A Categorical Perspective on Type Refinement Systems

Noam Zeilberger\textsuperscript{1}

University of Birmingham

Cambridge Logic Seminar
9 December 2016

\textsuperscript{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”!

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 \( f : A \rightarrow B \) 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:

\[
\frac{\Gamma \vdash t : A \quad A \leq B}{\Gamma \vdash t : B}
\quad
\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \land B}
\]

More fundamentally, the problem is that the naive reading does not distinguish terms from typing derivations.
Define the semantics of a typed language by induction on typing derivations, then prove a coherence theorem:

\[ \alpha \quad \beta \]
\[
\text{if } \Gamma \vdash t : A \text{ and } \Gamma \vdash t : A \text{ then } [\alpha] = [\beta] : [\Gamma] \rightarrow [A] 
\]

In general, coherence is a nontrivial theorem...


- 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
Intuitively, most type systems come with an “erasure” operation...

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

That would make a type refinement system, though, wouldn’t it? Because both $\mathcal{D}$ and $\mathcal{T}$ have types (= objects), and in some sense those of $\mathcal{D}$ “refine” those of $\mathcal{T}$...
An example

\[ \begin{align*}
\text{odd}, \text{even}, \text{pos} & \rightarrow \text{nat} \\
\text{true}, \text{false} & \rightarrow \text{bool} \\
\alpha & \rightarrow \text{id} \text{nat} \\
\beta & \rightarrow \text{succ} \\
\gamma, \delta & \rightarrow \text{isEven} \\
\text{odd} & \leq \text{nat} \text{pos} \\
\text{odd} =\Rightarrow & \text{succ} \text{even} \\
\text{even} =\Rightarrow & \text{isEven} \text{true} \\
\text{odd} =\Rightarrow & \text{isEven} \text{false} \\
\end{align*} \]
An example

\[D\]

\[\begin{align*}
\text{pos} & \xrightarrow{\text{odd}} \alpha \\
\beta & \xrightarrow{\text{even}} \gamma \\
\delta & \xrightarrow{\text{false}} \text{true}
\end{align*}\]

\[T\]

\[\begin{align*}
\text{succ} & \xrightarrow{\text{isEven}} \text{nat} \\
& \xrightarrow{\text{bool}} \text{true}, \text{false}
\end{align*}\]
An example

\[ D \]

\[ T \]

\[ \begin{align*}
\text{odd, even, pos} & \rightarrow \text{nat} \\
\text{true, false} & \rightarrow \text{bool} \\
\alpha & \rightarrow \text{id}_{\text{nat}} \\
\beta & \rightarrow \text{succ} \\
\gamma, \delta & \rightarrow \text{isEven}
\end{align*} \]
An example

\[
\begin{align*}
\alpha &: \text{odd} \leq_{\text{nat}} \text{pos} \\
\beta &: \text{odd} \xrightarrow{\text{succ}} \text{even} \\
\gamma &: \text{even} \xrightarrow{\text{isEven}} \text{true} \\
\delta &: \text{odd} \xrightarrow{\text{isEven}} \text{false}
\end{align*}
\]

\(\alpha \rightarrow \text{id}_{\text{nat}}\)
\(\beta \rightarrow \text{succ}\)
\(\gamma, \delta \rightarrow \text{isEven}\)

\(\text{odd, even, pos} \rightarrow \text{nat}\)
\(\text{true, false} \rightarrow \text{bool}\)

\(\text{(odd, even, pos} \sqsubseteq \text{nat})\)
\(\text{(true, false} \sqsubseteq \text{bool})\)
Functors are type refinement systems

\[ R \rightarrow A \]
\[ S, T \rightarrow B \]
\[ \alpha \rightarrow f \]
\[ \beta \rightarrow \text{id}_B \]

\[ \frac{\alpha}{R} \xrightarrow{f} S \]
\[ S \leq_B T \]
\[ (R \sqsubseteq A) \quad (S, T \sqsubseteq B) \]

NB: the functor \( U : \mathcal{D} \rightarrow \mathcal{T} \) need not be faithful!
The interpretation of typing rules

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

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

\[
\begin{align*}
R \implies S & \quad S \implies T \\
\quad_f & \quad \quad_g \\
\hline
R \implies T & \quad_f ;_g \\
\end{align*}
\]

\[
\begin{align*}
R \implies S & \quad S \leq T \\
\quad_f \\
\hline
R \implies T & \quad_f \\
\end{align*}
\]

\[
\begin{align*}
R \leq S & \quad S \implies T \\
\quad_g \\
\hline
R \implies T & \quad_g \\
\end{align*}
\]
A basic idea worth exploring...

