Programming = proving?
The Curry-Howard correspondence today

First lecture

The paths to discovery: the Curry-Howard correspondence, 1930–1970

Xavier Leroy

Collège de France

2018-11-21

COLLÈGE
DE FRANCE
1530

The path to Curry-Howard

- The lambda-calculus
  Initially a logic (terms = propositions!), evolved into the first functional programming language.

- Intuitionistic logic
  Birth of the idea of “proof terms” that materialize proofs.

- Simple types
  From Russell’s type theory to simply-typed lambda-calculus.

- Curry, 1957
  Propositions = types; provability = inhabitation
  (in the special case of combinatory logic).

- Howard, 1969–1980
  Propositions = types; proofs = programs;
  cut elimination = reductions.

I

The lambda-calculus

The lambda-notation

Circa 1930, Alonzo Church introduces the notation $\lambda x.\, e$ meaning "the function that maps parameter x to expression e".

Today we write $x \mapsto e$. However, this notation, popularized by Bourbaki, was introduced later.

These notations $\lambda x.\, e$ or $x \mapsto e$ are useful to clarify the way mathematicians talk about functions…

" the function x^2 "	$x \mapsto x^2$ or $\lambda x.\, x^2$
" the function $x^2 + y^2$ "	$(x, y) \mapsto x^2 + y^2$ or $\lambda x.\lambda y.\, x^2 + y^2$
" the family of functions cx^2 "	$c \mapsto (x \mapsto cx^2)$ or $\lambda c.\lambda x.\, cx^2$

Computing with the lambda-notation

Two main computation rules:

α-conversion: renaming a bound variable.

$$\lambda x.\, M =_\alpha \lambda y.\, M\{x \leftarrow y\} \qquad \text{if } y \text{ not free in } M$$

β-conversion: replacing the formal parameter by the actual argument in a function body.

$$(\lambda x.\, M)(N) =_\beta M\{x \leftarrow N\}$$

In ordinary mathematics we write:

Consider the function $f(x) = x + 1$. *We have that* $f(2) = 3$.

The lambda-notation decomposes this computation in multiple steps:

$$f(2) \;=\; (\lambda x.\, x + 1)\,(2) \;=_\beta\; 2 + 1 \;=\; 3$$

Why lambda?

Dana Scott tells that he once wrote Church

> *Dear professor Church: why "lambda"?*

and received the following reply

> *Eeny, meeny, miny, moe.*

Rosser tells that Church progressively simplified the notation $\hat{x}e$ that is used in Russell and Whitehead's *Principia Mathematica*:

$$\hat{x}e \quad \to \quad \wedge xe \quad \to \quad \lambda xe \quad \to \quad \lambda x[e] \quad \to \quad \lambda x.e$$

A lambda to bind them all

In $\lambda x.e$, variable x is said to be "bound" and can be renamed without changing the meaning.

Bound variables appear in various mathematical notations:

$$\{x \mid P(x)\} \qquad \forall x, P(x) \qquad \exists x, P(x)$$

$$\bigcup_{n \in N} A(n) \qquad \prod_{i=1}^{n} i \qquad \sum_{n=0}^{\infty} \frac{1}{n^2}$$

$$\lim_{n \to \infty} u_n \qquad \int_a^b f(x)dx$$

A lambda to bind them all

Church proposes to treat all these notations as operators taking a lambda-abstraction as argument:

$$\begin{aligned}
\forall x, P(x) &\equiv \forall(\lambda x.\, P(x)) \quad \text{(or just } \forall P\text{)} \\
\exists x, P(x) &\equiv \exists(\lambda x.\, P(x)) \\
\textstyle\sum_{n=0}^{\infty} \frac{1}{n^2} &\equiv \mathtt{sum}(0, \infty, \lambda n. \tfrac{1}{n^2}) \\
\textstyle\int_a^b f(x)dx &\equiv \mathtt{integr}(a, b, \lambda x. f(x))
\end{aligned}$$

This way, there remains only one construct that binds variables: lambda-abstraction.

This is the birth of what is known today as Higher-Order Abstract Syntax.

Church's first lambda-calculus (1932, 1933)

A logic, proposed as an alternative to set theory, where

- predicates and propositions are written in lambda-notation;
- predicates can take predicates as arguments (higher-order logic);
- every set A can, therefore, be represented by its membership predicate $x \mapsto x \in A$.

(This is a "propositions = terms" presentation.)

