Type refinement systems: combining intrinsic & extrinsic views of typing, categorically*

Noam Zeilberger

7 December 2022

Proofs, Computation and Meaning #3

*based on joint work with Paul-André Melliès

Preview of the talk

Goal: describe some perspectives on deductive systems through the lens of category theory, beginning with older perspectives and then arguing for a subtle but significant change in perspective.

No category theory background will be assumed.

But first a bit of propaganda...

Category theory: a well-kept secret?

Subject: categories: A well kept secret
Date: Mon, 7 Dec 2009 09:13:28 -0500
From: "Joyal, André" <joyal.andre@uqam.ca>
To: categories list <categories@mta.ca>

Category theory is a powerful mathematical language.
It is extremely good for organising, unifying and suggesting new directions of research. It is probably the most important mathematical development of the 20th century.

But we can't say that publicly.

Why category theory?

Joyal's (just slightly cheeky) words are backed by experience, in many areas of mathematics and increasingly in other disciplines.

A major reason for why CT is so useful is (no doubt) that it studies objects via their relationships with other objects, rather than in isolation. For example, this makes CT particularly well-adapted to:

- proving that two objects are isomorphic
- transferring theorems from one area to another
- moving up and down levels of abstraction

Outline

I. Category theory and propositions-as-types

II. Unifying intrinsic and extrinsic views of typing

III. A few examples

I. Category Theory and Propositions-as-Types

Type theory

From its origins in constructive logic and philosophy, TT has grown to have an important place in the study of programming languages and proof assistants.

Partially overlapping development with category theory since at least the 1960s (some common actors).

One overarching theme of TT research is the propositions-as-types analogy, which provides a bridge with proof theory.

Propositions-as-types

Fundamentally, an analogy between programming and proving, which has been used as a guide in the design of new programming languages and proof assistants.

Also, an empirical observation of a formal correspondence between historically important systems introduced in PT and TT.

Howard (1969): there is an exact correspondence between derivations in intuitionistic natural deduction and terms in simply-typed λ-calculus.

A common abstraction?

It has been said* that "cartesian closed categories serve as a common abstraction of type theory and propositional logic."

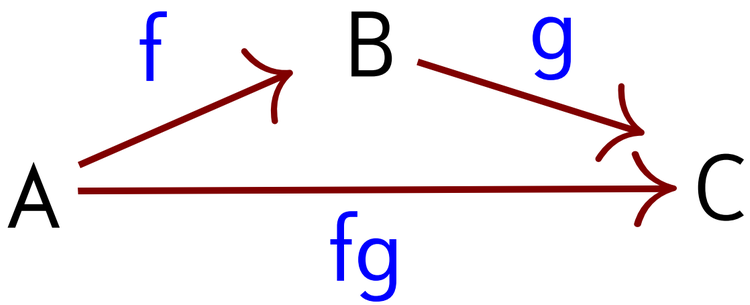
Let's try to unpack this claim...

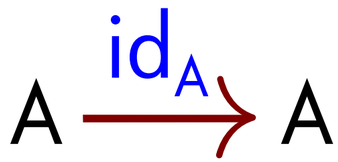
*The quote is by Bill Lawvere, from the 2006 introduction to his 1969 paper "Adjointness in Foundations", in *Reprints in Theory and Applications of Categories* 16. Lawvere mentions that he made this observation back in 1963, and it is referenced by Eilenberg and Kelly in their 1965 paper on "Closed categories", albeit without mention of type theory.

Categories (definition)

A **category** is a graph supplied with a rule for composing edges, as well as an identity edge for each node:

node = "object"
edge = "arrow" or "morphism"

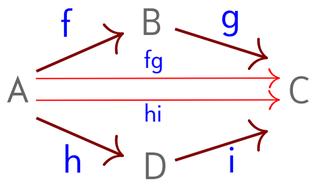




subject to associativity and unitality $(fg)h = f(gh)$ and $id_A\ f = f = f\ id_B$.
An **isomorphism** $A \approx B$ between a pair of objects in a category is a pair of arrows $f : A \to B$ and $g : B \to A$ such that $fg = id_A$ and $gf = id_B$.

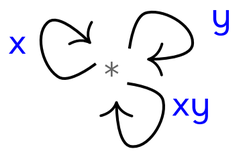
Categories (examples)

Given a graph G, the *free category* Free(G) of paths in G.



