A Categorical Perspective on Type Refinement Systems

Noam Zeilberger[1]

University of Birmingham

Cambridge Logic Seminar
9 December 2016

[1] Joint work with Paul-André Melliès.

What is a type refinement system?

Intuition: a **type refinement system** is a type system built over a typed programming language, as an extra layer of typing.

Examples (90s–10s): DML, SML Cidre, Stardust, Liquid Haskell, Typed Racket, TypeScript, Flow, ...

What is a type system?

As with many terms shared by large communities, it is difficult to define "type system" in a way that covers its informal usage by programming language designers and implementors but is still specific enough to have any bite.
– Benjamin Pierce (2002), TaPL

What is a type?

One reason it is hard to give a formal definition is because there are two competing *philosophies* of types...

"à la Church" *vs.* "à la Curry"

a.k.a.

intrinsic *vs.* extrinsic

Intuition from logic: types-as-sorts *vs.* types-as-predicates

The extrinsic view: an excerpt

We now proceed, in outline, as follows. We define a new class of expressions which we shall call types; then we say what is meant by a value **possessing** *a type. Some values have many types, and some have no type at all. In fact "wrong" has no type. But if a functional value has a type, then as long as it is applied to the right kind (type) of argument it will produce the right kind (type) of result- which cannot be "wrong"!*
– Robin Milner (1978), "A Theory of Type Polymorphism in Programming"

Problem: the "naive" reading of type theory through the lens of category theory is biased towards the *intrinsic* view of typing.

The naive reading

type system $\Rightarrow$ category of well-typed terms

$$x : A \vdash t : B \quad \Rightarrow \quad [\![A]\!] \xrightarrow{[\![x.t]\!]} [\![B]\!]$$

The problem with the naive reading

But every morphism $f : A \to B$ of a category is intrinsically associated with a unique pair of types! (Namely, A and B.)

This makes it difficult to interpret extrinsic typing rules such as the **subsumption rule** or **intersection introduction**:

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

More fundamentally, the problem is that the naive reading does not distinguish terms from typing derivations.

A more subtle reading

Define the semantics of a typed language by induction on typing derivations, then prove a **coherence theorem**:

$$\text{if } \Gamma \overset{\alpha}{\vdash} t : A \text{ and } \Gamma \overset{\beta}{\vdash} t : A \text{ then } [\![\alpha]\!] = [\![\beta]\!] : [\![\Gamma]\!] \to [\![A]\!]$$

In general, coherence is a nontrivial theorem...

Nicely discussed by Reynolds (1991–2000):

- The Coherence of Languages with Intersection Types
- *Theories of Programming Languages* (Chs. 15 & 16)
- The Meaning of Types: from Intrinsic to Extrinsic Semantics

Our goal: stay naive (rather than subtle), just not *too* naive!

Functors are type refinement systems

Remembering to forget

Intuitively, most type systems come with an "erasure" operation...

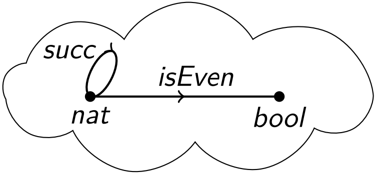
$$\text{Derivations} \\ \Big\downarrow \\ \text{Terms}$$

Well, what if we take an *arbitrary functor* $U : \mathscr{D} \to \mathscr{T}$ and try to view it as a type system? We'll think of the morphisms of $\mathscr{D}$ as derivations, and the morphisms of $\mathscr{T}$ as terms.

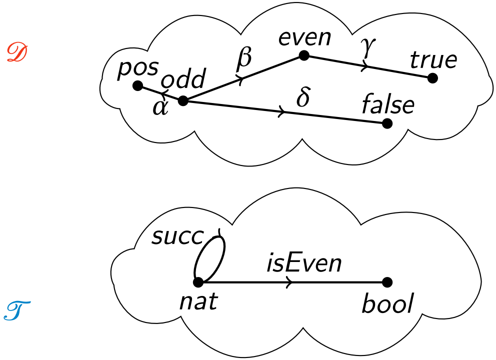
That would make a ***type refinement system,*** though, wouldn't it? Because both $\mathscr{D}$ and $\mathscr{T}$ have types (= objects), and in some sense those of $\mathscr{D}$ "refine" those of $\mathscr{T}$...