Sets = predicates

Operations over sets can easily be expressed in terms of predicates and predicate applications:

Membership:	$x \in A$	$\equiv$	$A\,x$ (application)
Comprehension:	$\{x \mid P(x)\}$	$\equiv$	P
Empty set:	$\emptyset$	$\equiv$	$\lambda x.\, 0$
Singleton:	$\{x\}$	$\equiv$	$\lambda y.\, x = y$
Binary union:	$A \cup B$	$\equiv$	$\lambda x.\, A\,x \vee B\,x$
General union:	$\bigcup_{x \in A} B(x)$	$\equiv$	$\lambda y.\, \exists x, A\,x \wedge B\,x\,y$

Russell's paradox can be expressed as well:

$\lambda x.\, \neg(x\,x)$	the set of all x such that $x \notin x$ …
$(\lambda x.\, \neg(x\,x))\ (\lambda x.\, \neg(x\,x))$	… belongs to itself?

Church's first lambda-calculus

A. Church, Postulates for the Foundation of Logic, *Annals of Math.* 33(2), 1932.

A. Church, Postulates for the Foundation of Logic, *Annals of Math.* 34(4), 1933.

Terms: $M, N ::= x$ variable

$\mid \lambda x.\, M$ function abstraction

$\mid M\ N$ function application

$\mid \Pi(M, N)$ $\forall x, M\ x \Rightarrow N\ x$

$\mid \Sigma(M)$ $\exists x, M\ x$

$\mid \&(M, N)$ M and N

$\mid \tilde{}(M)$ not M

$\mid \iota(M)$ an x such that $M\ x$

$\mid A(M, N)$???

+ 5 deduction rules, including α-conversion and β-conversion

+ 37 axioms (36 in version 2).

Logical inconsistency

Russell's paradox can be expressed: taking $U = (\lambda x.\ \neg(x\ x))\ (\lambda x.\ \neg(x\ x))$, we have $U =_\beta \neg U$. This does not cause inconsistency because Church's logic is a minimal intuitionistic logic.

Logical inconsistency is proved by:

- Kleene and Rosser, 1935.
  A complex encoding of Richard's paradox.
- Curry, 1942.
  Let A be a proposition. Consider $D = \lambda x.\ x\ x \Rightarrow A$ and $U = D\ D$.
  Then $U \Leftrightarrow (U \Rightarrow A)$, which implies A.

This is the end of (untyped) lambda-calculus as a logic.
But other applications appear...

The pure lambda-calculus

Circa 1935, Church and his students Kleene and Rosser study the "pure" lambda-calculus, without logical connectives:

$$M, N ::= x \mid \lambda x.\, M \mid M\ N$$

α-equivalence and β-reduction (= oriented conversion):

$$\lambda x.M =_\alpha \lambda y.M\{x \leftarrow y\} \quad (y \text{ not free in } M) \qquad (\lambda x.M)\ N \to_\beta M\{x \leftarrow N\}$$

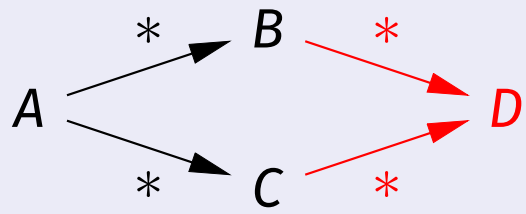
A notion of computation appears: iterate β-reduction until an irreducible "normal form" N is reached.

$$M \to_\beta M_1 \to_\beta M_2 \to_\beta \cdots \to_\beta N \not\to_\beta$$

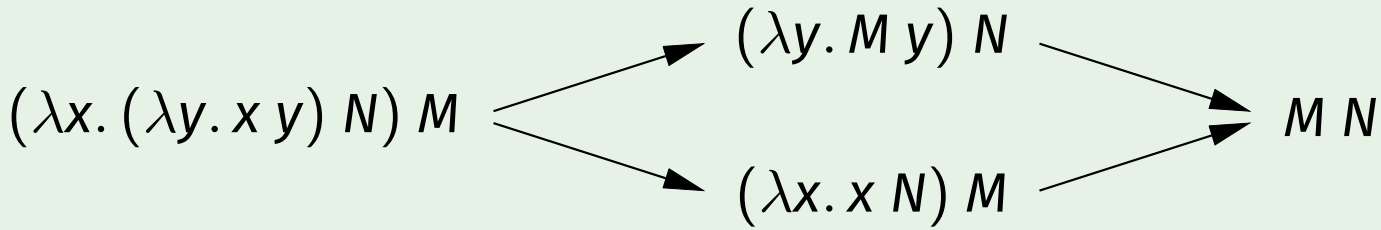
Good properties of reductions

Theorem (Church and Rosser, 1935)

β-reduction is confluent.