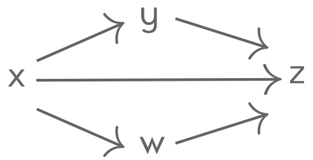
no non-trivial isomorphisms

Given a monoid M, the category B_M with one object *, an arrow $x : * \to *$ for every $x \in M$, and multiplication as composition.



isomorphisms = invertible elements

Given a preordered set P, the category P with an object x for every $x \in P$ and a unique arrow $x \to y$ iff $x \leq y$.



isomorphisms = equivalences

Categories (more examples)

Set : objects are sets, arrows are functions $f : A \to B$

Rel : objects are sets, arrows are relations $r \subseteq A \times B$

Graph : objects are graphs, arrows are homomorphisms $\varphi : G \to H$

Vec : objects are vector spaces, arrows are linear maps $\rho : V \to U$

Although these categories are "large" (= proper class of objects), they are "locally small" (= set of arrows between any pair of objects), which is often sufficient in categorical theorems.

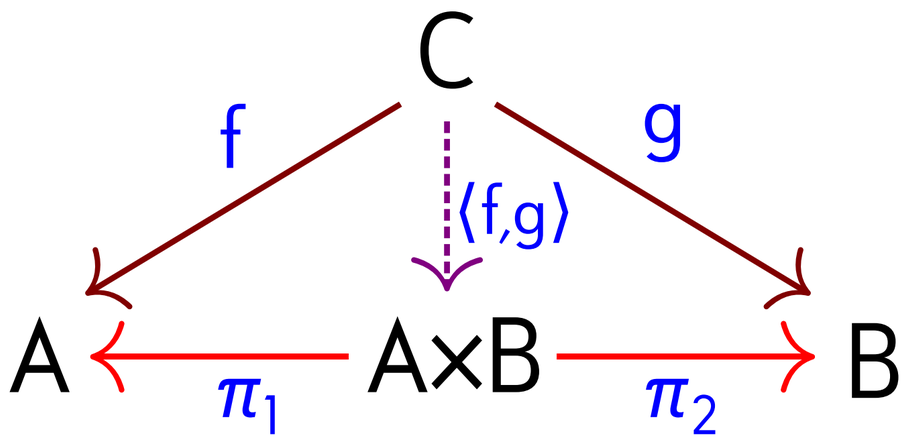
The category of categories

Categories themselves form the objects of a category Cat, whose arrows are *functors*. A **functor** $F : \mathbb{C} \to \mathbb{D}$ is a graph homomorphism that preserves composition and identities.

(As we will discuss shortly, Cat is really a "2-category" in the sense that we can naturally define arrows between functors...)

Product in a category (definition)

Let A and B be objects. A **product** of A and B is an object A×B equipped with arrows $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$, such that for any other object C and pair of arrows $f : C \to A$ and $g : C \to B$, there exists a unique $\langle f,g \rangle : C \to A \times B$ such that $\langle f,g \rangle \pi_1 = f$ and $\langle f,g \rangle \pi_2 = g$.



Product in a category (continued)

Fact: If a product of A and B exists, it is unique up to unique iso commuting with the projections. (This justifies talk of "the" product.)

Proof: Given $(A \times B, \pi_1, \pi_2)$ and $(A \times' B, \pi_1', \pi_2')$, apply the definition twice to construct arrows $h : A \times B \to A \times' B$ and $i : A \times' B \to A \times B$ commuting with the projections. This means that the compositions $hi : A \times B \to A \times B$ and $ih : A \times' B \to A \times' B$ commute with the projections. But since identity arrows also commute with the projections, we have $hi = id_{A \times B}$ and $ih = id_{A \times' B}$ by the uniqueness condition on the product.

More generally, defining objects by "universal properties" is a powerful technique for proving isomorphisms...

Natural transformations and adjunctions

Let F and G be two functors $\mathbb{C} \to \mathbb{D}$. A **natural transformation** $\theta : F \Rightarrow G$ is a family of arrows $\theta_A : F(A) \to G(A)$ in $\mathbb{D}$ such that $F(f)\theta_B = \theta_A G(f)$ for all arrows $f : A \to B$ in $\mathbb{C}$.

Given functors $L : \mathbb{C} \to \mathbb{D}$ and $R : \mathbb{D} \to \mathbb{C}$, an **adjunction** $L \dashv R$ consists of natural transformations $\eta : id_{\mathbb{C}} \Rightarrow LR$ and $\varepsilon : RL \Rightarrow id_{\mathbb{D}}$ satisfying the "triangle identities". Equivalently, $L \dashv R$ iff there is a bijection of arrows

$$L(A) \xrightarrow{f} B \quad \Leftrightarrow \quad A \xrightarrow{g} R(B)$$

natural in objects $A \in \mathbb{C}$ and $B \in \mathbb{D}$.