An example

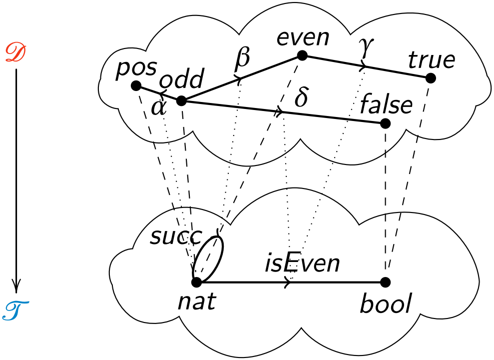
$\mathcal{T}$



An example


$\mathscr{D}$
pos
odd
even
true
false
α
β
γ
δ
succ
isEven
nat
bool
$\mathscr{T}$

An example



$$odd, even, pos \mapsto nat$$

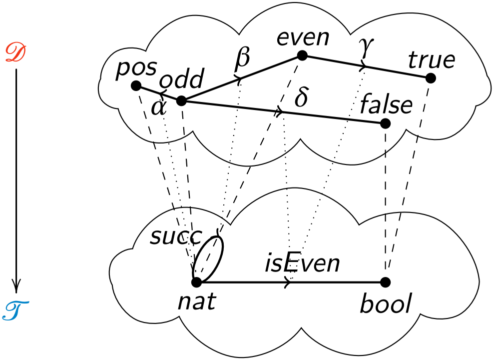
$$true, false \mapsto bool$$

$$\alpha \mapsto \mathrm{id}_{nat}$$

$$\beta \mapsto succ$$

$$\gamma, \delta \mapsto isEven$$

An example

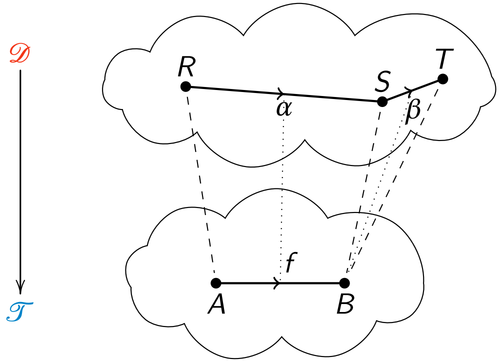


$$odd, even, pos \mapsto nat$$
$$true, false \mapsto bool$$
$$\alpha \mapsto \mathrm{id}_{nat}$$
$$\beta \mapsto succ$$
$$\gamma, \delta \mapsto isEven$$

$$odd \overset{\alpha}{\leq_{nat}} pos \qquad odd \overset{\beta}{\underset{succ}{\Longrightarrow}} even \qquad (odd, even, pos \sqsubset nat)$$

$$even \overset{\gamma}{\underset{isEven}{\Longrightarrow}} true \qquad odd \overset{\delta}{\underset{isEven}{\Longrightarrow}} false \qquad (true, false \sqsubset bool)$$

Functors are type refinement systems



$R \mapsto A$

$S, T \mapsto B$

$\alpha \mapsto f$

$\beta \mapsto \mathrm{id}_B$

$$R \overset{\alpha}{\underset{f}{\Longrightarrow}} S \qquad \overset{\beta}{S \leq_B T} \qquad (R \sqsubset A) \quad (S, T \sqsubset B)$$

NB: the functor $U : \mathcal{D} \to \mathcal{T}$ need *not* be faithful!

The interpretation of typing rules

We call a rule *admissible* relative to $U:\mathscr{D}\to\mathscr{T}$ if given derivations of the premises, we can construct a derivation of the conclusion.

Warmup. Show that the following rules are admissible for any U:

$$\frac{R \underset{f}{\Longrightarrow} S \quad S \underset{g}{\Longrightarrow} T}{R \underset{f;g}{\Longrightarrow} T}$$

$$\frac{R \underset{f}{\Longrightarrow} S \quad S \le T}{R \underset{f}{\Longrightarrow} T} \qquad \frac{R \le S \quad S \underset{g}{\Longrightarrow} T}{R \underset{g}{\Longrightarrow} T}$$

A basic idea worth exploring...

