An Introduction to the Curry-Howard Correspondence

Nick Benton
Microsoft Research, Cambridge
nick@microsoft.com

What's it all about?

- AKA *Curry-Howard Isomorphism* or *Propositions-as-Types Analogy*.

- A close connection between logic and type systems:
  - propositions $\cong$ types
  - proofs $\cong$ programs
  - proof normalization $\cong$ operational semantics, or
  - equality of proofs $\cong$ program equivalence

- Amazingly beautiful, powerful and robust mathematical idea.

- Has proved useful in designing and reasoning about programming languages, compiler intermediate languages, static analyses, program extraction, complexity theory, . . .

Natural Deduction Intuitionistic Propositional Logic

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \supset B}\ (\supset_{\mathcal{I}}) \qquad \frac{\begin{array}{cc}\vdots & \vdots \\ A \supset B & A\end{array}}{B}\ (\supset_{\mathcal{E}})$$

$$\frac{\begin{array}{cc}\vdots & \vdots \\ A & B\end{array}}{A \wedge B}\ (\wedge_{\mathcal{I}}) \qquad \frac{\begin{array}{c}\vdots \\ A \wedge B\end{array}}{A}\ (\wedge_{\mathcal{E}}) \qquad \frac{\begin{array}{c}\vdots \\ A \wedge B\end{array}}{B}\ (\wedge_{\mathcal{E}})$$

$$\frac{\begin{array}{c}\vdots \\ A\end{array}}{A \vee B}\ (\vee_{\mathcal{I}}) \qquad \frac{\begin{array}{c}\vdots \\ B\end{array}}{A \vee B}\ (\vee_{\mathcal{I}}) \qquad \frac{\begin{array}{ccc} & [A] & [B] \\ \vdots & \vdots & \vdots \\ A \vee B & C & C\end{array}}{C}\ (\vee_{\mathcal{E}})$$

'Sequent Style' Natural Deduction

$$\Gamma, A \vdash A \ (ax)$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \ (\supset_{\mathcal{I}}) \qquad \frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B} \ (\supset_{\mathcal{E}})$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \ (\wedge_{\mathcal{I}}) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \ (\wedge_{\mathcal{E}}) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \ (\wedge_{\mathcal{E}})$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \ (\vee_{\mathcal{I}}) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B} \ (\vee_{\mathcal{I}}) \qquad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \ (\vee_{\mathcal{E}})$$

Proof Theory 1

- Not just interested in which formulæ are *provable* – want to study the *proofs*; in particular, when two proofs should be considered equivalent.

- For ND, basic notion of equivalence induced by removal of ‘detours’, in which a connective is introduced and then immediately eliminated:

$$\dfrac{\dfrac{\begin{matrix}[A]\\ \vdots \\ B\end{matrix}}{A \supset B}\,(\supset_{\mathcal{I}}) \qquad \begin{matrix}\vdots\\ A\end{matrix}}{B}\,(\supset_{\mathcal{E}}) \qquad \longrightarrow_{\beta} \qquad \begin{matrix}\vdots\\ A\\ \vdots\\ B\end{matrix}$$

(May discharge n assumptions A in $(\supset_{\mathcal{I}})$, then must plug in n copies of the derivation of A.)

Proof Theory 2

- Eliminations followed by introductions induce further equalities, called η-*rules*.

- Some connectives (e.g. $\vee$) also lead to 'rearranging' equations, called *commuting conversions*.

- Theorems about the rewriting systems (β,$\beta\eta$,βc,$\beta\eta c$):
  - They're Church-Rosser (diamond property).
  - They're Strongly Normalising.
  - Hence, equality is decidable

Recap with suggestive spaces . . .

$$\Gamma, \quad A \vdash \quad A$$

$$\frac{\Gamma, \quad A \vdash \qquad B}{\Gamma \vdash \qquad\qquad A \supset B} \qquad \frac{\Gamma \vdash \qquad A \supset B \quad \Gamma \vdash \qquad A}{\Gamma \vdash \qquad B}$$

$$\frac{\Gamma \vdash \qquad A \quad \Gamma \vdash \qquad B}{\Gamma \vdash \qquad\qquad A \wedge B} \qquad \frac{\Gamma \vdash \qquad A \wedge B}{\Gamma \vdash \qquad A} \qquad \frac{\Gamma \vdash \qquad A \wedge B}{\Gamma \vdash \qquad B}$$

$$\frac{\Gamma \vdash \qquad A}{\Gamma \vdash \qquad\qquad A \vee B} \qquad \frac{\Gamma \vdash \qquad B}{\Gamma \vdash \qquad\qquad A \vee B}$$

$$\frac{\Gamma \vdash \qquad A \vee B \quad \Gamma, \qquad A \vdash \qquad C \quad \Gamma, \qquad B \vdash \qquad C}{\Gamma \vdash \qquad\qquad\qquad\qquad\qquad\qquad C}$$

. . . change notation slightly . . .

$$\Gamma, \quad A \vdash \quad A$$

$$\frac{\Gamma, \quad A \vdash \quad B}{\Gamma \vdash \quad A \to B} \qquad \frac{\Gamma \vdash \quad A \to B \quad \Gamma \vdash \quad A}{\Gamma \vdash \quad B}$$

$$\frac{\Gamma \vdash \quad A \quad \Gamma \vdash \quad B}{\Gamma \vdash \quad A \times B} \qquad \frac{\Gamma \vdash \quad A \times B}{\Gamma \vdash \quad A} \qquad \frac{\Gamma \vdash \quad A \times B}{\Gamma \vdash \quad B}$$

$$\frac{\Gamma \vdash \quad A}{\Gamma \vdash \quad A + B} \qquad \frac{\Gamma \vdash \quad B}{\Gamma \vdash \quad A + B}$$

$$\frac{\Gamma \vdash \quad A + B \quad \Gamma, \quad A \vdash \quad C \quad \Gamma, \quad B \vdash \quad C}{\Gamma \vdash \quad C}$$