Example



Corollary

If it exists, the normal form of a term is uniquely defined.

Lambda-calculus and computability

Church (1936) shows the first undecidability results
(= cannot be computed by a general recursive function):

- Whether a term M has a normal form is undecidable.
- Corollary: Hilbert's decision problem (Entscheidungsproblem) has no solution.

Kleene (1935) and Turing (1937) show equivalences between

- general recursive functions (Herbrand, Gödel, Kleene);
- functions computable by a Turing machine;
- functions definable as a λ-term
  via Church's encoding of natural numbers (1933):

$$n \equiv \lambda f.\, \underbrace{f \circ \cdots \circ f}_{n \text{ times}} \equiv \lambda f.\, \lambda x.\, \underbrace{f\ (f\ (\cdots\ (f\ x)))}_{n \text{ times}}$$

Lambda-calculus and functional languages

The pure lambda-calculus as the common ancestor and as the kernel of functional programming languages:

functional language	=	pure lambda-calculus
	+	reduction strategy
	+	data types
	+	type system (if applicable)

LISP 1.5 (1960) refers to Church and writes `(LAMBDA ...)` for anonymous functions. However, its semantics (dynamic scoping of bindings) does not follow β-reduction.

Landin (1965) proposes ISWIM, an extended lambda-calculus, as the basis for *The next 700 programming languages*.

After 1970, all functional languages contain lambda-calculus as a kernel: Common Lisp, Scheme, SML, Caml, Haskell, ...

II

Intuitionistic logic

Intuitionism and intuitionistic logic

Intuitionism: a philosophy of mathematics developed by L. E. J. Brouwer (1881–1966). Requires all mathematical objects to be accessible to intuition. Rejects non-constructive proofs, actual infinity, and Hilbert's axiomatic approach.

Intuitionistic logic: family of logics studied by Heyting, Glivenko, Gödel, Kolmogorov. Formalizes the "only constructive proofs" aspect of intuitionism.

Constructive proofs vs. non-constructive proofs

How to prove "there exists $x \in A$ such that $P(x)$" ?

1. Constructively: we define an element a of A from the
2. By contradiction: we assume $\exists x \in A.\ P(x)$ false, that is, we assume $\forall x \in A.\ \neg P(x)$ true, and we deduce a contradiction.

For an intuitionist, the first proof is the only valid way to show that $\exists x \in A.\ P(x)$ is true.

The second proof only shows that $\exists x \in A.\ P(x)$ is not absurd, which is a weaker result.

An intuitionistic adventure

(inspired by G. Dowek, *Computation, Proof, Machine*)

In the train back from the Netherlands, someone drops me a note:

> *The French classicist logicians want to kill you. Get off at the last stop before the French border.*
> *Note: the train may make extra unplanned stops.*

At which stop should I get off the train?

The set A of all stops outside France is 1- nonempty (Brussels $\in A$), 2- totally ordered, and 3- finite; hence, it contains a maximal element; this is "my" stop.

However, this proof is not constructive and fails to give me an effective criterion to decide at each stop whether to get off or stay…

Excluded middle

The excluded middle principle (*tertium non datur*, *tiers exclu*):

$$P \vee \neg P \quad \text{for all proposition } P$$

Equivalent to double negation elimination:

$$\neg\neg P \Rightarrow P \quad \text{for all proposition } P$$

This is the principle for proofs by contradiction.

Intuitionistic logic rejects excluded middle (and the many equivalent rules).

Truth vs provability

Classical logic: every proposition is *a priori* true or false.
Mathematical proof establishes which of these two cases apply.

Hence, for example, reasoning by truth tables in the propositional fragment:

P	Q	$P \Rightarrow Q$	$Q \Rightarrow P$	$(P \Rightarrow Q) \vee (Q \Rightarrow P)$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	1	1	1

Truth vs provability

Classical logic: every proposition is *a priori* true or false.
Mathematical proof establishes which of these two cases apply.

Intuitionistic logic: a proposition can be proved;
or its negation can be proved;
otherwise we do not know anything about the proposition.

The BHK interpretation

(Brouwer, Heyting, Kolmogorov)

Provability is characterized by the existence of a construction of the proposition.