P.-A. Melliès and I have coauthored several papers around this:

- Type refinement and monoidal closed bifibrations arXiv:1310.0263
- Functors are type refinement systems **POPL2015**
- An Isbell duality theorem for type refinement systems **MSCS** (to appear)
- A bifib. reconst. of Lawvere's presheaf hyperdoctrine **LICS2016**

I also wrote some expository notes for OPLSS 2016 (see webpage)

Outline

Our goals for today:

1. Functors are type refinement systems ✓
2. Reading Groth. in translation. (Also maybe: $\wedge$ and $\vee$.)
3. Monoidal closed refinement systems.
4. Using monoidal closed bifibrations as a logical framework.

Reading Grothendieck in translation

Pushforward refinements[2]

A **pushforward of** R **along** f is a refinement

$$\frac{R \sqsubset A \quad f : A \to B}{\mathrm{push}_f\, R \sqsubset B}$$

equipped with a pair of typing rules

$$\frac{}{R \underset{f}{\Longrightarrow} \mathrm{push}_f\, R}\ f_\diamond I \qquad \frac{R \underset{f;g}{\Longrightarrow} S}{\mathrm{push}_f\, R \underset{g}{\Longrightarrow} S}\ f_\diamond E$$

satisfying a pair of equations on typing derivations...

[2]... with respect to a given refinement system $U : \mathscr{D} \to \mathscr{T}$.

Pushforward refinements

...satisfying a pair of equations on typing derivations

$$\dfrac{\dfrac{}{R \underset{f}{\Longrightarrow} \mathsf{push}_f\, R}\ f_\diamond I \qquad \dfrac{\overset{\beta}{R \underset{f;g}{\Longrightarrow} S}}{\mathsf{push}_f\, R \underset{g}{\Longrightarrow} S}\ f_\diamond E}{R \underset{f;g}{\Longrightarrow} S} \quad = \quad \overset{\beta}{R \underset{f;g}{\Longrightarrow} S}$$

$$\overset{\eta}{\mathsf{push}_f\, R \underset{g}{\Longrightarrow} S} \quad = \quad \dfrac{\dfrac{\dfrac{}{R \underset{f}{\Longrightarrow} \mathsf{push}_f\, R}\ f_\diamond I \qquad \overset{\eta}{\mathsf{push}_f\, R \underset{g}{\Longrightarrow} S}}{R \underset{f;g}{\Longrightarrow} S}}{\mathsf{push}_f\, R \underset{g}{\Longrightarrow} S}\ f_\diamond E$$

Pullback refinements

A **pullback of** S **along** f is a refinement

$$\frac{f : A \to B \quad S \sqsubset B}{\mathrm{pull}_f\, S \sqsubset A}$$

equipped with a pair of typing rules

$$\frac{}{\mathrm{pull}_f\, S \underset{f}{\Longrightarrow} S}\; f^{\square}E \qquad \frac{R \underset{g;f}{\Longrightarrow} S}{R \underset{g}{\Longrightarrow} \mathrm{pull}_f\, S}\; f^{\square}I$$

satisfying the pair of equations on typing derivations...

Pullback refinements

...satisfying the pair of equations on typing derivations

$$\dfrac{\dfrac{R \overset{\beta}{\underset{g;f}{\Longrightarrow}} S}{R \underset{g}{\Longrightarrow} \mathsf{pull}_f\, S}\; f^{\square}I \qquad \dfrac{}{\mathsf{pull}_f\, S \underset{f}{\Longrightarrow} S}\; f^{\square}E}{R \underset{g;f}{\Longrightarrow} S}\; ; \qquad = \quad R \overset{\beta}{\underset{g;f}{\Longrightarrow}} S$$

$$R \overset{\eta}{\underset{g}{\Longrightarrow}} \mathsf{pull}_f\, S \quad = \quad \dfrac{\dfrac{R \overset{\eta}{\underset{g}{\Longrightarrow}} \mathsf{pull}_f\, S \qquad \dfrac{}{\mathsf{pull}_f\, S \underset{f}{\Longrightarrow} S}\; f^{\square}E}{R \underset{g;f}{\Longrightarrow} S}\; ;}{R \underset{g}{\Longrightarrow} \mathsf{pull}_f\, S}\; f^{\square}I$$

Grothendieck remixed

Proposition/Definition: A refinement system $U : \mathscr{D} \to \mathscr{T}$ is a **fibration** iff it has all pullbacks. It is an **opfibration** iff it has all pushforwards. It is a **bifibration** iff it has both.

