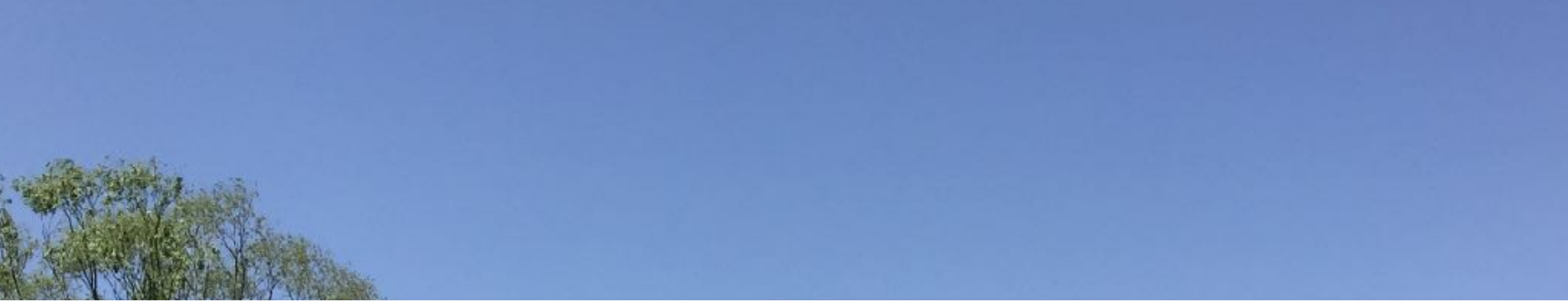


The category of open simply-typed lambda terms

Daniel Murfet

based on joint work with William Troiani

Reminder on category theory

- Every adjunction gives rise to a monad

$$\mathcal{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathcal{D} \qquad \mathcal{C}(Fx, y) \cong \mathcal{D}(x, Gy)$$

$$\mathcal{C}(FGx, x) \cong \mathcal{D}(Gx, Gx)$$

$$M = GF, \quad M = GFGF \longrightarrow GF$$

- Monads may be used to extend functional programming languages in a principled way to include “effects” (Moggi, Wadler). This is a foundational idea in Haskell.

Reminder on category theory

Let $\mathcal{C}$ be a Cartesian closed category

$$\mathcal{C} \underset{G=s\Rightarrow(-)}{\overset{F=s\times(-)}{\rightleftarrows}} \mathcal{C}$$

$$\mathcal{C}(Fx, y) = \mathcal{C}(s \times x, y) \cong \mathcal{C}(s, x \Rightarrow y) = \mathcal{C}(s, Gy)$$

- **Aim**: to define the category of lambda terms (the basic example of a category of programs) and study monads on this category.

- We take a nonstandard point of view, which is

  - thoroughly monadic (following Lawvere),
  - treats open and closed terms on an equal footing.

- **Applications**: leads to a Curry-Howard correspondence between open lambda terms and proofs with hypotheses, and shows that the structural rules of logic (weakening, contraction, exchange) are monadic.

Curry-Howard correspondence

logic	**programming**	**category theory**
formula	type	object
proof	program	morphism
cut	function application	composition
implication intro	lambda abstraction	universal factorisation
structural rules	**-**	**monad product, unit**
open context	**free variables**	**Kleisli categories**

Simply-typed Lambda calculus

- Types: $\sigma ::= a \mid \sigma_1 \Rightarrow \sigma_2, \quad e.g. \quad \sigma \Rightarrow (\tau \Rightarrow \sigma)$
- Preterms: $M ::= x^\sigma \mid (M^{\sigma \Rightarrow \rho}\ N^\sigma) : \rho \mid (\lambda x^\sigma\ .\, N^\rho) : \sigma \Rightarrow \rho$
- α-equivalence: $\lambda x^\sigma\ .\, M =_\alpha \lambda y^\sigma\ .\, M[x := y]$
- β-equivalence: $((\lambda x^\sigma\ .\, M)N) =_\beta M[x := N]$
- η-equivalence: $\lambda x^\sigma\ .\, (Mx) =_\eta M$
- Free variables: $\mathrm{FV}(\lambda x^{\sigma \Rightarrow \rho}\ .\, (xy^\sigma)) = \{y : \sigma\}$