Cartesian closed categories (slick definition)

A category is **cartesian** if it has all finite products.

which generalize binary products in the obvious way

Eqiuvalently, $\mathbb{C}$ is cartesian iff the canonical functors

$$\mathbb{C} \xrightarrow{\Delta_{\mathbb{C}}} \mathbb{C}\times\mathbb{C} \qquad \mathbb{C} \xrightarrow{!_{\mathbb{C}}} 1$$

relying on the fact that Cat is cartesian!

have right adjoints!

A cartesian category is **closed** if for every object A, the functor $-\times A : \mathbb{C} \to \mathbb{C}$ has a right adjoint $A \Rightarrow - : \mathbb{C} \to \mathbb{C}$.

thus, the notion of ccc can be defined just using adjunctions, as highlighted by Lawvere.

Cartesian closed categories (examples)

A lattice is cartesian closed just in case it is a Heyting algebra.

$$x \wedge y \leq z \iff x \leq y \Rightarrow z$$

Set is cartesian closed, with $A \times B$ the cartesian product of sets and $A \Rightarrow B = B^A$ the set of functions from A to B.

$$A \times B \to C \iff A \to C^B$$

Graph and Cat are likewise cartesian closed.

Rel and Vec are *not* cartesian closed.

Propositions-as-types revisited

Both natural deduction and simply-typed λ-calculus can be used to define **free cccs**, where the objects are formulas/types built up from a set of atoms using conjunction/product types and implication/function types, and arrows are equivalence classes of proofs/well-typed terms modulo normalization.

The fact that the resulting cccs are free means that they somehow capture the notion of being cartesian closed...but it also means that the two presentations are isomorphic!

Is that all, really?

The notion of ccc seems to lack a bit of the flexibility of simple TT and intuitionistic ND that would merit calling it a "common abstraction".

For one, Howard's paper first described a correspondence between derivations in purely implicative ND and terms of pure STLC, before extending to other connectives. Yet the defn. of ccc relies on products.

Also, we know that propositions-as-types can be extended to substructural logics and linear lambda calculi, which do not form cccs.

Achieving a (somewhat) more perfect harmony

(promoted by Lambek)

One way of addressing this mismatch is by passing from categories to **multicategories** (or even polycategories), which generalize categories by allowing arrows with multiple inputs (and outputs).

For example, in a multicategory one can define the **tensor product** of two objects A and B as an object A⊗B equipped with a universal binary morphism m : A,B → A⊗B. Similary one can define their **internal hom** as an object A⊸B equipped with a universal binary morphism ev : A⊸B,A → B.

"universal" means that pre/post composition with m and ev sets up 1-1 correspondences of multimorphisms
Γ,A,B,Δ → C ⇔ Γ,A⊗B,Δ → C and Γ,A → B ⇔ Γ → A⊸B

More cracks in the foundations

There is a deeper issue with viewing a type system as (merely) giving rise to a category/multicategory of well-typed terms, namely that it collapses the distinction (which is important in TT) between terms, *typing judgments,* and *typing derivations*. A related issue is that it overlooks the distinction between the *intrinsic* and the *extrinsic* views of typing, seemingly rejecting the extrinsic view.

We will now explain a slight shift in perspective that accounts for these distinctions categorically, with the side-benefit of being able to model a much wider class of deductive and computational systems.

II. Unifying Intrinsic & Extrinsic Views of Typing

Intrinsic versus extrinsic typing

From "The Meaning of Types" (2000) by John C. Reynolds:

> There are two very different ways of giving denotational semantics to a programming language (or other formal language) with a nontrivial type system. ***In an intrinsic semantics, only phrases that satisfy typing judgements have meanings.*** Indeed, meanings are assigned to the typing judgements, rather than to the phrases themselves, so that a phrase that satisfies several judgements will have several meanings.
>
> In contrast, ***in an extrinsic semantics, the meaning of each phrase is the same as it would be in a untyped language,*** regardless of its typing properties. In this view, ***a typing judgement is an assertion that the meaning of a phrase possesses some property.***

An intrinsic bias

The usual way of modelling type systems by categories is biased towards the intrinsic view of typing, perhaps because every arrow of a category might be said to have intrinsic type $f : A \to B$, where $A = \mathrm{src}(f)$ and $B = \mathrm{tgt}(f)$.

Indeed, what could it mean for f to possess other types??

This seems to make it tricky to model more "extrinsic" features of type systems like subtyping and polymorphism.