Grothendieck remixed

In a refinement system $\mathscr{D} \to \mathscr{T}$ with (chosen) pushforwards, each morphism $f : A \to B$ induces a *functor* $\mathsf{push}_f : \mathscr{D}_A \to \mathscr{D}_B$,

$$\frac{R \sqsubset B \quad f : A \to B}{\mathsf{push}_f\, R \sqsubset A} \qquad \frac{R_1 \leq_A R_2}{\mathsf{push}_f\, R_1 \leq_B \mathsf{push}_f\, R_2}$$

where each $\mathscr{D}_A$ is the subcategory of $\mathscr{D}$ consisting of refinements $R \sqsubset A$ and subtyping derivations $R_1 \leq_A R_2$ as morphisms.

Grothendieck remixed

We can *derive* the subtyping rule explicitly from the typing rules:

$$\dfrac{\dfrac{R_1 \underset{\mathrm{id}_A}{\Longrightarrow} R_2 \qquad \dfrac{}{R_2 \underset{f}{\Longrightarrow} \mathsf{push}_f\, R_2}\ f_\diamond I}{\dfrac{R_1 \underset{\mathrm{id}_A; f}{\Longrightarrow} \mathsf{push}_f\, R_2}{R_1 \underset{f;\mathrm{id}_B}{\Longrightarrow} \mathsf{push}_f\, R_2}\ \sim}}{\mathsf{push}_f\, R_1 \underset{\mathrm{id}_B}{\Longrightarrow} \mathsf{push}_f\, R_2}\ f_\diamond E$$

Grothendieck remixed

Moreover, we can show that

$$\text{push}_{(g\circ f)}\, R \equiv \text{push}_g\, \text{push}_f\, R \qquad \text{push}_{\mathrm{id}}\, R \equiv R$$

where $\equiv$ denotes "vertical" isomorphism, i.e., pairs of subtyping derivations which compose to the identity.

All this is just another way to say that a (cloven) opfibration $\mathscr{D} \to \mathscr{T}$ induces a (pseudo)functor $\mathscr{T} \to \mathbf{Cat}$.

Grothendieck remixed

A RS that is a *bibration* admits invertible inferences

$$\dfrac{\dfrac{\mathrm{push}_f\, R \leq_B S}{R \underset{f}{\Longrightarrow} S}}{R \leq_A \mathrm{pull}_f\, S}$$

meaning that every $f : A \to B$ gives rise to an adjunction:

$$\mathscr{D}_A \underset{\mathrm{pull}_f}{\overset{\mathrm{push}_f}{\underset{\bot}{\rightleftarrows}}} \mathscr{D}_B$$

Example: SubSet → Set

Formally, the objects of **SubSet** are pairs $(A, R \subseteq A)$, its morphisms $(A, R) \to (B, S)$ are functions $f : A \to B$ such that

$$\forall a.\, a \in R \Rightarrow f(a) \in S$$

and $U : \mathbf{SubSet} \to \mathbf{Set}$ is the projection $(A, R) \mapsto A$.

Pushforward and pullback given by *image* and *inverse image:*

$$\mathrm{push}_f\,(A, R) = (B, \{\, f(a) \mid a \in R \,\})$$
$$\mathrm{pull}_f\,(B, S) = (A, \{\, a \mid f(a) \in S \,\})$$

Other examples of bifibrations

Other typical "semanticky" refinement systems:

- **Downset** → **Poset**: types = posets, terms = monotone functions, refinements = downwards closed subsets
- **Psh** → **Cat**: types = categories, terms = functors, refinements = presheaves, derivations = *natural transformations*
- **Rel.** → **Rel**: like **SubSet** → **Set**, but with terms = relations instead of functions
- **Dist.** → **Dist**: like **Psh** → **Cat**, but with terms = distributors instead of functors

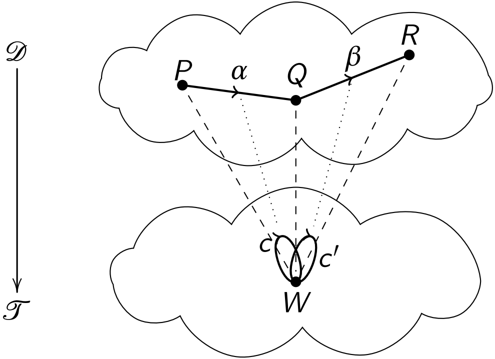
All of these are bifibrations.