Simply-typed Lambda calculus

- Terms are preterms up to alpha equivalence
- For a type σ we refer to the set of terms of this type by Λ_σ and the set of all terms of this type up to beta (or beta-eta)-equivalence by

$$\Lambda_\sigma / =_\beta \qquad \Lambda_\sigma / =_{\beta\eta}$$

- Terms up to beta-equivalence do not have a well-defined set of free variables, so we define

$$\mathrm{FV}_\beta(M) = \bigcap_{N =_\beta M} \mathrm{FV}(N)$$

Some examples

$$(a) \quad F : \tau \Rightarrow \rho \, , G : \sigma \Rightarrow \tau \qquad \lambda x^{\sigma} \, . \, (F(Gx)) : \sigma \Rightarrow \rho$$

$$(b) \quad \mathrm{id}_{\sigma} := \lambda x^{\sigma} \, . \, x : \sigma \Rightarrow \sigma$$

$$\begin{aligned} (c) \quad \lambda y^{\sigma} \, . \, (F(\mathrm{id}_{\sigma} \, y)) &= \lambda y \, . \, (F((\lambda x \, . \, x) y)) \\ &=_{\beta} \lambda y \, . \, (F y) \\ &=_{\eta} F \end{aligned}$$

Desiderata for a category of lambda terms

- Every type should be an object.
- Every term (open or closed) should be a morphism.
- The basic operations on lambda terms should have natural categorical avatars:
  - *function application*
  - *lambda abstraction*
  - *structural rules (discarding, copying, re-ordering inputs)*
- The usual approach (Lambek-Scott) assumes product types.

Definition : The category $\mathcal{L}$ has objects (let $\Phi_\Rightarrow = \{\text{simple types}\}$)

$$\mathrm{ob}(\mathcal{L}) = \Phi_\Rightarrow \cup \{\mathbf{1}\}$$

and morphisms given for types $\sigma, \tau \in \Phi_\Rightarrow$ by

$$\begin{aligned}
\mathcal{L}(\sigma, \tau) &= \Lambda_{\sigma \Rightarrow \tau} / =_{\beta\eta} \\
\mathcal{L}(\mathbf{1}, \sigma) &= \Lambda_\sigma / =_{\beta\eta} \\
\mathcal{L}(\sigma, \mathbf{1}) &= \{\star\} \\
\mathcal{L}(\mathbf{1}, \mathbf{1}) &= \{\star\}\,,
\end{aligned}$$

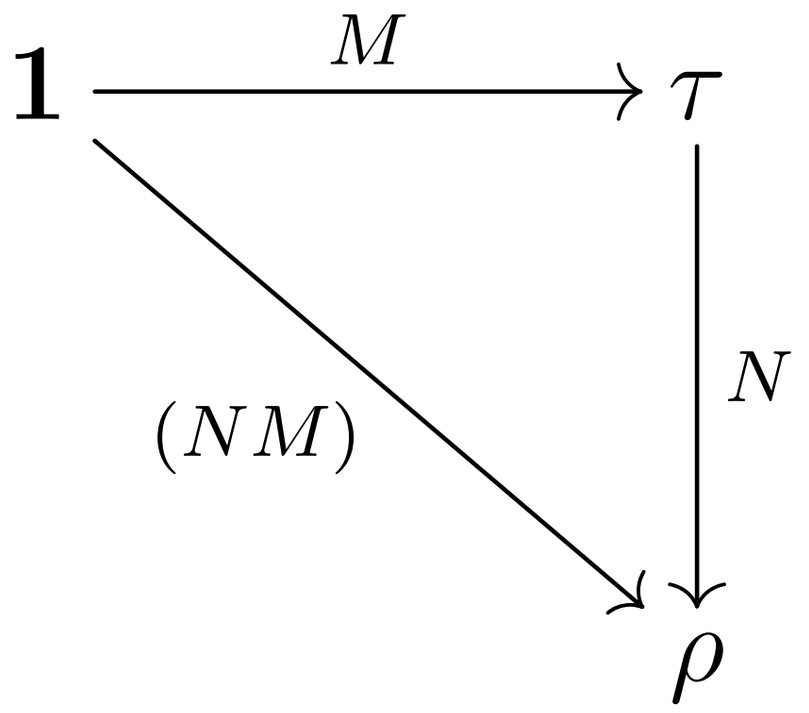
where $\star$ is a new symbol. For $\sigma, \tau, \rho \in \Phi_\Rightarrow$ the composition rule is the function