P.-A. Mellisès 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: ∧ and ∨.)
3. Monoidal closed refinement systems.
4. Using monoidal closed bifibrations as a logical framework.
Reading Grothendieck in translation
Pushforward refinements

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

$$
\frac{R \sqsubseteq A \quad f : A \to B}{\text{push}_f R \sqsubseteq B}
$$

equipped with a pair of typing rules

\[
\begin{align*}
R \Rightarrow S \quad & f;g \quad S \\
R \Rightarrow \text{push}_f R \quad f \otimes l \quad & \text{push}_f R \Rightarrow S \quad f \otimes E
\end{align*}
\]
satisfying a pair of equations on typing derivations...

\[2\ldots\text{with respect to a given refinement system } U : \mathcal{D} \to \mathcal{T}.\]
Pushforward refinements

...satisfying a pair of equations on typing derivations

\[
\begin{align*}
R & \Rightarrow \text{push}_f R \\
& \xRightarrow{f \circ l} \\
f \circ g & \text{push}_f R \Rightarrow S \\
& \xRightarrow{f \circ E}
\end{align*}
\]

\[
\begin{align*}
R & \Rightarrow S \\
f \circ g & \Rightarrow S
\end{align*}
\]

\[
\begin{align*}
R & \Rightarrow \text{push}_f R \\
& \xRightarrow{f \circ l} \\
& \xRightarrow{\text{push}_f R \Rightarrow S} \\
& \xRightarrow{f \circ E}
\end{align*}
\]

\[
\begin{align*}
\text{push}_f R & \Rightarrow S \\
& \xRightarrow{f \circ E}
\end{align*}
\]
Pullback refinements

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

$$
f : A \rightarrow B \quad S \sqsubseteq B
$$

$pull_f S \sqsubseteq A$

equipped with a pair of typing rules

$$
\begin{align*}
R \Rightarrow S & \quad \quad f ; f
\end{align*}
$$

$$
\begin{align*}
pull_f S \Rightarrow S & \quad f \square E \\
g \Rightarrow pull_f S & \quad f \square I
\end{align*}
$$

satisfying the pair of equations on typing derivations...
Pullback refinements

...satisfying the pair of equations on typing derivations

\[
\begin{align*}
\beta & \quad R \Rightarrow S \\
\text{f}^\Box & \Rightarrow \text{pull}_f S \\
\text{pull}_f S & \Rightarrow S \\
\hline
\text{R} & \Rightarrow \text{S} \\
\text{f}^\Box & \Rightarrow \text{E} \\
\beta & \Rightarrow \text{g} ; \text{f} \\
\end{align*}
\]

\[
\begin{align*}
\eta & \quad R \Rightarrow \text{pull}_f S \\
\text{pull}_f S & \Rightarrow S \\
\hline
\text{R} & \Rightarrow \text{S} \\
\text{f}^\Box & \Rightarrow \text{E} \\
\eta & \Rightarrow \text{g} ; \text{f} \\
\end{align*}
\]
Proposition/Definition: A refinement system $U : \mathcal{D} \to \mathcal{T}$ is a \textbf{fibration} iff it has all pullbacks. It is an \textbf{opfibration} iff it has all pushforwards. It is a \textbf{bifibration} iff it has both.
Grothendieck remixed

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

$$\begin{align*}
R \sqsubseteq B & \quad f : A \to B \\
\text{push}_f R \sqsubseteq A & \\
R_1 \leq_A R_2 & \quad \text{push}_f R_1 \leq_B \text{push}_f R_2
\end{align*}$$

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

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

\[
\begin{align*}
R_1 & \Rightarrow R_2 & R_2 & \Rightarrow \text{push}_f R_2 & f \circ l \\
\text{id}_A & & f & & \\
\hline
R_1 & \Rightarrow \text{push}_f R_2 & & & \\
\text{id}_A;f & & & & \\
\hline
R_1 & \Rightarrow \text{push}_f R_2 & & & \\
f;\text{id}_B & & & & \\
\hline
\text{push}_f R_1 & \Rightarrow \text{push}_f R_2 & & & f \circ E
\end{align*}
\]
Moreover, we can show that

