CM0081 Automata and Formal Languages
Introduction to Agda

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2024-1

Propositions-as-Types Principle

Three correspondence's levels

Wadler [2015] introduces correspondence's levels by:

(i) Propositions-as-types

For each proposition in the logic there is a corresponding type in the programming language—and vice versa.

Propositions-as-Types Principle

Three correspondence's levels

Wadler [2015] introduces correspondence's levels by:

(i) Propositions-as-types

For each proposition in the logic there is a corresponding type in the programming language—and vice versa.

(ii) Proofs-as-programs

For each proof of a given proposition, there is a program of the corresponding type—and vice versa.

Propositions-as-Types Principle

Three correspondence's levels

Wadler [2015] introduces correspondence's levels by:

(i) Propositions-as-types

For each proposition in the logic there is a corresponding type in the programming language—and vice versa.

(ii) Proofs-as-programs

For each proof of a given proposition, there is a program of the corresponding type—and vice versa.

(iii) Simplification of proofs as evaluation of programs

For each way to simplify a proof there is a corresponding way to evaluate a program—and vice versa.

Propositions-as-Types Principle

Other names

- The Curry-Howard correspondence/isomorphism

† [Sørensen and Urzyczyn 2006, p. viii].

Propositions-as-Types Principle

Other names

- The Curry-Howard correspondence/isomorphism
- The Brouwer - Heyting - Kolmogorov - Schönfinkel - Curry - Meredith - Kleene - Feys - Gödel - Läuchli - Kreisel - Tait - Lawvere - Howard - de Bruijn - Scott - Martin-Löf - Girard - Reynolds - Stenlund - Constable - Coquand - Huet - $\cdots$ - correspondence†

† [Sørensen and Urzyczyn 2006, p. viii].

Natural Deduction (Conjunction and Implication)

Preliminaries

- Propositions: $A, B, C, \ldots$
- Judgement: $A\,\mathrm{true}$ (assert, proposition A is true)
- Form of the inference rules

$$\frac{J_1 \quad \ldots \quad J_n}{J}\,\mathrm{rule\ name}$$

  where J (conclusion) and $J_1, \ldots, J_n$ (premises) are all judgements.
- Types of inference rules: Introduction rules and elimination rules

Natural Deduction (Conjunction and Implication)

Inference rules for conjunction

- Introduction rule (composing information)

$$\frac{A\,\mathrm{true} \qquad B\,\mathrm{true}}{A \wedge B\,\mathrm{true}}\ \wedge\mathrm{I}$$

- Elimination rules (retrieving/using information)

$$\frac{A \wedge B\,\mathrm{true}}{A\,\mathrm{true}}\ \wedge\mathrm{E}_1 \qquad\qquad \frac{A \wedge B\,\mathrm{true}}{B\,\mathrm{true}}\ \wedge\mathrm{E}_2$$

Natural Deduction (Conjunction and Implication)

Inference rules for implication

- Introduction rule (hypothetical judgement)

$$\dfrac{\begin{array}{c}[A\,\mathrm{true}]^i\\ \vdots \\ B\,\mathrm{true}\end{array}}{A \supset B\,\mathrm{true}}\ \supset\mathrm{I}^i$$

- Elimination rule (modus ponens)

$$\dfrac{A \supset B\,\mathrm{true} \qquad A\,\mathrm{true}}{B\,\mathrm{true}}\ \supset\mathrm{E}$$

Natural Deduction (Conjunction and Implication)

Example

A proof that $(A \wedge B) \supset (B \wedge A)\,\mathrm{true}$.

$$
\dfrac{\dfrac{\dfrac{[A \wedge B\,\mathrm{true}]^i}{B\,\mathrm{true}}\,\wedge\mathrm{E}_2 \qquad \dfrac{[A \wedge B\,\mathrm{true}]^i}{A\,\mathrm{true}}\,\wedge\mathrm{E}_1}{B \wedge A\,\mathrm{true}}\,\wedge\mathrm{I}}{(A \wedge B) \supset (B \wedge A)\,\mathrm{true}}\,\supset\mathrm{I}^i
$$

Typed Lambda Calculus (Product and Function Types)

- Types

$$
\begin{aligned}
\sigma, \tau ::= &\ \sigma \to \tau && \text{function type} \\
\mid &\ \sigma \times \tau && \text{product type}
\end{aligned}
$$

- λ-terms

$$
\begin{aligned}
M, N ::= &\ x && \text{variable} \\
\mid &\ \lambda x.M && \lambda\text{-abstraction} \\
\mid &\ M\ N && \text{application} \\
\mid &\ \langle M , N \rangle \mid \mathrm{fst}\, M \mid \mathrm{snd}\, M && \text{pairs and projections}
\end{aligned}
$$

- Judgement: $M : \sigma$ (λ-term M has type σ)

Typed Lambda Calculus (Product and Function Types)

Type assignment rules for product types

- Introduction rule (pair formation)

$$\frac{M : \sigma \qquad N : \tau}{\langle M , N \rangle : \sigma \times \tau} \times \mathrm{I}$$

- Elimination rules (pair projections)

$$\frac{M : \sigma \times \tau}{\mathrm{fst}\, M : \sigma} \times \mathrm{E}_1 \qquad\qquad \frac{M : \sigma \times \tau}{\mathrm{snd}\, M : \tau} \times \mathrm{E}_2$$

Typed Lambda Calculus (Product and Function Types)

Type assignment rules for function types

- Introduction rule (λ-abstraction)

$$\dfrac{\begin{array}{c}[x : \sigma]^i \\ \vdots \\ M : \tau\end{array}}{\lambda x.M : \sigma \to \tau} \to\mathrm{I}^i$$