$$\begin{aligned}
\mathcal{L}(\tau, \rho) \times \mathcal{L}(\sigma, \tau) &\longrightarrow \mathcal{L}(\sigma, \rho) \\
(N, M) &\longmapsto \lambda x^\sigma \,.\, (N(Mx))\,,
\end{aligned}$$

where $x \notin \mathrm{FV}(N) \cup \mathrm{FV}(M)$. We write the composite as $N \circ M$. In the remaining special cases the composite is given by the rules

$$\begin{aligned}
\mathcal{L}(\tau, \rho) \times \mathcal{L}(\mathbf{1}, \tau) &\longrightarrow \mathcal{L}(\mathbf{1}, \rho)\,, \qquad & N \circ M &= (NM)\,, \\
\mathcal{L}(\mathbf{1}, \rho) \times \mathcal{L}(\mathbf{1}, \mathbf{1}) &\longrightarrow \mathcal{L}(\mathbf{1}, \rho)\,, \qquad & N \circ \star &= N\,, \\
\mathcal{L}(\mathbf{1}, \rho) \times \mathcal{L}(\sigma, \mathbf{1}) &\longrightarrow \mathcal{L}(\sigma, \rho)\,, \qquad & N \circ \star &= \lambda t^\sigma \,.\, N\,,
\end{aligned}$$

Function application is composition



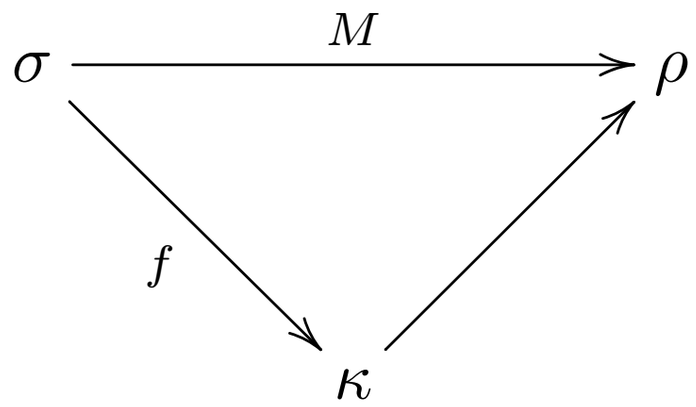
$M : \tau$ $\qquad$ $N : \tau \Rightarrow \rho$

Lambda abstraction is factorisation

Let Y be the set of all variables, then $Q \subseteq Y$ is *cofinite* if $Y \setminus Q$ is finite.

Definition. Let Q be a cofinite set of variables. A *Q-factorisation* of a term $M : \sigma \Rightarrow \rho$ is a commutative diagram in $\mathcal{L}$ with $\mathrm{FV}_\beta(f) \subseteq Q$:

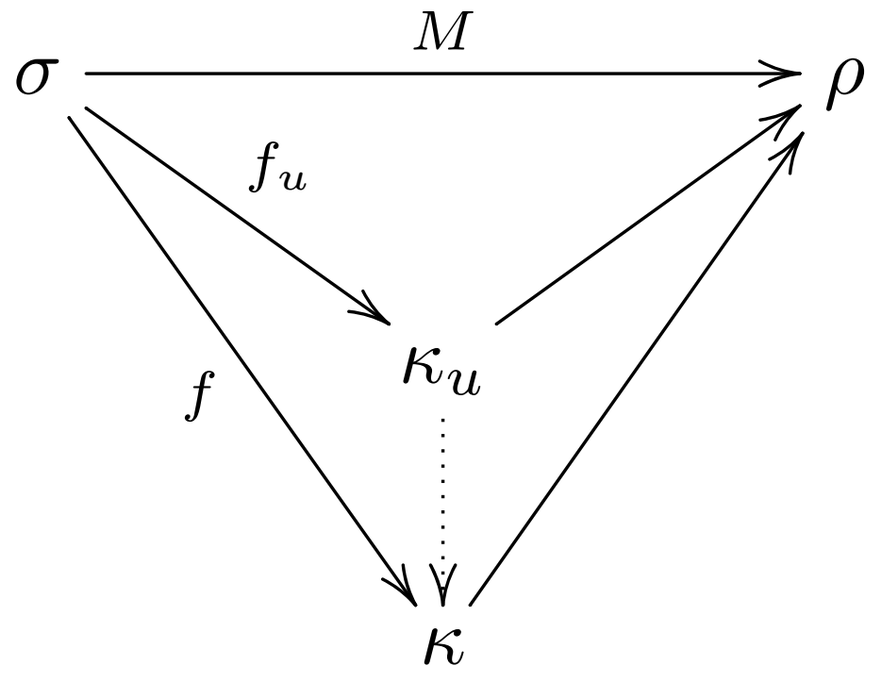
$$Q = Y \setminus \{q : \tau\}$$
$$q \notin \mathrm{FV}_\beta(f)$$