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

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 $\mathcal{D} \to \mathcal{T}$ induces a (pseudo)functor $\mathcal{T} \to \textbf{Cat}$. 
A RS that is a *bifibration* admits invertible inferences

\[
\begin{align*}
\text{push}_f \ R \leq_B \ S & \Rightarrow R \Rightarrow_S f \\
& \Rightarrow R \leq_A \text{pull}_f \ S
\end{align*}
\]

meaning that every \( f : A \rightarrow B \) gives rise to an adjunction:
Example: \textbf{SubSet} \rightarrow \textbf{Set}

Formally, the objects of \textbf{SubSet} are pairs \((A, R \subseteq A)\), its morphisms \((A, R) \rightarrow (B, S)\) are functions \(f : A \rightarrow B\) such that

\[
\forall a. a \in R \Rightarrow f(a) \in S
\]

and \(U : \textbf{SubSet} \rightarrow \textbf{Set}\) is the projection \((A, R) \mapsto A\).

Pushforward and pullback given by \textit{image} and \textit{inverse image}:

\[
push_f (A, R) = (B, \{f(a) | a \in R\})
pull_f (B, S) = (A, \{a | f(a) \in S\})
\]
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 $\mathcal{T}$ as a one-object category of commands.

Take $\mathcal{D}$ 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...

\[
\frac{R_1 \sqcup A}{R_1 \lor R_2 \sqcup A} \quad \frac{R_2 \sqcup A}{R_1 \land R_2 \sqcup A}
\]

\[
\frac{R_1 \Rightarrow S}{R_1 \lor R_2 \Rightarrow S} \quad \frac{R_2 \Rightarrow S}{R_1 \lor R_2 \Rightarrow S} \quad \lor E \quad \frac{R_i \Rightarrow R_1 \lor R_2}{\text{id}_A \lor R_1 \lor R_2} \quad \lor l_i
\]

\[
\frac{R_1 \land R_2 \Rightarrow R_i}{\text{id}_A \land R_1 \land R_2} \quad \land E_i \quad \frac{S \Rightarrow R_1}{\text{id}_A \lor R_1} \quad \frac{S \Rightarrow R_2}{\text{id}_A \lor R_2} \quad \land I
\]

...satisfying “$\beta$” and “$\eta$” equations (analogous to push/pull).
Distributivity principles

We can prove these equivalences in general:

\[ \text{push}_f(R \lor S) \equiv \text{push}_f R \lor \text{push}_f S \]  
\[ \text{pull}_g(R \land S) \equiv \text{pull}_g R \land \text{pull}_g S \]  

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

\[ \text{push}_f(R \land S) \leq \text{push}_f R \land \text{push}_f S \]  
\[ \text{pull}_g R \lor \text{pull}_g S \leq \text{pull}_g(R \lor S) \]

(Exercise: find counterexamples going backwards!)
Monoidal closed refinement systems
Monoidal closed refinement systems

The presence of push/pull/$\lor/\land$ 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 $\to$ Set and Rel. $\to$ Rel

**SubSet $\to$ Set** is a cartesian closed refinement system:

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

**Rel. $\to$ Rel** is a symmetric monoidal closed refinement system:

\[
(A, R) \otimes (B, S) = (A \times B, \{(a, b) \mid a \in R \land b \in S\})
\]
\[
(A, R) \Rightarrow (B, S) = (A \times B, \{(a, b) \mid a \in R \Rightarrow b \in S\})
\]
Refinement vs. typing vs. subtyping

A mc refinement system admits the following refinement rules

\[
\begin{align*}
R \sqsubset A & \quad S \sqsubset B & \quad R \sqsubset A & \quad S \sqsubset B \\
R \otimes S \sqsubset A \otimes B & & R \to S \sqsubset A \to B
\end{align*}
\]

and typing rules

\[
\begin{align*}
R_1 \implies f & \quad S_1 \implies g & \quad R \otimes S \implies T \\
R_1 \otimes S_1 \implies f \otimes g & & S \implies R \to T
\end{align*}
\]

and subtyping rules

\[
\begin{align*}
R_1 \leq_A R_2 & \quad S_1 \leq_B S_2 & \quad R_2 \leq_A R_1 & \quad S_1 \leq_B S_2 \\
R_1 \otimes S_1 \leq_{A \otimes_B} R_2 \otimes S_2 & & R_1 \to S_1 \leq_{A \to B} R_2 \to S_2
\end{align*}
\]
Using monoidal closed bifibrations as a logical framework
Monoidal closed bifibrations