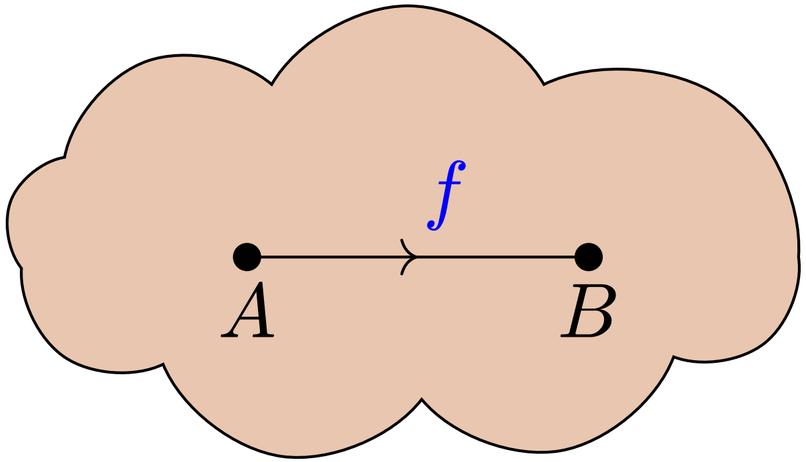
Type systems as functors

In "Functors are Type Refinement Systems" (2015), P-A Melliès and I argued that type systems are better modelled as functors $p : \mathbb{D} \to \mathbb{T}$ from a category $\mathbb{D}$ whose morphisms are typing derivations to a category $\mathbb{T}$ whose morphisms are the terms corresponding to the *underlying subjects of those derivations.*

One advantage of this perspective is that typing judgments receive a first-class mathematical status, and that typing may be naturally formulated as a "lifting problem" along the functor.