$$\mathrm{FV}_\beta(M) = \bigcap_{N =_\beta M} \mathrm{FV}(N)$$

Lambda abstraction is factorisation

Theorem. There is a universal Q-factorisation.



If $Q = Y \setminus \{q : \tau\}$ then

$$\kappa_u = \tau \Rightarrow \rho$$

$$f_u = \lambda q \,.\, M$$

Definition 4.7. For a subset $Q \subseteq Y$ we define a subcategory $\mathcal{L}_Q \subseteq \mathcal{L}$ by

$$\mathrm{ob}(\mathcal{L}_Q) = \mathrm{ob}(\mathcal{L}) = \Phi_{\Rightarrow} \cup \{\mathbf{1}\}$$

and for types σ, ρ

$$\begin{aligned}
\mathcal{L}_Q(\sigma, \rho) &= \{M \in \mathcal{L}(\sigma, \rho) \mid \mathrm{FV}_\beta(M) \subseteq Q\}, \\
\mathcal{L}_Q(\mathbf{1}, \sigma) &= \{M \in \mathcal{L}(\mathbf{1}, \sigma) \mid \mathrm{FV}_\beta(M) \subseteq Q\}, \\
\mathcal{L}_Q(\sigma, \mathbf{1}) &= \mathcal{L}(\sigma, \mathbf{1}) = \{\star\}, \\
\mathcal{L}_Q(\mathbf{1}, \mathbf{1}) &= \mathcal{L}(\mathbf{1}, \mathbf{1}) = \{\star\}.
\end{aligned}$$

Note that the last two lines have the same form as the first two, using the convention that $\mathrm{FV}_\beta(\star) = \emptyset$. We denote the inclusion functor by $I_Q : \mathcal{L}_Q \longrightarrow \mathcal{L}$. We write $\mathcal{L}_{cl}$ for $\mathcal{L}_Q$ when $Q = \emptyset$ and call this the category of *closed* lambda terms.

Theorem. If Q is cofinite, the functor $I_Q : \mathcal{L}_Q \to \mathcal{L}$ has a right adjoint.

Theorem. If A is finite, the functor $I_A : \mathcal{L}_{cl} \to \mathcal{L}_A$ has a right adjoint.

$$\mathcal{L}_{cl} \underset{\Gamma_{\mathfrak{a}}}{\overset{I}{\rightleftarrows}} \mathcal{L}_A \qquad T_{\mathfrak{a}} = \Gamma_{\mathfrak{a}} \circ I \text{ is a monad}$$

($\mathfrak{a}$ an ordering of A)

Relation to the Lambek-Scott category

In the standard approach to associating a category to the simply-typed lambda calculus, due to Lambek and Scott [5, §I.11], one extends the lambda calculus to include product types, denoted $\sigma \times \tau$, and the objects of the category $\mathcal{C}_{\Rightarrow,\times}$ are the types of the extended calculus (which includes an empty product $\mathbf{1}$) and the set $\mathcal{C}_{\Rightarrow,\times}(\sigma,\rho)$ is a set of equivalence classes of pairs $(x:\sigma, M:\rho)$ where x is a variable and M is a term with $\mathrm{FV}(M) \subseteq \{x\}$.

Despite appearances, $\mathcal{C}_{\Rightarrow,\times}$ should really be thought of as the category of *closed* lambda terms, as the following theorem makes clear. Let $\mathcal{C}_{\Rightarrow}$ denote the full subcategory of $\mathscr{C}_{\Rightarrow,\times}$ whose objects are elements of the set $\Phi_{\Rightarrow}$.