As we will discuss later, these are also examples of (cartesian or symmetric) *monoidal closed refinement systems*.

Example: Hoare logic

Take $\mathscr{T}$ as a one-object category of *commands*.

Take $\mathscr{D}$ as a category of *predicates* and *valid Hoare triples*.



Now push = strongest post, pull = weakest pre... but existence depends on particular class of commands and predicates!

Union and intersection refinements

A **union/intersection** of R_1 and R_2 is a refinement...

$$\frac{R_1 \sqsubset A \quad R_2 \sqsubset A}{R_1 \vee R_2 \sqsubset A} \qquad \frac{R_1 \sqsubset A \quad R_2 \sqsubset A}{R_1 \wedge R_2 \sqsubset A}$$

$$\frac{R_1 \underset{f}{\Longrightarrow} S \quad R_2 \underset{f}{\Longrightarrow} S}{R_1 \vee R_2 \underset{f}{\Longrightarrow} S}\ \vee E \qquad \frac{}{R_i \underset{\mathrm{id}_A}{\Longrightarrow} R_1 \vee R_2}\ \vee I_i$$

$$\frac{}{R_1 \wedge R_2 \underset{\mathrm{id}_A}{\Longrightarrow} R_i}\ \wedge E_i \qquad \frac{S \underset{f}{\Longrightarrow} R_1 \quad S \underset{f}{\Longrightarrow} R_2}{S \underset{f}{\Longrightarrow} R_1 \wedge R_2}\ \wedge I$$

...satisfying "β" and "η" equations (analogous to push/pull).

Distributivity principles

We can prove these equivalences in general:

$$\text{push}_f(R \vee S) \equiv \text{push}_f\, R \vee \text{push}_f\, S \tag{1}$$

$$\text{pull}_g(R \wedge S) \equiv \text{pull}_g\, R \wedge \text{pull}_g\, S \tag{2}$$

But the following hold only going forwards (in general):

$$\text{push}_f(R \wedge S) \leq \text{push}_f\, R \wedge \text{push}_f\, S \tag{3}$$

$$\text{pull}_g\, R \vee \text{pull}_g\, S \leq \text{pull}_g(R \vee S) \tag{4}$$

(Exercise: find counterexamples going backwards!)

Monoidal closed refinement systems

Monoidal closed refinement systems

The presence of push/pull/$\vee$/$\wedge$ is a *property* of a refinement system, which can be expressed for any functor $U : \mathcal{D} \to \mathcal{T}$.

On the other hand, we might ask that $\mathcal{D}$ and $\mathcal{T}$ come with some extra structure, and that U preserves that structure.

A **monoidal closed refinement system** is defined as a strict monoidal closed functor between monoidal closed categories.

(SMC and CC refinement systems are defined analogously.)

Examples: **SubSet** → **Set** and **Rel**$_\bullet$ → **Rel**

SubSet → **Set** is a cartesian closed refinement system:

$$(A, R) \times (B, S) = (A \times B, \{(a, b) \mid a \in R \wedge b \in S\})$$
$$(A, R) \to (B, S) = (B^A, \{f \mid \forall a. a \in R \Rightarrow f(a) \in S\})$$

Rel$_\bullet$ → **Rel** is a symmetric monoidal closed refinement system:

$$(A, R) \otimes (B, S) = (A \times B, \{(a, b) \mid a \in R \wedge b \in S\})$$
$$(A, R) \multimap (B, S) = (A \times B, \{(a, b) \mid a \in R \Rightarrow b \in S\})$$