- Elimination rule (application)

$$\dfrac{M : \sigma \to \tau \qquad N : \sigma}{M\,N : \tau} \to\mathrm{E}$$

Typed Lambda Calculus (Product and Function Types)

Example

A proof that $\lambda h.\langle \operatorname{snd} h, \operatorname{fst} h\rangle : (\sigma \times \tau) \to (\tau \times \sigma)$.

$$\dfrac{\dfrac{\dfrac{[h : \sigma \times \tau]^i}{\operatorname{snd} h : \tau}\times\mathrm{E}_2 \qquad \dfrac{[h : \sigma \times \tau]^i}{\operatorname{fst} h : \sigma}\times\mathrm{E}_1}{\langle \operatorname{snd} h, \operatorname{fst} h\rangle : \tau \times \sigma}\times\mathrm{I}}{\lambda h.\langle \operatorname{snd} h, \operatorname{fst} h\rangle : (\sigma \times \tau) \to (\tau \times \sigma)}\to\mathrm{I}^i$$

Correspondence's Levels

Example (propositions as types)

(implication)	$A \supset B$	as	$\sigma \to \tau$	(function type)
(conjunction)	$A \wedge B$	as	$\sigma \times \tau$	(product type)

Example (proofs as programs)

$$\dfrac{\dfrac{\dfrac{[A \wedge B\,\mathrm{true}]^i}{B\,\mathrm{true}}\wedge\mathrm{E}_2 \qquad \dfrac{[A \wedge B\,\mathrm{true}]^i}{A\,\mathrm{true}}\wedge\mathrm{E}_1}{B \wedge A\,\mathrm{true}}\wedge\mathrm{I}}{(A \wedge B) \supset (B \wedge A)\,\mathrm{true}}\supset\mathrm{I}^i$$

Proof

$$\dfrac{\dfrac{\dfrac{[h : \sigma \times \tau]^i}{\mathrm{snd}\,h : \tau}\times\mathrm{E}_2 \qquad \dfrac{[h : \sigma \times \tau]^i}{\mathrm{fst}\,h : \sigma}\times\mathrm{E}_1}{\langle\,\mathrm{snd}\,h\,,\ \mathrm{fst}\,h\,\rangle : \tau \times \sigma}\times\mathrm{I}}{\lambda\,h.\,\langle\,\mathrm{snd}\,h\,,\ \mathrm{fst}\,h\,\rangle : (\sigma \times \tau) \to (\tau \times \sigma)}\to\mathrm{I}^i$$

Program

Proof Assistants

Description

'Proof assistants are computer systems that allow a user to do mathematics on a computer, but not so much the computing (numerical or symbolical) aspect of mathematics but the aspects of proving and defining. So a user can set up a mathematical theory, define properties and do logical reasoning with them.' [Geuvers 2009, p. 3]

Example

- Based on set theory: ISABELLE/ZFC, METAMATH and MIZAR
- Based on higher-order logic: HOL4, HOL LIGHT and ISABELLE/HOL
- Bases on type theories: AGDA, COQ and LEAN.

Agda

What is Agda?

- Dependently typed functional programming language
- Dependently typed interactive proof assistant

Agda

Long trandition: The ALF/Agda family (Gothenburg - Sweden)

- ALF
- Agda
- Alfa. Graphical interface for Agda.
- AgdaLight. Experimental version of Agda.
- Agda 2
  - Based on Martin-Löf Type Theory (also known as Constructive Type Theory or Intuitionistic Type Theory).
  - Direct manipulation of proofs-objects.
  - Backends to Haskell and JavaScript.
  - Written in Haskell.
  - Interaction via Emacs.

Further Reading

Propositions-as-types principle

- Wadler [2015]. Propositions as Types.
- Sørensen and Urzyczyn [2006]. Lectures on the Curry-Howard Isomorphism.

Agda

- Bove and Dybjer [2009]. Dependent Types at Work.
- Norell [2009]. Dependently Typed Programming in Agda.
- Stump [2016]. Verified Functional Programming in Agda.

Demo

References

Bove, A. and Dybjer, P. (2009). Dependent Types at Work. In: LerNet ALFA Summer School 2008. Ed. by Bove, A., Soares Barbosa, L., Pardo, A. and Sousa Pinto, J. Vol. 5520. Lecture Notes in Computer Science. Springer, pp. 57–99. DOI: 10.1007/978-3-642-03153-3_2 (cit. on p. 20).

Geuvers, H. (2009). Proof Assistants: History, Ideas and Future. Sadhana 34.1, pp. 3–25. DOI: 10.1007/s12046-009-0001-5 (cit. on p. 17).

Norell, U. (2009). Dependently Typed Programming in Agda. In: Advanced Functional Programming (AFP 2008). Ed. by Koopman, P., Plasmeijer, R. and Swierstra, D. Vol. 5832. Lecture Notes in Computer Science. Springer, pp. 230–266. DOI: 10.1007/978-3-642-04652-0_5 (cit. on p. 20).

Sørensen, M.-H. and Urzyczyn, P. (2006). Lectures on the Curry-Howard Isomorphism. Vol. 149. Studies in Logic and the Foundations of Mathematics. Elsevier (cit. on pp. 5, 6, 20).

Stump, A. (2016). Verified Functional Programming in Agda. ACM and Morgan & Claypool. DOI: 10.1145/2841316 (cit. on p. 20).

References

Wadler, P. (2015). Propositions as Types. Communications of the ACM 58.12, pp. 75–84. DOI: 10.1145/2699407 (cit. on pp. 2–4, 20).