- There exists a construction of $\top$ ("true").
- There exists no construction of $\bot$ ("false").
- A construction of $P_1 \wedge P_2$ is a pair (c_1, c_2) where c_1 is a construction of P_1 and c_2 a construction of P_2.
- A construction of $P_1 \vee P_2$ is a pair (i, c) where $i \in \{1, 2\}$ and c is a construction of P_i.

The BHK interpretation

- A construction of $P_1 \Rightarrow P_2$ is a "process" that, given a construction of P_1, produces a construction of P_2.

We can elaborate the word "process" in several ways:

arbitrary mathematical function	(not always constructive)
algorithm	(a very intuitionistic idea!)
recursive function	($\rightarrow$ Kleene's realizability)
lambda-term	($\rightarrow$ Curry-Howard).

The BHK interpretation

- A construction of $\neg P$, treated like $P \Rightarrow \bot$, is an algorithm that, from a (hypothetical) construction of P produces a non-existent object.

Note: the existence of a construction of $\neg P$ is much stronger than the non-existence of a construction of P.

- A construction of $\forall x, P(x)$ is an algorithm that, given a value for x, produces a construction of $P(x)$.
- A construction of $\exists x, P(x)$ is a pair (a, construction of $P(a)$).

Back to excluded middle

Let TM be the set of all Turing machines. Consider

$$Q \stackrel{def}{=} \forall m \in TM, \ \ \text{terminates}(m) \vee \neg\text{terminates}(m)$$

Q follows immediately from excluded middle ($P \vee \neg P$ for all P).

Yet, a construction of Q would be an algorithm that, given a Turing machine m, returns $(1, c)$ if m terminates and $(2, c)$ otherwise.

Since the halting problem is undecidable, such an algorithm does not exist.

Excluded middle is therefore not constructive, since there is no construction that validates it.

(This construction would be a universal decision procedure!)

A formalization of intuitionistic logic

Manipulates intuitionistic sequents of the form $\Gamma \vdash P$
(read: “under hypotheses Γ we can deduce P”.)

Comprises axioms (“this sequent is true”) and inference rules
(“if the sequents above are true, then the sequent below is true”).
For example:

$$P \vdash P$$

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}$$

Rules of intuitionistic logic

Implication:

("I" stands for Introduction, "E" for Elimination)

$$\Gamma_1, P, \Gamma_2 \vdash P \quad (\text{Ax})$$

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q} \ (\Rightarrow\text{I}) \qquad \frac{\Gamma \vdash P \Rightarrow Q \qquad \Gamma \vdash P}{\Gamma \vdash Q} \ (\Rightarrow\text{E, } \textit{modus ponens})$$

Conjunction:

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \ (\wedge\text{I}) \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P} \ (\wedge\text{E}_1) \qquad \frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q} \ (\wedge\text{E}_2)$$

Rules of intuitionistic logic

Disjunction:

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \vee Q}\ (\vee \mathrm{I}_1) \qquad \frac{\Gamma \vdash Q}{\Gamma \vdash P \vee Q}\ (\vee \mathrm{I}_2)$$

$$\frac{\Gamma \vdash P \vee Q \qquad \Gamma, P \vdash R \qquad \Gamma, Q \vdash R}{\Gamma \vdash R}\ (\vee \mathrm{E})$$

Negation:

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash P}\ (\bot \mathrm{E}, \textit{ex falso quodlibet})$$

III

Types

Types

Folk wisdom:

> *Don't compare apples to oranges.*
> *On n'additionne pas des choux et des carottes.*

Physical wisdom: dimensional analysis

$d = v.t$	dist = dist	homogeneous, possibly correct
$d = v/t$	dist $\neq$ dist.time^{-2}	not homogeneous, always false

Types in programming languages

To detect and reject some programming errors.

```
"foo"(12)              (character string ≠ function)
cos(45, "degrees")     (wrong number of arguments)
```

As partial specifications of data structures and module interfaces.

```
type 'a tree = Leaf of 'a | Node of 'a tree * 'a tree
val assoc: 'a -> ('a * 'b) list -> 'b
```

To clarify the meaning of programs and facilitate compilation.

```
integer n
real x
n = n * 2 + 1          (integer arithmetic)
x = x * 2 + 1          (floating-point arithmetic)
```

Types in mathematical logic

In elementary mathematics: an intuitive notion.

For example, in Euclidean geometry, a point is not a line,
and we never talk about the intersection of two points.

In naive set theory: no types *a priori*.