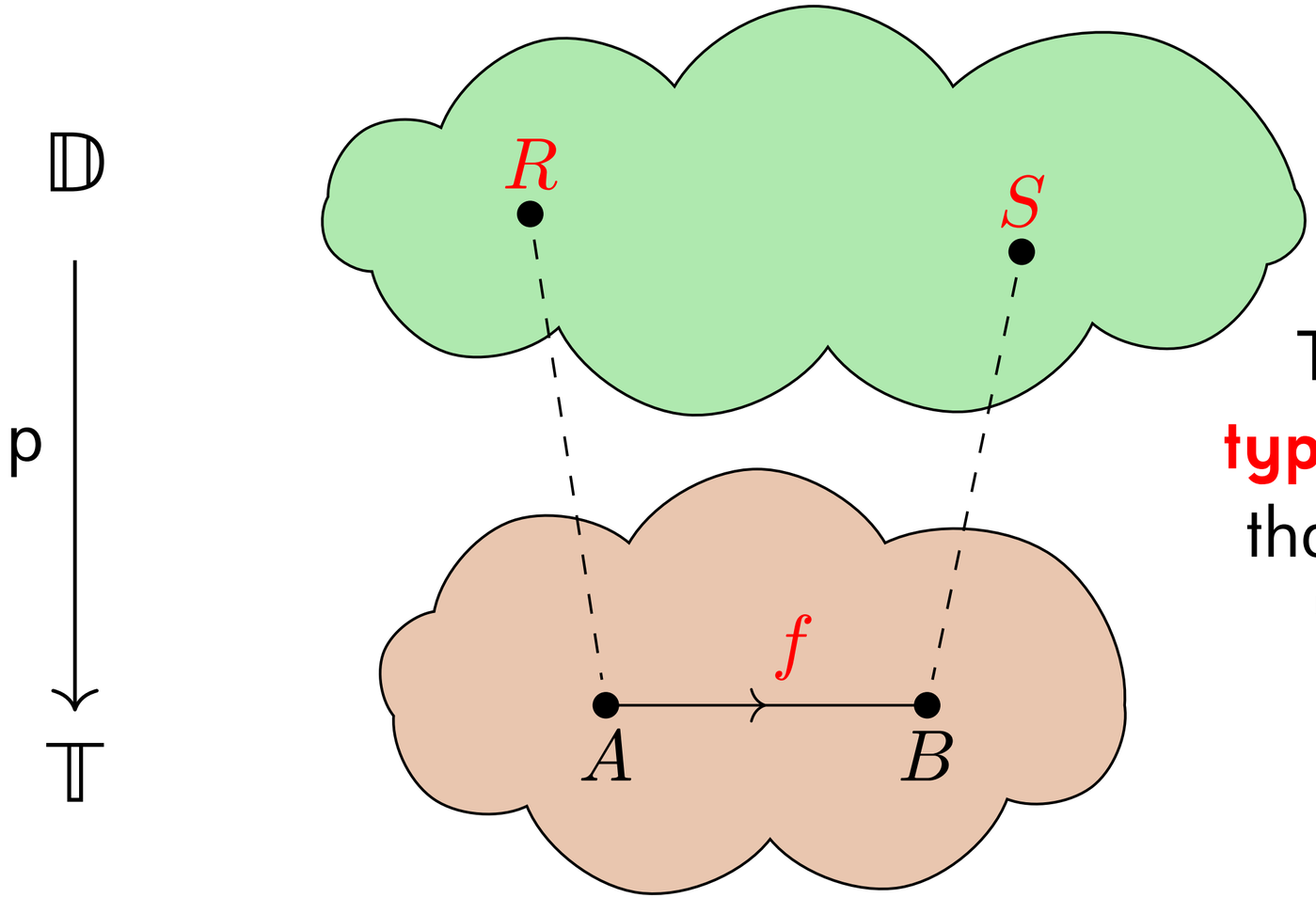
Typing as a lifting problem

$\mathbb{T}$



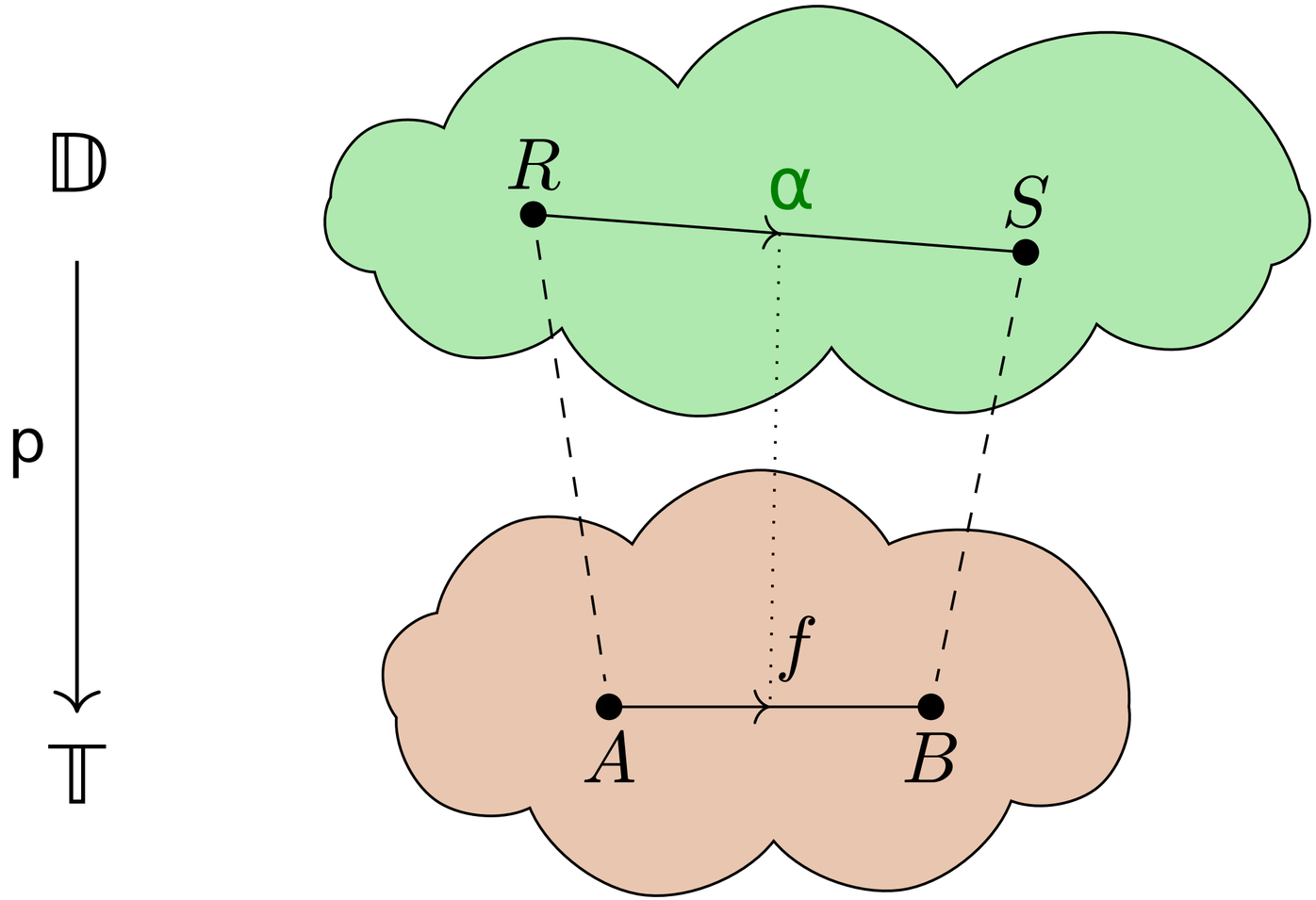
f is a **term** with
"intrinsic" type A → B