Refinement *vs.* typing *vs.* subtyping

A mc refinement system admits the following refinement rules

$$\frac{R \sqsubset A \quad S \sqsubset B}{R \otimes S \sqsubset A \otimes B} \qquad \frac{R \sqsubset A \quad S \sqsubset B}{R \multimap S \sqsubset A \multimap B}$$

and typing rules

$$\frac{R_1 \underset{f}{\Longrightarrow} R_2 \quad S_1 \underset{g}{\Longrightarrow} S_2}{R_1 \otimes S_1 \underset{f \otimes g}{\Longrightarrow} R_2 \otimes S_2} \qquad \frac{R \otimes S \underset{f}{\Longrightarrow} T}{\overline{S \underset{curry(f)}{\Longrightarrow} R \multimap T}}$$

and subtyping rules

$$\frac{R_1 \leq_A R_2 \quad S_1 \leq_B S_2}{R_1 \otimes S_1 \leq_{A \otimes B} R_2 \otimes S_2} \qquad \frac{R_2 \leq_A R_1 \quad S_1 \leq_B S_2}{R_1 \multimap S_1 \leq_{A \multimap B} R_2 \multimap S_2}$$

Using monoidal closed bifibrations as a logical framework

Monoidal closed bifibrations

Of particular interest is when $U : \mathcal{D} \to \mathcal{T}$ is both a mc refinement system and (independently) a bifibration.
(cf. Hermida, Hasegawa, Katsumata.)

For one, we automatically get some distributivity principles:

$$\mathsf{push}_{(f \otimes g)}(R \otimes S) \equiv \mathsf{push}_f R \otimes \mathsf{push}_g S \quad (5)$$

$$\mathsf{push}_f R \multimap \mathsf{pull}_g S \equiv \mathsf{pull}_{(f \multimap g)}(R \multimap S) \quad (6)$$

But the real magic starts to happen when we combine these logical connectives with specific gadgets in $\mathcal{T}$...

From Hoare logic to separation logic

Say we want to define separating conjunction and magic wand...

$$\frac{P \sqsubset W \quad Q \sqsubset W}{P * Q \sqsubset W} \qquad \frac{}{emp \sqsubset W} \qquad \frac{P \sqsubset W \quad Q \sqsubset W}{P \mathrel{-\!\!*} Q \sqsubset W}$$

Before: W the unique object of a one-object category $\mathscr{T}$
Now: W a *monoid object* in a monoidal closed category $\mathscr{T}$![3]

$$\begin{aligned} P * Q &\stackrel{\text{def}}{=} \text{push}_m(P \otimes Q) \\ emp &\stackrel{\text{def}}{=} \text{push}_e\, I \\ P \mathrel{-\!\!*} Q &\stackrel{\text{def}}{=} \text{pull}_{curry(m)}(P \multimap Q) \end{aligned}$$

where $m : W \otimes W \to W$ and $e : I \to W$ are the monoid operations.

[3] Or a commutative monoid in a smc category if you prefer.

From Hoare logic to separation logic

Modelling[4] this signature in $\mathbf{Rel}_\bullet \to \mathbf{Rel}$...

$$h \in P * Q \iff \exists h_1, h_2.\, m(h_1, h_2, h) \wedge h_1 \in P \wedge h_2 \in Q$$
$$h \in P -\!\!* \; Q \iff \forall h', h''.\, m(h, h', h'') \wedge h' \in P \Rightarrow h'' \in Q$$

recovers the standard set-theoretic semantics of separation logic, where the relation $m : W \times W \nrightarrow W$ encodes the graph of a partial commutative monoid multiplication $m(h_1, h_2, h) \iff h_1 \circledast h_2 = h$.

---

[4]Here, a "model" is a structure-preserving morphism of refinement systems:

$$\begin{array}{ccc} \mathscr{D} & \xrightarrow{[\![-]\!]} & \mathbf{Rel}_\bullet \\ \downarrow & & \downarrow \\ \mathscr{T} & \xrightarrow[{[\![-]\!]}]{} & \mathbf{Rel} \end{array}$$

The fibrational Day construction

More generally, if $U : \mathscr{D} \to \mathscr{T}$ is a mc bifibration and A is a monoid in $\mathscr{T}$, then $\mathscr{D}_A$ is monoidal closed by:

$$R \otimes_A S \stackrel{\text{def}}{=} \text{push}_m(R \otimes S)$$
$$I_A \stackrel{\text{def}}{=} \text{push}_e I$$
$$R \multimap_A S \stackrel{\text{def}}{=} \text{pull}_{curry(m)}(R \multimap S)$$

where $m : A \otimes A \to A$ and $e : I \to A$ are the monoid operations.