For example, the encoding of natural numbers $n + 1 \stackrel{def}{=} n \cup \{n\}$.

But we get paradoxes such as Russell's: $\{x \mid x \notin x\}$.

Type theories: add types to a logic to rule out paradoxes.

> *Typing is a sort of superego forbidding certain forms of logical incest of the kind $x \in x$.*
>
> *(J.-Y. Girard, The blind spot)*

Russell's type theory

(1908; used in *Principia Mathematica*, 1910–1912)

Russell states a "vicious circle principle":

> *Whatever involves all of a collection must not be one of the collection.*

To this end, he associates an order (a natural number) to each object of the theory, in such a way that

- order of a set $>$ order of the elements of the set (no $x \in x$)
- order of a quantifier $\forall x.P >$ order of variable x (predicativity)

Russell's ramified types

Function types t annotated by natural numbers a:

$$
\begin{array}{lll}
t^a ::= 0^0 & & \text{base propositions} \\
\quad \mid (t_1^{a_1}, \ldots, t_n^{a_n})^{\,a} & & \text{predicate of arity } n \\
& & \text{with } a > \max(a_1, \ldots, a_n)
\end{array}
$$

Informal but complicated typing rules.

Types are too intensional: equivalent logical formulas have different orders.

A reductibility axiom (for every formula F there exists a formula $G \Leftrightarrow F$ whose order is minimal) that fails to convince.

De-ramification efforts (Ramsey (1926), Hilbert and Ackermann, ...) to get rid of orders and keep only simple types (function types).

(Kamareddine, Laan, Nederpelt, *A modern perspective on type theory*, part 1.)

Church's simple theory of types

(A. Church, A Formulation of the Simple Theory of Types, *J. Symbolic Logic* 5(2), 1940.)

A grammar of well-typed terms, indexed by their types.

Types:	$\alpha, \beta ::=$	ι	natural numbers
		$\mid o$	logical propositions
		$\mid \alpha \to \beta$	functions from α to β
Terms:	$M, N ::=$	x_α	variable
		$\mid (\lambda x_\alpha .\, M_\beta)_{\alpha\to\beta}$	abstraction
		$\mid (M_{\alpha\to\beta}\ N_\alpha)_\beta$	application
		$\mid \neg_{o\to o}$	logical negation
		$\mid \wedge_{o\to o\to o}$	conjunction
		$\mid \forall_{(\alpha\to o)\to o}$	universal quantification
		$\mid \iota_{(\alpha\to o)\to\alpha}$	an x such that $P(x)$

Plus: α-conversion, β-conversion.

Plus: an axiomatization of Peano arithmetic.

The simply-typed lambda-calculus

The modern presentation separates the syntax of terms from the typing judgment $\Gamma \vdash M : \alpha$
(read: term M has type α under hypotheses Γ).

Terms: $M, N ::= x \mid \lambda x{:}\alpha.\, M \mid M\ N$

Typing rules:

$$\Gamma_1, x : \alpha, \Gamma_2 \vdash x : \alpha$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x{:}\alpha.\, M : \alpha \to \beta} \qquad \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash M\ N : \beta}$$

Types for the constants used by Church:

$$\neg : o \to o \qquad \forall_\alpha : (\alpha \to o) \to o$$
$$\wedge : o \to o \to o \qquad \iota_\alpha : (\alpha \to o) \to \alpha$$

Types and reductions

Theorem (Subject reduction)

If $\Gamma \vdash M : \alpha$ *and* $M \to_\beta M'$, *then* $\Gamma \vdash M' : \alpha$

Theorem (Strong normalization)

If $\Gamma \vdash M : \alpha$, *all sequences of* β*-reductions starting with* M *are finite.*

Corollary: the simply-typed lambda-calculus is not Turing-complete.

IV

The Curry-Howard correspondence

Combinatory logic

As early as 1930, Curry studies combinatory logic: a variant of the λ-calculus without abstraction $\lambda x.M$, but with combinators predefined by their conversion rules. For example:

Terms: $M, N ::= M\ N \mid S \mid K \mid I$

Rules:
$$\begin{aligned} I\ x &= x \\ K\ x\ y &= x \\ S\ x\ y\ z &= x\ z\ (y\ z) \end{aligned}$$

Every λ-term can be encoded in combinatory logic if a suitable combinator basis is provided, such as S, K, I.

