The Curry-Howard correspondence

In the first talk we discussed the overall aim of the seminar, which is to understand some aspects of the theory of topoi. One of the things which makes this subject interesting is that it has connections to logic (e.g. classifying topoi of first-order theories, which are logical concepts) and also the theory of computation. The purpose of today's talk is to clarify what this word "computation" means and explain why categories naturally appear in the theory of computation. The approach will be a bit idiosyncratic, since in preparing this talk I found that the strongest argument I can give for why you ought to care about computation is to explain how I came to care. So, YMMV.

What is a function?

- Kids learn: a function is a rule $x \mapsto x^2$, a domain $\mathbb{N}$ and range $\mathbb{N}$.
- Grownups know better: a function $\mathbb{N} \to \mathbb{N}$ is a subset $f \subseteq \mathbb{N} \times \mathbb{N}$ s.t.

$$\forall x \in \mathbb{N}\left(\exists y \in \mathbb{N}\left((x,y) \in G \wedge \forall y' \in \mathbb{N}\left((x,y') \in G \Rightarrow y = y' \right) \right) \right)$$

- The kids are right: $f = \{ (x,y) \in \mathbb{N} \times \mathbb{N} \mid y = x^2 \}$. The formula $F : y = x^2$ determines the function f. In a general sense the "computational" outlook on mathematics distinguishes F from f and emphasises the former:
  - F is implicit and finite,
  - f is explicit and infinite.

To see what this has to do with computation, let us consider two functions

$$\mathbb{N} \xrightarrow{f} \mathbb{N} \xrightarrow{g} \mathbb{N}$$

$$f(x) = x^2 \qquad g(y) = y^3$$

$$f = \{(x,y) \in \mathbb{N}^2 \mid y = x^2\} \qquad g = \{(y,z) \in \mathbb{N}^2 \mid z = y^3\}$$

(call this F) (call this G)

The set-theoretic definition of function composition is

$$g \circ f = \{(x,z) \in \mathbb{N}^2 \mid \exists y \in \mathbb{N}\,(\,(x,y) \in f \wedge (y,z) \in g\,)\}$$

$$= \{(x,z) \in \mathbb{N}^2 \mid \exists y \in \mathbb{N}\,(\,y = x^2 \wedge z = y^3\,)\}$$

$$= \{(x,z) \in \mathbb{N}^2 \mid \exists y \in \mathbb{N}\,(\,y = x^2 \wedge z = (x^2)^3\,)\}$$

(note the substitution)

$$= \{(x,z) \in \mathbb{N}^2 \mid z = x^6\}$$

- This example illustrates the general point: in many cases there is a mathematical object (formula, algorithm, program, ...) anterior to functions, and there is a process (computation, rewriting, reduction to normal form, ...) which is anterior to function composition. The theory of computation is the study of these anterior objects and processes.

- We will develop this idea more systematically using λ-calculus and categories.

(λ-calculus: algorithms & reduction; categories: functions & composition)

λ-calculus

(see Lambek, Scott "Introduction to higher order categorical logic" p. 72)

The λ-calculus (due to Church) is one way of formalising the concept of an algorithm. We only have time for a very brief treatment, but see the eavlier semester of this seminar (http://therisingsea.org/post/seminar-ch/) for a more detailed introduction.

The simply-typed λ-calculus is constructed from a set Σ of atomic types. The set of types and preterms is defined by:

types if β, γ are types so are $\beta \to \gamma$, $\beta \times \gamma$. There is a special type 1. (for empty product)

preterms
- for each type β there is an infinite set of variables V_β of type β, write $x : \beta$ for $x \in V_\beta$.
- if $M : \beta \to \gamma$, $N : \beta$ are preterms then $(M\,N) : \gamma$ is a preterm
- if $M : \gamma$ is a preterm and $x : \beta$ is a variable, $(\lambda x . M) : \beta \to \gamma$ is a preterm. (we say x is bound in M)
- if $M : \beta \times \gamma$ is a preterm, $\pi_1 M : \beta, \pi_2 M : \gamma$ are preterms
- if $M : \beta$, $N : \gamma$ are preterms then $\langle M, N \rangle : \beta \times \gamma$ is a preterm.

Note There is a special variable $* : 1$ and all other terms of type 1 are identified with it.

terms • a **term** is an equivalence class of preterms, under **α-equivalence**

$$(\lambda x.M) =_\alpha (\lambda y.M[x:=y]) \qquad \circledast$$

↑ replace occurences of x by y

where every occurrence of y in M is bound (i.e. in the scope of a λy).
We also close $=_\alpha$ under "contexts" i.e. $(N\ (\lambda x.M)) =_\alpha (N\ (\lambda y.M[x:=y]))$
and so on, for arbitrary nestings of $\circledast$. From now on $=$ means $=_\alpha$.

Upshot in $(\lambda x.M)$ the identity of "x" is irrelevant, only the pattern of occurrences.