Typing as a lifting problem



The triple (R,f,S) form a **typing judgment**, asserting that f may be assigned an "extrinsic" type R → S

Typing as a lifting problem



α is a **typing derivation** providing evidence for the judgment

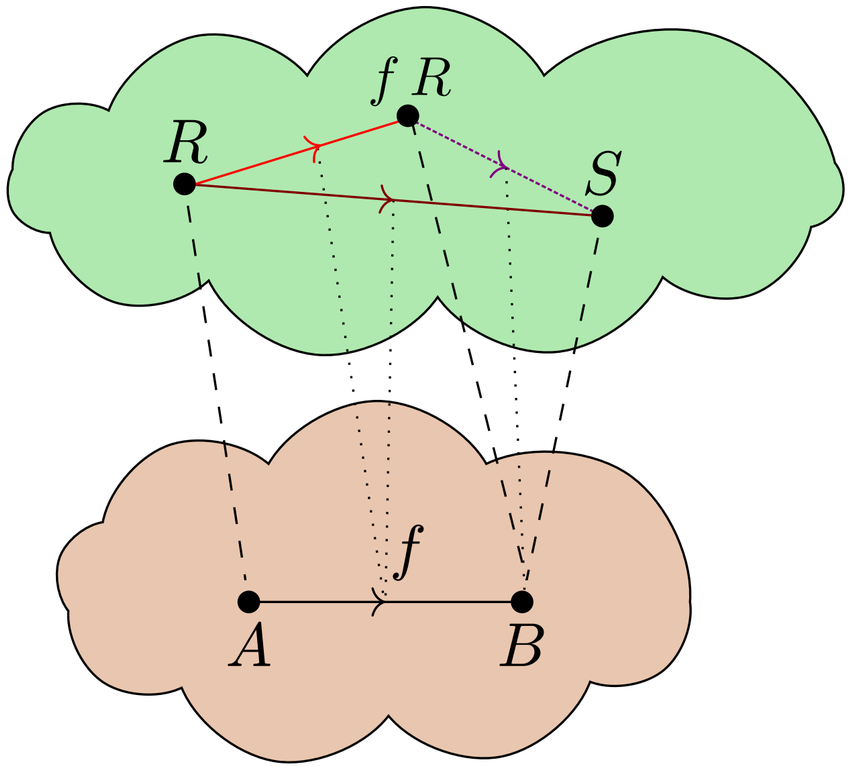
Defining type refinements up to isomorphism

A functor p : $\mathbb{D} \to \mathbb{T}$ really defines a "type refinement" system, in the sense that the types (objects) of $\mathbb{D}$ may be seen as refining the existing types (objects) of $\mathbb{T}$, and we write $R \sqsubset A$ to mean $p(R) = A$. (But an important special case is when $\mathbb{T}$ has only one type.)

As we saw, a very powerful idea in CT is to define objects by their universal properties. We can do the same to define objects $R \sqsubset A$, with even more expressive power since we are working relative to a functor rather than merely within a category.