(The Day construction on presheaves is an instance of this.)

Fibrational biorthogonality

Kind of similarly, if $U : \mathscr{D} \to \mathscr{T}$ is a mc fibration and $plug : A \otimes B \to C$ is any pairing operation in $\mathscr{T}$, then every refinement $\bot \sqsubset C$ induces a contravariant adjunction

$$\mathscr{D}_A \underset{{}^{\perp}(-)}{\overset{(-)^{\perp}}{\underset{\perp}{\rightleftarrows}}} \mathscr{D}_B^{\mathrm{op}}$$

where the operations $(-)^{\perp}$ and ${}^{\perp}(-)$ are defined by:

$$R^{\perp} \stackrel{\mathsf{def}}{=} \mathrm{pull}_{lcurry(plug)}(R \multimap \bot)$$

$${}^{\perp}S \stackrel{\mathsf{def}}{=} \mathrm{pull}_{rcurry(plug)}(\bot \multimapinv S)$$

Simply typed lambda calculus à la Curry (à la Scott)

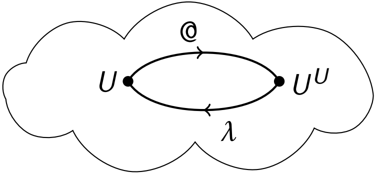
STLC can be thought of as a refinement of pure lambda calculus:

$$\begin{array}{c} \text{STLC} \\ \big\downarrow \\ \text{LC} \end{array}$$

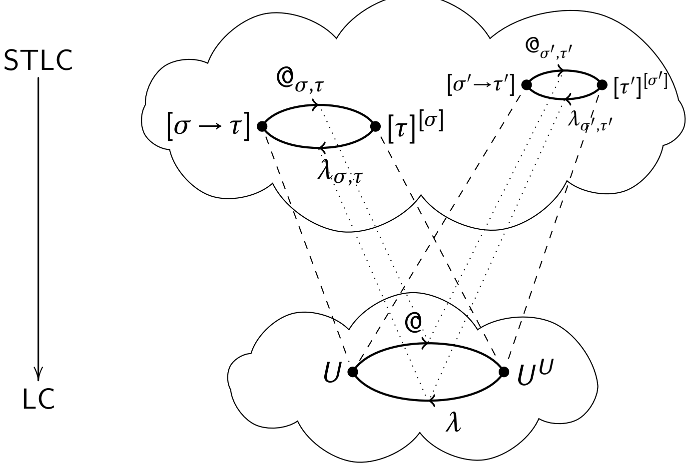
We can formalize this as a cartesian closed refinement system over the free *ccc with a reflexive object*...

Simply typed lambda calculus à la Curry (à la Scott)

LC



Simply typed lambda calculus à la Curry (à la Scott)



Simply typed lambda calculus à la Curry (à la Scott (à la Plotkin))

This definition only asks for the (LF-like) axioms[5]

$$\frac{}{[\sigma \to \tau] \underset{@}{\Longrightarrow} [\sigma] \to [\tau]}\ @_{\sigma,\tau} \qquad \frac{}{[\sigma] \to [\tau] \underset{\lambda}{\Longrightarrow} [\sigma \to \tau]}\ \lambda_{\sigma,\tau}$$

But we might impose additional conditions on models.

For example, we might interpret simple types by a *logical relation*. Abstractly, a type-indexed family $R_\sigma \sqsubset U$ is **logical** just in case

$$R_{\sigma\to\tau} \equiv \mathrm{pull}_{@}(R_\sigma \to R_\tau)$$

OTOH, we might also consider interpretations where

$$R_{\sigma\to\tau} \equiv \mathrm{push}_{\lambda}(R_\sigma \to R_\tau)$$

These Qs seem to be connected to bidirectional typing...

[5]...and perhaps corresponding β/η equations on derivations.

Conclusion

Summary:

- A (naive!) categorical perspective on extrinsic typing.
- Fibrations are fine, but we can also have fun with functors!
- Attentive to the logical interplay push/$\otimes$ *vs.* pull/$\multimap$
- Just a starting point for mathematical study.

Thanks for listening!