Theorem 5.1. *There is an equivalence of categories*

$$F : \mathcal{C}_{\Rightarrow} \to \mathcal{L}_{cl}^{\neq \mathbf{1}}$$

which is the identity $F(\tau) = \tau$ on objects, and is defined on morphisms by

$$F_{\sigma,\rho} : \mathcal{C}_{\Rightarrow}(\sigma,\rho) \longrightarrow \mathcal{L}_{cl}^{\neq \mathbf{1}}(\sigma,\rho)\,, \qquad (x:\sigma, M:\rho) \mapsto \lambda x\,.\,M\,.$$

Relation to the Lambek-Scott category

$$\mathcal{C}_{\Rightarrow}(\sigma \times \rho, \tau) \cong \mathcal{C}_{\Rightarrow}(\sigma, \rho \Rightarrow \tau)$$

$$(z : \sigma \times \rho, M : \tau) \mapsto \lambda x^{\sigma} y^{\rho} \,.\, ((\lambda z \,.\, M)\langle x, y \rangle)$$

Structural rules are monadic

The *structural rules* of the sequent calculus of intuitionistic logic are

$$\text{(Contraction):}\quad \frac{\mathfrak{b}, q:\tau, q':\tau \vdash M(q,q'):\sigma}{\mathfrak{b}, q:\tau \vdash M(q,q):\sigma}\ \text{ctr}$$

$$\text{(Weakening):}\quad \frac{\mathfrak{b} \vdash M:\sigma}{\mathfrak{b}, q:\tau \vdash M:\sigma}\ \text{weak}$$

$$\text{(Exchange):}\quad \frac{\mathfrak{b}, q_1:\tau_1, q_2:\tau_2 \vdash M:\sigma}{\mathfrak{b}, q_2:\tau_2, q_1:\tau_1 \vdash M:\sigma}\ \text{ex}$$

where $\mathfrak{b}$ denotes a typing context, which is an ordered list of typed variables. From the point of view of lambda calculus these rules correspond respectively to the identification of two free variables (contraction) the introduction of a spurious dependence on a free variable (weakening) and the exchange of the order of two free variables in the context which is viewed as an ordered list (exchange). These structural rules can be recognised in the categorical presentation of lambda calculus given in this paper, using the structure presented on the category $\mathcal{L}_{cl}$ by the monads $T_{\mathfrak{a}}$ discussed above.

Let us first explain the interpretation $[\![-]\!]$ of typing judgements for open lambda terms in $\mathcal{L}$. If there is a typing judgement for $M : \sigma$ of the form

$$\mathfrak{a} \vdash M : \sigma$$

then the denotation is just the term M, but with the context $\mathfrak{a}$ recorded either by the monad $T_{\mathfrak{a}}$ or by working in the category $\mathcal{L}_A$ where A is the underlying set of $\mathfrak{a}$:

$$[\![\mathfrak{a} \vdash M : \sigma]\!] = M \in \mathcal{L}_{cl}(\mathbf{1}, T_{\mathfrak{a}}\sigma) = \mathcal{L}_A(\mathbf{1}, \sigma)\,.$$

Now, observe that

$$[\![\mathfrak{b}, q : \tau, q' : \tau \vdash M : \sigma]\!] = M \in \mathcal{L}_{cl}(\mathbf{1}, T_{\mathfrak{b},q,q'}\sigma)\,.$$

But there is an obvious isomorphism of monads $T_{\{q:\tau\}} \cong T_{\{q':\tau\}}$ on $\mathcal{L}_{cl}$, so that we have, using the multiplication on the monad $T_{\{q:\tau\}}$, the map

$$\begin{aligned}
\mathcal{L}_{cl}(\mathbf{1}, T_{\mathfrak{b},q,q'}\sigma) &\cong \mathcal{L}_{cl}(\mathbf{1}, T_{\mathfrak{b}}T_{\{q:\tau\}}T_{\{q':\tau\}}\sigma) \\
&\cong \mathcal{L}_{cl}(\mathbf{1}, T_{\mathfrak{b}}T_{\{q:\tau\}}T_{\{q:\tau\}}\sigma) \\
&\longrightarrow \mathcal{L}_{cl}(\mathbf{1}, T_{\mathfrak{b}}T_{\{q:\tau\}}\sigma) \\
&\cong \mathcal{L}_{cl}(\mathbf{1}, T_{\mathfrak{b},q:\tau}\sigma)\,.
\end{aligned}$$

The image under this map of $[\![\mathfrak{b}, q : \tau, q' : \tau \vdash M : \sigma]\!]$ is precisely the result of applying the contraction rule to the typing judgement.