply Typed Lambda-Calculus (cont.)

Definition (Judgement ($\Gamma \vdash M : \tau$)).

1. The λ-term M has the type τ in Γ
2. The program M is a proof of the formula τ in Γ

Theorem (Curry-Howard correspondence).

1. If $\Gamma \vdash M : \tau$, then $rg(\Gamma) \vdash \tau$ in $\mathrm{NJ}(\rightarrow)$.
2. If $\Delta \vdash \tau$ in $\mathrm{NJ}(\rightarrow)$, then $\Gamma \vdash M : \tau$ for some M and some Γ with $rg(\Gamma) = \Delta$.

The Curry-Howard Correspondence: The Simply Typed Lambda-Calculus (cont.)

Definition (Rules).

$$\Gamma, \sigma \vdash \sigma \quad (\mathrm{Ax}) \qquad\qquad \Gamma, x : \sigma \vdash x : \sigma \quad (\mathrm{Var})$$

$$\frac{\Gamma, \sigma \vdash \tau}{\Gamma \vdash \sigma \to \tau}\ (\to I) \qquad\qquad \frac{\Gamma, x : \sigma \vdash y : \tau}{\Gamma \vdash \lambda x.y : \sigma \to \tau}\,(\mathrm{Abs})$$

$$\frac{\Gamma \vdash \sigma \to \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau}\ (\to E) \qquad\qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\,(\mathrm{App})$$

The Curry-Howard Correspondence: The Simply Typed Lambda-Calculus (cont.)

Example.

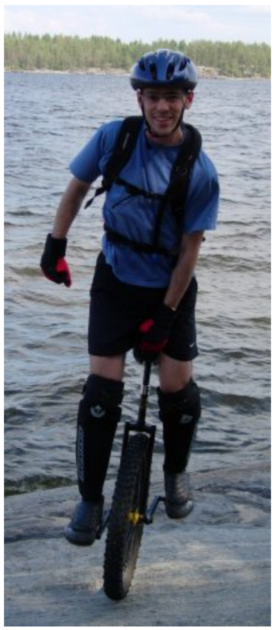
$\Gamma = \{x : \sigma \to \tau \to \rho, y : \sigma \to \tau, z : \sigma\}$

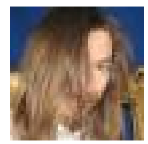
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash x : \sigma \to \tau \to \rho \quad \Gamma \vdash z : \sigma}{\Gamma \vdash xz : \tau \to \rho}\,(\mathrm{App}) \quad \dfrac{\Gamma \vdash y : \sigma \to \tau \quad \Gamma \vdash z : \sigma}{\Gamma \vdash yz : \tau}\,(\mathrm{App})}{\Gamma \vdash xz(yz) : \rho}\,(\mathrm{App})}{x : \sigma \to \tau \to \rho, y : \sigma \to \tau \vdash \lambda z.xz(yz) : \sigma \to \rho}\,(\mathrm{Abs})}{x : \sigma \to \tau \to \rho \vdash \lambda yz.xz(yz) : (\sigma \to \tau) \to \sigma \to \rho}\,(\mathrm{Abs})}{\vdash \lambda xyz.xz(yz) : (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho}\,(\mathrm{Abs})$$

Agda

From: Chalmers University of Technology, Sweden

Agda.cabal.Author: Ulf Norell, Nils Anders Danielsson, Catarina Coquand, Makoto Takeyama, Andreas Abel, ...

Agda as an Interactive Theorem Prover

(From A. Setzer. Interactive Theorem Proving for Agda Users)

Interactive Theorem Proving

- Proofs are fully checked by the system
- Proof steps have to be carried out by the user
- Advantages:
  - Correctness guaranteed (provided the theorem prover is correct)
  - Everything which can be proved by hand, should be possible to be proved in such systems
- Disadvantages:
  - It takes much longer than proving by hand. Similar to programming.
  - Requires experts in theorem proving

Agda as an Interactive Theorem Prover (cont.)

Agda's core: The type Set and the dependent function types $(x : A) \to B$ (Martin-Löf's logical framework.)

Agda code: See file src/eg.agda

Conclusions

logic	typed λ-calculus
formula	type
propositional variable	type variable
implication	function space
proof	λ-term
assumption	object variable
introduction	constructor
elimination	destructor
normal proof	normal form
provability	inhabitation

(M.-H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006, p. 89)

References

- Agda
  - The Agda wiki: `http://wiki.portal.chalmers.se/agda/`
  - A. Bove and P. Dybjer. Dependent types at work. In A. Bove, L. Soares Barbosa, A. Pardo, and J. Sousa Pinto, editors, *LerNet ALFA Summer School 2008*, volume 5520 of *LNCS*, pages 57–99. Springer, 2009.
  - U. Norell. Dependently typed programming in Agda. Eprint: `wiki.portal.chalmers.se/agda/`, 2008.
- Intuitionist logic

  D. van Dalen. *Logic and Structure*. Springer, 4 edition, 2004.

References (cont.)

- Martin-Löf type theories
  - P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
  - B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory*. Oxford University Press, 1990.

References (cont.)

- The Curry-Howard correspondence

  M.-H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006

- The λ-calculus
  - J. R. Hindley and J. Seldin. *Lambda-Calculus and Combinators. An Introduction.* Cambridge University Press, 2008.
  - H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* Elsevier, revised edition, 6th impression edition, 2004.

- The typed λ-calculus

  H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Clarendon Press, 1992.