One-step β-reduction is a relation on terms generated by

$$((\lambda x.M)\ N) \rightsquigarrow_\beta M[x:=N]$$

called β-redexes

$$\pi_1\langle M, N\rangle \rightsquigarrow_\beta M$$

$$\pi_2\langle M, N\rangle \rightsquigarrow_\beta N$$

Def^n A λ-term is **normal** if it contains no β-redexes.

Theorem Starting from any λ-term M, any sequence of choices of β-redexes to reduce terminates in a normal form (**strong normalisation**) and that normal form is independent of the choices (**confluence**).

Upshot A λ-term is an *algorithm* and β-reduction is *computation*.

Def^n $=_\beta$ is the equivalence relation on terms generated by $\rightsquigarrow_\beta$.

To make the connection to the earlier example, suppose we have a special type $\mathbb{N}$ and constants (special terms) $n : \mathbb{N}$ for $n \in \mathbb{N}$, and a multiplication $\mu : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ (which is again a special term). Then

$$\mathcal{F} := (\lambda x^{\mathbb{N}}.\ (\mu \langle x, x \rangle)) : \mathbb{N} \longrightarrow \mathbb{N} \qquad (\text{recall } f(x) = x^2)$$

$$\mathcal{G} := (\lambda y^{\mathbb{N}}.\ (\mu \langle (\mu \langle y, y \rangle), y \rangle)) : \mathbb{N} \longrightarrow \mathbb{N} \qquad (\text{recall } g(y) = y^3)$$

The "plugging" of $\mathcal{G}$ and $\mathcal{F}$ is

$$(\lambda z^{\mathbb{N}}.\ (\mathcal{G}\ (\mathcal{F}\ z))) \rightsquigarrow_\beta (\lambda z.(\mathcal{G}\ (\mu \langle z, z \rangle))) \qquad (\dagger)$$

(this is my y)

notion of composition or plugging of λ-terms

$$\rightsquigarrow_\beta (\lambda z.(\mu \langle (\mu \langle (\mu \langle z, z \rangle), (\mu \langle z, z \rangle) \rangle), (\mu \langle z, z \rangle) \rangle))$$

$$\rightsquigarrow (\lambda z.\ z^6)$$

encoded appropriately, and with some "standard" arrangement of μ's and brackets

associativity of μ is a "special" relation of the kind referred to in the Note below

Note There may be other constants, term forming operations and equations on terms in the system, so there are in fact many λ-calculi. The one without "any extras" is called pure λ-calculus.

If we define

$$\mathcal{C}(\mathbb{N},\mathbb{N}) := \{\text{closed } \lambda\text{-terms of type } \mathbb{N}\to\mathbb{N}\}/=_\beta$$

then plugging of λ-terms induces a composition

$$\begin{aligned} c : \mathcal{C}(\mathbb{N},\mathbb{N})\times\mathcal{C}(\mathbb{N},\mathbb{N}) &\longrightarrow \mathcal{C}(\mathbb{N},\mathbb{N}) \\ ([g]_\beta,[f]_\beta) &\longmapsto [(\lambda z.(g\,(f\,z)))]_\beta \\ &= [(\lambda z.z^6)]_\beta \end{aligned}$$

which more generally defines composition in the category $\mathcal{C}$ of types (=objects) and closed terms (=morphisms) in simply-typed λ-calculus. This category is the principled way of associating functions, e.g. g in the diagram

$$\begin{array}{ccc} \mathcal{C}(*,\mathbb{N}) & \xrightarrow{c([g],-)} & \mathcal{C}(*,\mathbb{N}) \\ \cong\downarrow & & \cong\downarrow \\ \mathbb{N} & \xrightarrow[g]{} & \mathbb{N} \end{array} \qquad g(x)=x^3$$

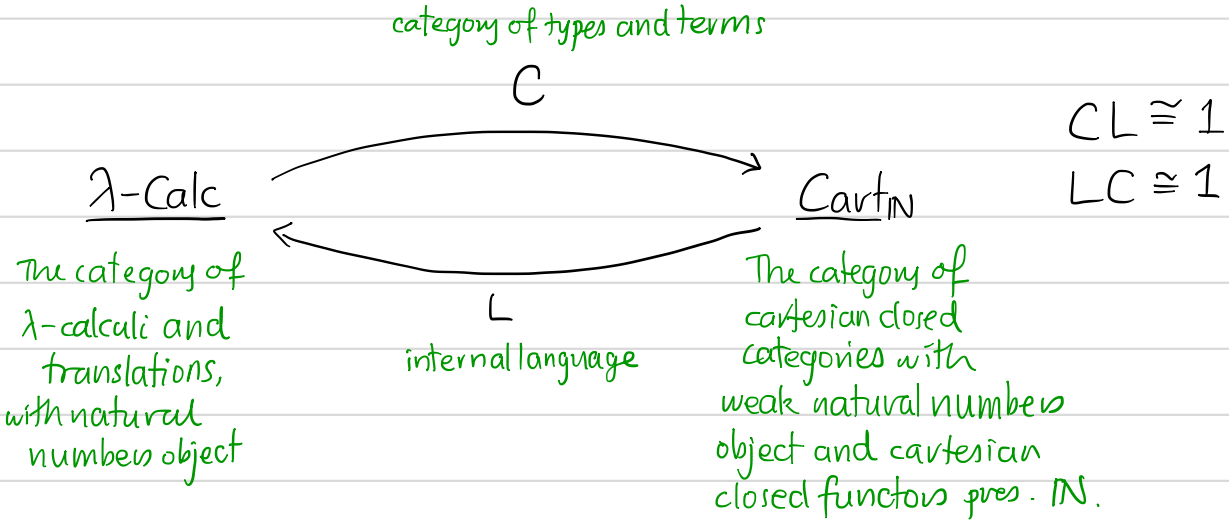
to algorithms, in this case the term g in λ-calculus. This gives an explicit context in which algorithms are anterior to functions, and anterior to function composition is the process of "plugging and then reducing to normal form".