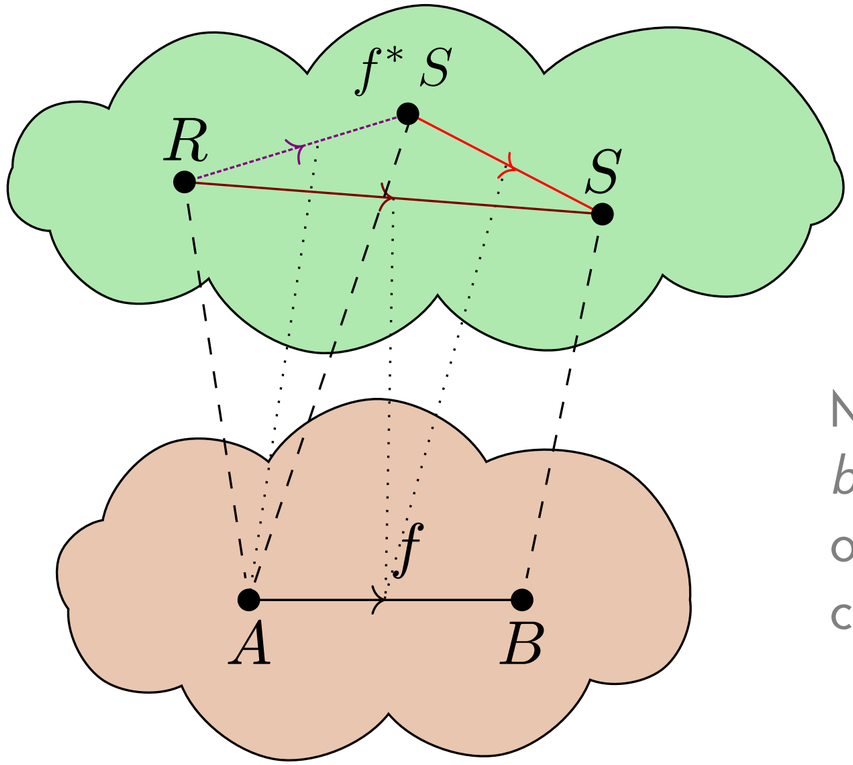
Pushforward in a type refinement system

Given R ⊏ A and f : A → B, a **pushforward of R along f** is a type f R ⊏ B equipped with a universal derivation of (R, f, f R).



Pullback in a type refinement system

Given f : A → B and S ⊏ B, a **pullback of S along f** is a type f* S ⊏ A equipped with a universal derivation of (f* S, f, S).



NB: a functor p : 𝔻 → 𝕋 is a *bifibration* if it has pushes and pulls of all objects of 𝔻 along all compatible morphisms of 𝕋.

Intersection and union types

Intersection types may be defined similarly to categorical products.

Let $R \sqsubset A$ and $S \sqsubset A$ be two refinements of the same type.
An **intersection of R and S** is an object $R \cap S \sqsubset A$ equipped with
a universal pair of derivations of $(R \cap S, id_A, R)$ and $(R \cap S, id_A, S)$.

(Union types are defined dually.)

III. A Few Examples

Example #1: Hoare Logic

Joke: "How can you tell apart a mathematician and a computer scientist? Tell them that x = x+1. The mathematician will derive a contradiction, the computer scientist will increment a register."

This joke gets at the difference between "math logic" and "CS logic", which the philosophy of type refinement systems aims to reconcile.

Hoare Logic provides an earlier, important example of an attempt at unifying math logic & CS logic.

Example #1: Hoare Logic

The basic judgment {P}c{Q} combines two formulas and a command, with the following interpretation: *if the state initially satisfies P, then after executing c it will satisfy Q.*

For example:

$\{x < 10\}x = x+1\{x \leq 10\}$ $\{x \text{ even}\}x = x+1\{x \text{ odd}\}$

Example #1: Hoare Logic

A couple of important inference rules:

$$\frac{\{P\}c_1\{Q\} \quad \{Q\}c_2\{R\}}{\{P\}c_1;c_2\{R\}}$$

(rule of sequential composition)

$$\frac{P \Rightarrow P' \quad \{P'\}c\{Q'\} \quad Q' \Rightarrow Q}{\{P\}c\{Q\}}$$

(rule of consequence)

Example #1: Hoare Logic

Hoare Logic may be naturally modelled as a functor $p : \mathbb{D} \to \mathbb{T}$!

$\mathbb{T}$: one-object category whose arrows are commands.
$\mathbb{D}$: objects are formulas, arrows $P \to Q$ are commands c s.t. $\{P\}c\{Q\}$.
p : the evident forgetful functor.

The rules of sequential composition and of consequence are semantically valid, as indeed they are for any functor $p : \mathbb{D} \to \mathbb{T}$.

Push and pull = "strongest postcondition" and "weakest precondition".

Example #2: substructural logics

We can models formulas of substructural logics as refinements (in $\mathbb{D}$) of a "type of contexts" W equipped with some algebraic structure (in $\mathbb{T}$).

For example, suppose contexts may be concatenated, modelled as an associative operation $m : W \times W \to W$ in $\mathbb{T}$. Given $A \sqsubset W$ and $B \sqsubset W$, we can define their tensor product $A \otimes B \sqsubset W$ by first forming the "external product" $A \times B \sqsubset W \times W$, then pushing forward along m, i.e., $A \otimes B \stackrel{\text{def}}{=} \text{push}(m, A \times B)$. We can similarly define the internal hom by taking the "external hom" and pulling back along the currying of m.