Typing combinatory logic using simple types:

$$\frac{M : \alpha \to \beta \qquad N : \alpha}{M\ N : \beta} \qquad I : \alpha \to \alpha \qquad K : \alpha \to \beta \to \alpha$$

$$S : (\alpha \to \beta \to \gamma) \to (\alpha \to \beta) \to (\alpha \to \gamma)$$

Curry's correspondence

On pages 313–314 of their book *Combinatory logic* (1958), Curry and Feys remark that, if we read the arrow in type $\tau_1 \to \tau_2$ like the implication $P \Rightarrow Q$ between propositions,

- The types for the combinators S, K, I are the axioms that define implication in a Hilbert-style logic:

$$\begin{array}{rcll} I & : & \alpha \to \alpha & P \Rightarrow P \\ K & : & \alpha \to \beta \to \alpha & P \Rightarrow Q \Rightarrow P \\ S & : & (\alpha \to \beta \to \gamma) \to & (P \Rightarrow Q \Rightarrow R) \Rightarrow \\ & & \quad (\alpha \to \beta) \to (\alpha \to \gamma) & \quad (P \Rightarrow Q) \Rightarrow (P \Rightarrow R) \end{array}$$

- The typing rule for function application is *modus ponens*:

$$\frac{M : \alpha \to \beta \qquad N : \alpha}{M\,N : \beta} \qquad\qquad \frac{P \Rightarrow Q \qquad P}{Q}$$

Curry's correspondence

Via Curry's correspondence, a proposition *P* is provable if and only if the corresponding type α is inhabited, that is, there exists a closed term with type α.

Howard's manuscript

W. A. Howard, *The formulae-as-types notion of construction*, 1969

Extends and modernize Curry's result:

- Lambda-calculus instead of combinatory logic.
- Intuitionistic sequents instead of Hilbert-style axioms.
- Treats the connectors $\wedge$, $\vee$, $\neg$ in addition to $\Rightarrow$;
  discusses the quantifiers $\forall$ and $\exists$.
- Correspondence between reduction over terms and cut elimination over proofs.

First circulated as photocopies of hand-written notes.
Eventually published (identically) in 1980 in the book
To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism.

Curry-Howard: implication

$$\Gamma_1, x : A, \Gamma_2 \vdash x : A$$

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

Curry-Howard: implication

$$\overline{\Gamma}_1, \quad A, \overline{\Gamma}_2 \vdash \quad A$$

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

$\overline{\Gamma}$ is Γ without variable names, for example $\overline{x : A, y : A} = A, A$.

Curry-Howard: conjunction

Extend the lambda-calculus with pairs and projections:

Types: $A, B ::= \ldots \mid A \times B$

Terms: $M, N ::= \ldots \mid \langle M, N \rangle \mid \pi_1\, M \mid \pi_2\, M$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : B}{\Gamma \vdash \langle M, N \rangle : A \times B} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_1\, M : A} \qquad \frac{\Gamma \vdash M : A \times B}{\Gamma \vdash \pi_2\, M : B}$$

Curry-Howard: conjunction

Extend the lambda-calculus with pairs and projections:

Types: $A, B ::= \ldots \mid A \times B$

Terms: $M, N ::= \ldots \mid \langle M, N \rangle \mid \pi_1\, M \mid \pi_2\, M$

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

Curry-Howard: disjunction

Extend the lambda-calculus a sum type and the corresponding operations (see lecture 3 on week 2):

Types: $A, B ::= \ldots \mid A + B$

Terms: $M, N ::= \ldots \mid \mathtt{inj}_1\, M \mid \mathtt{inj}_2\, M$
$\mid \mathtt{match}\, M\, \mathtt{with}\, \mathtt{inj}_1\, x_1 \Rightarrow N_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow N_2\, \mathtt{end}$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathtt{inj}_1\, M : A + B} \qquad \frac{\Gamma \vdash N : B}{\Gamma \vdash \mathtt{inj}_2\, N : A + B}$$

$$\frac{\Gamma \vdash M : A + B \quad \Gamma, x_1 : A \vdash N_1 : C \quad \Gamma, x_2 : B \vdash N_2 : C}{\Gamma \vdash \mathtt{match}\, M\, \mathtt{with}\, \mathtt{inj}_1\, x_1 \Rightarrow N_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow N_2\, \mathtt{end} : C}$$

Curry-Howard: disjunction

Extend the lambda-calculus a sum type and the corresponding operations (see lecture 3 on week 2):

