Functors are Type Refinement Systems

Paul-André Melliès, Noam Zeilberger

Functors are Type Refinement Systems

Paul-André Melliès
CNRS, Université Paris Diderot
Sorbonne Paris Cité
mellies@pps.univ-paris-diderot.fr

Noam Zeilberger
MSR-Inria Joint Centre
noam.zeilberger@gmail.com

Abstract

The standard reading of type theory through the lens of category theory is based on the idea of viewing a type system as a category of well-typed terms. We propose a basic revision of this reading: rather than interpreting type systems as categories, we describe them as *functors* from a category of typing derivations to a category of underlying terms. Then, turning this around, we explain how in fact *any* functor gives rise to a generalized type system, with an abstract notion of typing judgment, typing derivations and typing rules. This leads to a purely categorical reformulation of various natural classes of type systems as natural classes of functors.

The main purpose of this paper is to describe the general framework (which can also be seen as providing a categorical analysis of *refinement types*), and to present a few applications. As a larger case study, we revisit Reynolds' paper on "The Meaning of Types" (2000), showing how the paper's main results may be reconstructed along these lines.

Categories and Subject Descriptors F.3.2 [*Semantics of Programming Languages*]

Keywords type theory; category theory; refinement types

1. Introduction

One basic difficulty with type theory as a mathematical "theory" is that in practice, the word "type" actually covers two very different usages:

1. Sometimes, like the syntactician's *parts of speech*, types serve to define the basic grammar of well-formed expressions; in this usage, all expressions carry a type, and there is no need (or even sense) to consider the meaning of "untyped" expressions.
2. Other times, like the semanticist's *predicates*, types serve as a way of identifying subsets of expressions with certain desirable properties; in this usage, every expression carries an independent meaning, and typing judgments serve to assert some property of that meaning.

POPL '15, January 15–17, 2015, Mumbai, India.

ACM 978-1-4503-3300-9/15/01. . . $15.00.
http://dx.doi.org/10.1145/2676726.2676970

These two different uses of types are often associated respectively with Alonzo Church and Haskell Curry (hence "types à la Church" and "types à la Curry"), while John Reynolds referred to these as the *intrinsic* and the *extrinsic* views of typing in his book, *Theories of Programming Languages* [30].

Traditionally, readings of type theory through the lens of category theory have sided towards the intrinsic ("à la Church") view. This is natural given the analogy

$$\text{type system} \sim \text{category}$$

which observes for example that a well-typed term

$$x_1 : A_1, \ldots, x_n : A_n \vdash e : B$$

of the simply-typed lambda calculus may be interpreted as a morphism

$$A_1 \times \cdots \times A_n \xrightarrow{\ e\ } B$$

in a cartesian-closed category [17]. This favors the intrinsic interpretation, since any morphism of a category

$$A \xrightarrow{\ f\ } B$$

is intrinsically associated with a pair of types, namely, its domain $\mathrm{dom}(f) = A$ and codomain $\mathrm{cod}(f) = B$.

On the other hand, there are type-theoretic situations where such an interpretation is plainly problematic. For example, type systems including a notion of intersection or subtyping

$$\frac{\Gamma \vdash e : B \quad \Gamma \vdash e : C}{\Gamma \vdash e : B \cap C} \qquad \frac{\Gamma \vdash e : B \quad B \leq C}{\Gamma \vdash e : C}$$

involve making multiple judgments about the same expression, but in a category, it is not even *grammatical* to write the same morphism between a different pair of objects[1]

$$* \quad \begin{array}{l} A \xrightarrow{\ f\ } B \\ A \xrightarrow{\ f\ } C \end{array}$$

What Reynolds originally observed [29, 30] is that an intrinsic semantics for such a type system must really interpret *typing derivations* rather than terms. This leads to questions of *coherence* (i.e., whether two derivations of the same typing judgment have the same meaning), and in later work [31], Reynolds gave a particularly elegant proof of coherence, as a corollary to a pair of more general results (a logical relations theorem and a "bracketing" theorem) relating an intrinsic semantics of typing derivations to an *extrinsic* semantics defined directly on untyped terms.

[1] Here and below we adopt the linguist's practice of writing an asterisk to the left of an expression which is ungrammatical (with respect to some linguistic conventions made clear from context).