Upshot There is more structure in λ-calculus than in the category $\mathcal{C}$. This difference is what the theory of computation is about!

(ccc)

LS This construction associates a Cartesian closed category $\mathcal{C}(\mathcal{L})$ to any λ-calculus $\mathcal{L}$, and all CCCs arise this way:

Theorem (Lambek-Scott.) There is an equivalence of categories



In light of our previous discussion, one might say: λ-calculus provides a computational enrichment $\mathcal{L}(\mathcal{T})$ for any Cartesian closed category $\mathcal{T}$.

To summarise: the theory of computation is about algorithms and plugging. Category theory is an abstract theory of functions and composition. Since algorithms determine functions, it is natural that part of the structure of computation can be organised using category theory. But this is the "boring" part!

↑ Nonetheless category theory is very useful, as a source of concepts which can be used to organise the more mysterious anterior world of algorithms.

BHK

According to Stanford's philosophy encyclopedia (see the entry on the development of intuitionistic logic) the standard explanation of intuitionistic logic today is the BHK (Brouwer-Heyting-Kolmogorov) interpretation:

- A proof of $A \wedge B$ is given by presenting a proof of A and a proof of B
- A proof of $A \vee B$ is given by presenting either a proof of A or a proof of B.
- A proof of $A \longrightarrow B$ is a construction which permits us to transform any proof of A into a proof of B.

To appreciate the truly radical nature of this last point, it is useful to know that there are, in general, proofs of $A \longrightarrow B$ which give rise to different transformations of proofs of A into proofs of B. The computational perspective on intuitionistic logic de-emphasises the question of whether $A \longrightarrow B$ is provable and emphasises the structure of the collection of proofs (viewed as transformations of proofs into other proofs).

This point of view is made precise by the Curry-Howard correspondence.

CH For a more detailed discussion of the Curry-Howard correspondence with detailed references see last semester's lecture notes at (http://therisingsea.org/post/seminar-ch/). We only have time for a sketch. The correspondence relates intuitionistic logic and λ-calculus. For simplicity we consider only the $\to$ connective on the logical side (so not $\wedge, \vee$) and the λ-calculus side (so not $\times$).

- Intuitionistic logic (more precisely, natural deduction) has propositional variables $p, q, r, \ldots$ and formulas $\mathcal{Y}, \phi, \ldots$ built from these variables using $\to$, e.g. $(p \to q) \to (q \to p)$. A sequent is

$$x_1 : \mathcal{S}_1, \ldots, x_n : \mathcal{S}_n \vdash \mathcal{Y}$$

where the x_i are some labels, used to distinguish multiple copies of a formula. Read the sequent as: from hypotheses $\mathcal{S}_1, \ldots, \mathcal{S}_n$ we may deduce $\mathcal{Y}$. A proof of a sequent is a derivation of it from axioms via deduction rules

$$\Gamma, x : \mathcal{S} \vdash \mathcal{S} \qquad (\text{Axiom}, \Gamma \text{ arbitrary})$$

$$\frac{\Gamma, x : \mathcal{S} \vdash \mathcal{Y}}{\Gamma : \mathcal{S} \to \mathcal{Y}} (\to I) \qquad \frac{\Gamma \vdash \mathcal{S} \to \mathcal{Y} \quad \Gamma \vdash \mathcal{S}}{\Gamma \vdash \mathcal{Y}} (\to E)$$

← modus ponens

We identify proofs that are "α-equivalent".

Theorem (Curry '58, Howard '69) For any formula/type $\mathcal{S}$ there is a bijection between proofs of $\vdash \mathcal{S}$ and closed λ-terms of type $\mathcal{S}$.

Technically this is very shallow, but it is nonetheless important, as it gives a foundation to BHK (proofs as programs/algorithms/transformations) and thereby has the potential to move computation from the periphery of a mathematician's worldview ("I don't care about programs") to somewhere near the center ("Oh, but every intuitionistic proof is an example of a program..."). This happened to me.

Conclusion