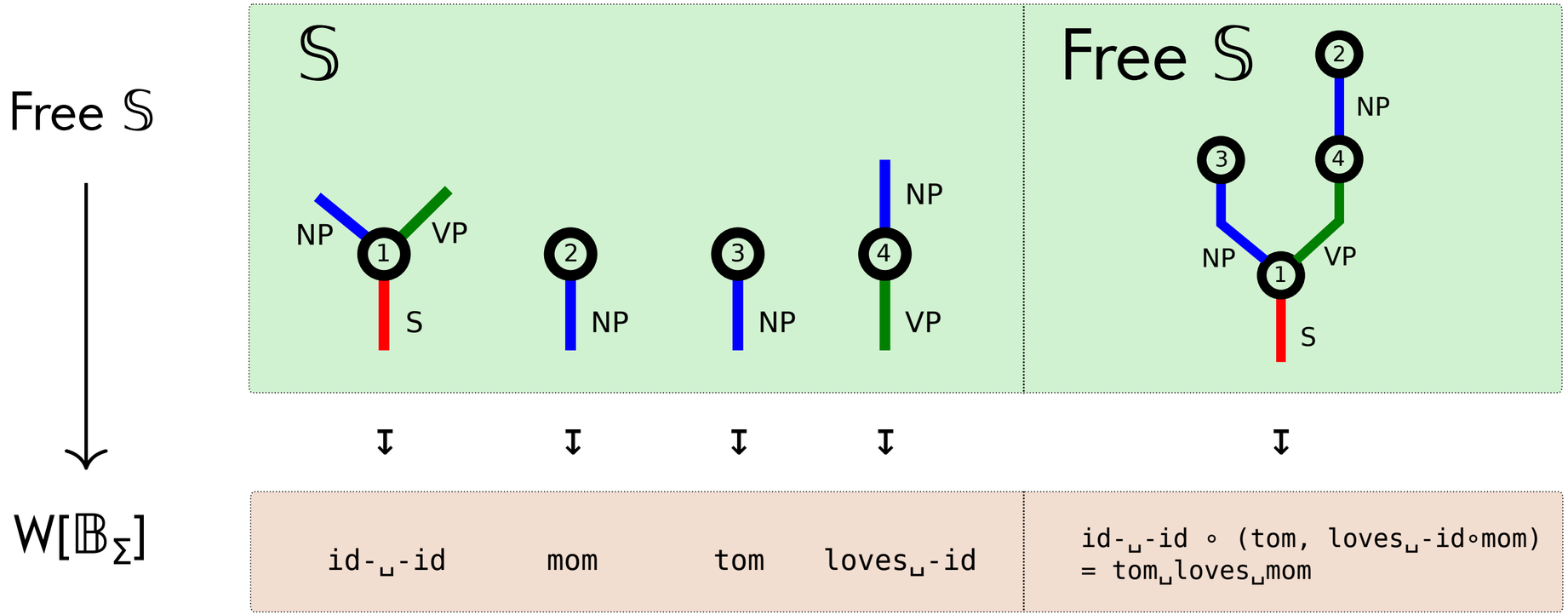
For details, see our POPL 2015 paper, and see also "A fibrational framework for substructural and modal logics" by Licata, Shulman, and Riley (FSCD 2017), which develops this idea in more depth.

Example #3: context-free grammars

In recent work (MFPS 2022), we explained how CFGs may be naturally represented as functors of multicategories, and used this to give a new proof of the Chomsky-Schützenberger Representation Theorem, which relates context-free languages and regular languages. Underlying the proof is a surprising "contour / splicing" adjunction $C \dashv W : Cat \to Multicat$ between categories and multicategories.

Example #3: context-free grammars

```
1 : S → NP VP
2 : NP → mom
3 : NP → tom
4 : VP → loves NP
```



Summary

Category theory is a very powerful tool for conceptual analysis.

this is no longer a secret!

But type systems and other deductive systems are better modelled as *functors* rather than as categories.

Doing so broadens the scope of what counts as a "deductive system", and opens up a wealth of interesting mathematical questions around type inference, proof search, parsing, etc.

Bibliography

Some classics:

F. W. Lawvere, "Adjointness in Foundations" (1969), Reprints in TAC 16 (2006)

C. A. R. Hoare, "An Axiomatic Basis for Computer Programming", CACM 12:10 (1969)

J. Lambek & P. J. Scott, *Introduction to higher order categorical logic* (1986)

J. Lambek, "Multicategories revisited", Contemporary Math. 92 (1989)

J. C. Reynolds, "The Meaning of Types: From Intrinsic to Extrinsic Semantics" (2000)

Bibliography

Papers by PAM and NZ on type refinement systems:

"Type refinement and monoidal closed bifibrations", arXiv:1310.0263

"Functors are Type Refinement Systems", POPL 2015

"An Isbell Duality Theorem for Type Refinement Systems", MSCS 28:6

"A bifibrational reconstruction of Lawvere's presheaf hyperdoctrine", LICS 2016

"Parsing as a lifting problem and the C-S representation theorem", MFPS 2022