Types: $A, B ::= \ldots \mid A + B$

Terms: $M, N ::= \ldots \mid \mathtt{inj}_1\, M \mid \mathtt{inj}_2\, M$
$\mid \mathtt{match}\, M\, \mathtt{with}\, \mathtt{inj}_1\, x_1 \Rightarrow N_1 \mid \mathtt{inj}_2\, x_2 \Rightarrow N_2\, \mathtt{end}$

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

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

Curry-Howard: absurdity

Extend the lambda-calculus with an empty type and an "eliminator" for this type (see lecture 3 on week 2):

Types: $A, B ::= \ldots \mid \bot$

Terms: $M, N ::= \ldots \mid \texttt{match}\ M\ \texttt{with end}$

$$\frac{\Gamma \vdash M : \bot}{\Gamma \vdash \texttt{match}\ M\ \texttt{with end} : A}$$

(The syntax $\texttt{match}\ M\ \texttt{with end}$ is the one used in Coq and comes from the fact that type $\bot$ can be defined as an inductive type with zero constructors.)

Curry-Howard: absurdity

Extend the lambda-calculus with an empty type and an "eliminator" for this type (see lecture 3 on week 2):

Types: $A, B ::= \ldots \mid \bot$

Terms: $M, N ::= \ldots \mid \texttt{match}\ M\ \texttt{with}\ \texttt{end}$

$$\frac{\overline{\Gamma} \vdash \quad \bot}{\overline{\Gamma} \vdash \quad A}$$

(The syntax $\texttt{match}\ M\ \texttt{with}\ \texttt{end}$ is the one used in Coq and comes from the fact that type $\bot$ can be defined as an inductive type with zero constructors.)

What does β-reduction corresponds to?

Howard (1980) notices that β-reduction corresponds to cut elimination in the implicative fragment ($\Rightarrow$) of the logic.

$$\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash M : B}}{\Gamma \vdash \lambda x.\, M : A \to B} \qquad \begin{matrix}\vdots \\ \Gamma \vdash N : A\end{matrix}}{\Gamma \vdash (\lambda x.M)\ N : B}$$

becomes

$$\dfrac{\begin{matrix}\vdots \\ \Gamma \vdash N : A\end{matrix}}{\dfrac{\vdots}{\Gamma \vdash M\{x \leftarrow N\} : B}}$$

What does β-reduction corresponds to?

Howard (1980) notices that β-reduction corresponds to cut elimination in the implicative fragment ($\Rightarrow$) of the logic.

$$\dfrac{\dfrac{\vdots}{\overline{\Gamma}, \quad A \vdash \quad B}}{\overline{\Gamma} \vdash \quad A \Rightarrow B} \qquad \begin{matrix}\vdots \\ \overline{\Gamma} \vdash \quad A\end{matrix}}{\overline{\Gamma} \vdash \quad B}$$

becomes

$$\dfrac{\begin{matrix}\vdots \\ \overline{\Gamma} \vdash \quad A\end{matrix}}{\begin{matrix}\vdots \\ \overline{\Gamma} \vdash \quad B\end{matrix}}$$

Cuts

A *cut* is an intermediate statement (a lemma) that we prove even though it is not a subformula of the final statement (the theorem).

Example (A proof with a cut)

We show P, then $P \Rightarrow Q$, and we conclude Q.
P is a cut.

Example (A proof without cuts)

We show P, then Q, and we conclude $P \wedge Q$.
P and Q are subformulas of $P \wedge Q$ and therefore not cuts.

Cut elimination

Cut elimination = rewrite a proof until it contains no cuts.

A cut-free proof has no mathematical value.
(A nice proof has lemmas and intermediate results!)

Cut elimination helps showing that a logic is consistent (there are no cut-free proofs of “false”).

Cut elimination

For the implicative fragment ($\Rightarrow$) it suffices to transform the derivation as follows:

$$\dfrac{\dfrac{\vdots}{\Gamma, P \vdash Q}}{\Gamma \vdash P \Rightarrow Q}\,(\Rightarrow\mathrm{I}) \qquad \begin{matrix}\vdots \\ \Gamma \vdash P\end{matrix}}{\Gamma \vdash Q}\,(\Rightarrow\mathrm{E})$$

becomes

$$\dfrac{\begin{matrix}\vdots \\ \Gamma \vdash P\end{matrix}}{\begin{matrix}\vdots \\ \Gamma \vdash Q\end{matrix}}$$

Intuition: we used a lemma P to prove a theorem Q. We get rid of P by replacing every use of P in the proof of Q by a copy of the proof of P.