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

For one, we automatically get some distributivity principles:

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

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

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...

\[
\begin{align*}
P \sqsubset W & \quad Q \sqsubset W \\
\frac{}{P \ast Q \sqsubset W} & \quad \text{emp} \sqsubset W \\
\frac{P \sqsubset W \quad Q \sqsubset W}{P \rightarrow* Q \sqsubset W}
\end{align*}
\]

Before: \( W \) the unique object of a one-object category \( \mathcal{T} \)
Now: \( W \) a \textit{monoid object} in a monoidal closed category \( \mathcal{T} \)\(^3\)

\[
\begin{align*}
P \ast Q & \overset{\text{def}}{=} \text{push}_m(P \otimes Q) \\
\text{emp} & \overset{\text{def}}{=} \text{push}_e l \\
P \rightarrow* Q & \overset{\text{def}}{=} \text{pull}_{\text{curry}(m)}(P \rightarrow o Q)
\end{align*}
\]

where \( m : W \otimes W \rightarrow W \) and \( e : l \rightarrow W \) are the monoid operations.

\(^3\)Or a commutative monoid in a smc category if you prefer.
From Hoare logic to separation logic

Modelling⁴ this signature in \( \text{Rel.} \rightarrow \text{Rel} \ldots \)

\[
\begin{align*}
h \in P \ast Q & \iff \exists h_1, h_2. \ m(h_1, h_2, h) \land h_1 \in P \land h_2 \in Q \\
h \in P \rightarrow Q & \iff \forall h', h''. \ m(h, h', h'') \land h' \in P \Rightarrow h'' \in Q
\end{align*}
\]

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

⁴Here, a “model” is a structure-preserving morphism of refinement systems:
The fibrational Day construction

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

$$R \otimes_A S \overset{\text{def}}{=} \text{push}_m(R \otimes S)$$

$$I_A \overset{\text{def}}{=} \text{push}_e I$$

$$R \multimap_A S \overset{\text{def}}{=} \text{pull}_{\text{curry}(m)}(R \multimap S)$$

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 $U : \mathcal{D} \to \mathcal{T}$ is a mc fibration and $plug : A \otimes B \to C$ is any pairing operation in $\mathcal{T}$, then every refinement $\bot \sqsubseteq C$ induces a contravariant adjunction

\[
\begin{array}{ccc}
\mathcal{D}_A & \bot & \mathcal{D}_B^{\text{op}} \\
\bot & \phi & \bot
\end{array}
\]

where the operations $(-)^\bot$ and $\bot(-)$ are defined by:

\[
R^\bot \overset{\text{def}}{=} \text{pull}_{\text{curry}(plug)}(R \circ \bot) \\
\bot S \overset{\text{def}}{=} \text{pull}_{\text{rcurry}(plug)}(\bot \circ S)
\]
Simply typed lambda calculus à la Curry (à la Scott)

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

\[ \text{STLC} \downarrow \text{LC} \]

We can formalize this as a cartesian closed refinement system over the free \textit{ccc with a reflexive object}...
Simply typed lambda calculus à la Curry (à la Scott)
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\(^5\)

\[
\dfrac{[\sigma \rightarrow \tau] \Rightarrow \sigma \Rightarrow \tau}{\Theta_{\sigma, \tau}} \quad \dfrac{[\sigma] \Rightarrow [\tau] \Rightarrow \sigma \Rightarrow \tau}{\Lambda_{\sigma, \tau}}
\]

But we might impose additional conditions on models.

For example, we might interpret simple types by a \textit{logical relation}. Abstractly, a type-indexed family \(R_{\sigma} \sqsubseteq U\) is \textbf{logical} just in case

\[
R_{\sigma \rightarrow \tau} \equiv \text{pull}_@ (R_{\sigma} \rightarrow R_{\tau})
\]

OTOH, we might also consider interpretations where

\[
R_{\sigma \rightarrow \tau} \equiv \text{push}_\lambda (R_{\sigma} \rightarrow R_{\tau})
\]

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

\(^5\)...and perhaps corresponding \(\beta/\eta\) equations on derivations.
Summary:

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

Thanks for listening!