. . . just add terms to get Simply Typed λ-Calculus!

$$\Gamma, x : A \vdash x : A$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (\lambda x : A.M) : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M\,N : B}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash (M, N) : A \times B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1 M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2 M : B}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathrm{in}_1 M : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \mathrm{in}_2 M : A + B}$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x_1 : A \vdash N_1 : C \quad \Gamma, x_2 : B \vdash N_2 : C}{\Gamma \vdash \text{case } M \text{ of } \mathrm{in}_1 x_1 \Rightarrow N_1 \mid \mathrm{in}_2 x_2 \Rightarrow N_2 : C}$$

Induced Equations on Terms

$$\begin{array}{rcl}
(\lambda x : A.M)\,N & \rightarrow_\beta & M[N/x] \\
\pi_i(M_1, M_2) & \rightarrow_\beta & M_i \\
\text{case in}_i M \text{ of in}_1 x_1 \Rightarrow N_1 \mid \text{in}_2 x_2 \Rightarrow N_2 & \rightarrow_\beta & N_i[M/x_i]
\end{array}$$

$$\begin{array}{rcl}
\lambda x : A.\ M\ x & \rightarrow_\eta & M \\
(\pi_1 M, \pi_2 M) & \rightarrow_\eta & M \\
\text{case } M \text{ of in}_1 x_1 \Rightarrow \text{in}_1 x_1 \mid \text{in}_2 x_2 \Rightarrow \text{in}_2 x_2 & \rightarrow_\eta & M
\end{array}$$

Proof Theory 3

Can also present the logic in two other natural ways:

- Sequent Calculus (left and right rules and cut) together with cut-elimination theorem corresponds to a kind of λ-calculus with pattern-matching.

- Axiomatic (Hilbert) System (axioms plus *modus ponens*) and its normalisation procedure corresponds to combinators.

There's also a class of denotational models (Cartesian Closed Categories with finite coproducts) which exactly model both the logic and the term calculus.

So . . .

We get a perfect three-way correspondence between

- Intuitionistic Propositional Logic (in 3 (essentially) equivalent forms): propositions, proofs, equality of proofs

- Simply Typed λ-calculus: types, terms, equality of terms

- Bicartesian Closed Categories: objects, morphisms, equality of morphisms

So What?

Each of the three corners of the triangle has its own intuitions, aesthetics (criteria for success) and proof techniques.

So we can take a new logic and see what that corresponds to in terms of programming language, or try to come up with a well-behaved logic which corresponds to some programming language feature.

Proof techniques developed for proving things about logics have been applied to programming languages (e.g. reducibility, forms of Kripke semantics).

Higher-Order Propositional Logics

- Second order propositional intuitionistic logic (with quantification over propositions) corresponds to the Girard-Reynolds polymorphic lambda calculus.
  - Models polymorphism in programming languages like Haskell, SML (actually used inside compilers).
  - Allows encoding of inductive datatypes like lists and trees (Ponder even used these encodings directly).

- Abstract types modelled by existential quantification.

- Slightly fancier logics give rise to richer type systems, such as System F_ω which has been used to study module systems and object-oriented programming.

Types as Specifications

Ordinary types can clearly be seen as rather weak specifications.

Richer type theories (e.g. Calculus of Constructions, Martin Löf Type Theory) allow more precise specifications to be expressed as types. Reading these as propositions, a (constructive) proof of the proposition allows a program meeting the specification to be extracted automatically.

So (roughly), a proof of $\forall x \in A.\exists y \in B.\phi(x, y)$ allows the extraction of a program $f : A \to B$ such that $\forall x \in A.\phi(x, f(x))$.

Making this work well is hard: lots of work but progress seems quite slow and the logical aspect sometimes gets a bit lost.

Classical Logic

Ordinary classical logic seems to have no constructive content - the proof theory identifies all proofs of any formula.

But that hasn't stopped people trying to find some, either by translation into known constructive systems or new reformulations of classical logic.

It's got something to do with languages with richer control operators. For example, the double-negation translation of classical logic into intuitionistic logic gives continuation-passing style.

One of the motivations for classical linear logic is that it has all the symmetries of classical logic (involutive negation, de Morgan dualities) but also has constructive content.

Linear Logic

Girard's linear logic allows propositions to be used exactly once and then reintroduces ability to duplicate and discard via modal operators. Arises from careful analysis of structural rules.

- Connections with process calculi
- Game semantics
- Proof nets and optimal reduction and control-flow analysis
- Use of LL to define new computational complexity classes
- Attempts to define linear types systems to control memory usage
- ILL as a metalanguage for denotational semantics (e.g. for CBV functional languages or for action calculi)

Computational Lambda Calculus

- Devised by Moggi as a categorical metalanguage for denotational semantics of languages with effects

- A very useful programming paradigm, e.g. in Haskell

- Turns out to correspond to an intuitionistic modal logic (S4 style possibility) which has been recently discovered in hardware verification and origninally proposed by Curry in 1947.

- Moggi's equational theory is just consequence of the proof theory

- Related language in MLj, working with commuting conversion normal form ('taking the proof theory seriously') which also led to an unusual construct for exception handling.

Staged Computation

- Term calculus for intuitionistic S4 possibility (derived from work on ILL) discovered to be applicable to staged computation.
- Related results for intuitionistic temporal logic with ‘next’
- Improves on previous, more directly programming-motivated systems
- Latest work draws on categorical intuitions from study of LL