When this transformation cannot be applied any more, the derivation is cut-free and has the subformula property.

Reductions and cut elimination

Howard observes that a term in β-normal form corresponds to a cut-free demonstration in the fragment $(\Rightarrow)$ of the logic.

The result extends to conjunctions.

Reduction rules for products:

$$\pi_1 \langle M, N \rangle \to M \qquad \pi_2 \langle M, N \rangle \to N$$

A term in normal form for these rules and for β correspond to a cut-free demonstration in the fragment $(\Rightarrow, \wedge)$ of the logic.

Reductions and cut elimination

If we add $\vee$ and $\bot$, the result no longer holds: the usual reduction rules for sums

$$\texttt{match}\ \texttt{inj}_k\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ \texttt{end} \to N_k\{x_k \leftarrow M\}$$

do not suffice to eliminate all cuts. We need to add many unusual reduction rules to handle the so-called "commutative cuts" (Prawitz, 1965). For example, in the case of $\top$:

$$\begin{aligned}
(\texttt{match}\ M\ \texttt{with}\ \texttt{end})\ N &\to \texttt{match}\ M\ \texttt{with}\ \texttt{end} \\
\pi_i\ (\texttt{match}\ M\ \texttt{with}\ \texttt{end}) &\to \texttt{match}\ M\ \texttt{with}\ \texttt{end} \\
\texttt{match}\ (\texttt{match}\ M\ \texttt{with}\ \texttt{end})\ \texttt{with}\ \ldots\ \texttt{end} &\to \texttt{match}\ M\ \texttt{with}\ \texttt{end}
\end{aligned}$$

Reductions and cut elimination

And for disjunction:

$$(\texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ \texttt{end})\ N$$
$$\to \quad \texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1\ N \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ N\ \texttt{end}$$

$$\pi_i\ (\texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ \texttt{end})$$
$$\to \quad \texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow \pi_i\ N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow \pi_i\ N_2\ \texttt{end}$$

$$\begin{array}{l}\texttt{match}\ (\texttt{match}\ M\ \texttt{with}\ \texttt{inj}_1\ x_1 \Rightarrow N_1 \mid \texttt{inj}_2\ x_2 \Rightarrow N_2\ \texttt{end}) \\ \texttt{with}\ \texttt{inj}_1\ y_1 \Rightarrow P_1 \mid \texttt{inj}_2\ y_2 \Rightarrow P_2\ \texttt{end}\end{array}$$
$$\begin{array}{ll}\to & \texttt{match}\ M\ \texttt{with} \\ & \quad \texttt{inj}_1\ x_1 \Rightarrow (\texttt{match}\ N_1\ \texttt{with}\ \texttt{inj}_1\ y_1 \Rightarrow P_1 \mid \texttt{inj}_2\ y_2 \Rightarrow P_2\ \texttt{end}) \\ & \mid\ \texttt{inj}_2\ x_2 \Rightarrow (\texttt{match}\ N_2\ \texttt{with}\ \texttt{inj}_1\ y_1 \Rightarrow P_1 \mid \texttt{inj}_2\ y_2 \Rightarrow P_2\ \texttt{end})\end{array}$$

V

Summary

Curry-Howard in 1970

An isomorphism between simply-typed λ-calculus and intuitionistic logic that equates

- types and propositions;
- terms and proofs;
- reductions and cut elimination.

Is this just a coincidence?

Is the result specific to simply-typed λ-calculus?
Can we extend it to richer programming languages?

Is the result specific to intuitionistic logic?
Can we extend it to other logics? to classical logic?

Some answers in the forthcoming lectures!

VI

Further reading

Further reading

Popularization:

- Gilles Dowek. *Computation, Proof, Machine.* Cambridge University Press, 2015.
- Palmström. *The Lambda Calculus for Absolute Dummies (like myself)*, 2012. `http://palmstroem.blogspot.com/2012/05/lambda-calculus-for-absolute-dummies.html`

Reference material:

- Jean-Yves Girard. *The blind spot: Lectures on logic.* European Mathematical Society, 2011. Mainly chapters 4 and 5.
- Benjamin Pierce. *Types and Programming Languages.* MIT Press, 2002. Mainly chapters 5, 9, 11 and 12.
- Morten Heine Sørensen, Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* 1998. Chapters 1 to 5.
  `https://disi.unitn.it/~bernardi/RSISE11/Papers/curry-howard.pdf`