Conceptually, Reynolds' intrinsic semantics may be formulated as a functor

$$[\![-]\!]_D : \text{Derivations} \to \text{Meanings}$$

from a *category* of typing derivations to some semantic category of meanings, while his extrinsic semantics may be seen as a functor

$$[\![-]\!]_T : \text{Terms} \to \text{Meanings}$$

from a category of untyped terms to the same category of meanings. On the other hand, albeit somewhat hidden in Reynolds' original analysis, implicitly there is also a "forgetful" functor

$$\mathbf{U} : \text{Derivations} \to \text{Terms}$$

from typing derivations to terms, since every typing derivation is *about* some underlying term. The logical relations and bracketing theorems can then be phrased as describing relationships among these three functors.

Our starting point here will be the observation that this analysis may be turned around: in fact, *any* functor

$$\mathbf{U} : \mathcal{D} \to \mathcal{T}$$

may be alternatively viewed as a "type system" in a generalized sense, if we interpret the (arbitrary) category $\mathcal{D}$ as a category of typing derivations and the (arbitrary) category $\mathcal{T}$ as a category of terms. This will lead us to a purely categorical way of speaking about typing derivations and terms—but also conversely to a purely type-theoretic way of speaking about functors.

In some ways, this very abstract view goes back to ideas developed after Grothendieck, in particular by Jean Bénabou, who promoted the idea that any functor may be seen as a "generalized fibration" [2] (we will describe how Grothendieck fibrations themselves can be expressed quite naturally in type-theoretic terms, as type systems with "inverse image types"). Our approach is also closely related to—and partly inspired by—the concept of *refinement* in type theory, viewing $\mathbf{U}$ as the functor which forgets refinement information. In the paper, we will adopt some of the language typically used to speak about refinement type systems [27] in order to speak about general functors—in effect providing a simple and natural categorical semantics of refinement types.

2. Reading a functor as a refinement system

For completeness and in order to fix notations, we begin by recalling the formal definitions of *category* and *functor*.

Definition 1. A **category** consists of:

- A collection of *objects* $(A, B, \dots)$.
- A collection of *morphisms* $(f, g, \dots)$, together with operations dom and cod assigning to each morphism a unique source and target. We write $f : A \to B$ to indicate that $\mathrm{dom}(f) = A$ and $\mathrm{cod}(f) = B$.
- Composition and identity: for any pair of morphisms $f : A \to B$ and $g : B \to C$, a morphism $(f; g) : A \to C$, as well as for every object A, a morphism $\mathrm{id}_A : A \to A$ (we often omit the subscript on id when it can be deduced from context).
- Such that associativity and unit laws hold:

$$(f; g); h = f; (g; h)$$
$$f; \mathrm{id} = f = \mathrm{id}; f$$

Definition 2. Let $\mathcal{D}$ and $\mathcal{T}$ be categories. We say that $\mathbf{U}$ is a **functor** from $\mathcal{D}$ to $\mathcal{T}$ when it determines the following:

- for each object S of $\mathcal{D}$, an object $\mathbf{U}(S)$ of $\mathcal{T}$, and
- for each morphism $\alpha : S \to T$ of $\mathcal{D}$, a morphism $\mathbf{U}(\alpha) : \mathbf{U}(S) \to \mathbf{U}(T)$ of $\mathcal{T}$,
- such that composition and identity are preserved:

$$\mathbf{U}(\alpha; \beta) = (\mathbf{U}(\alpha); \mathbf{U}(\beta))$$
$$\mathbf{U}(\mathrm{id}_S) = \mathrm{id}_{\mathbf{U}(S)}$$

■

Now, for the remainder of the section we will assume a fixed, arbitrary functor $\mathbf{U} : \mathcal{D} \to \mathcal{T}$, and consider various notions relative to $\mathbf{U}$.

Definition 3. We say that an object $S \in \mathcal{D}$ **refines** an object $A \in \mathcal{T}$ if $\mathbf{U}(S) = A$.

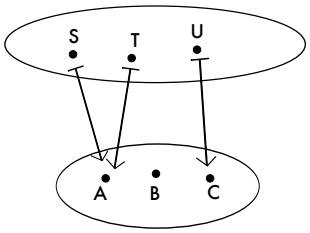
Definition 4. A **typing judgment** is a triple (S, f, T) such that S and T refine the domain and codomain of f, respectively, i.e., such that $f : A \to B$, $\mathbf{U}(S) = A$ and $\mathbf{U}(T) = B$, for some arbitrary A and B. In the special case where $f = \mathrm{id}$ (implying that $\mathbf{U}(S) = \mathbf{U}(T)$), we also call this a **subtyping judgment**.

Definition 5. A **derivation** of a typing judgment (S, f, T) is a morphism $\alpha : S \to T$ in $\mathcal{D}$ such that $\mathbf{U}(\alpha) = f$. ■

We emphasize again that these definitions are all parameterized by a fixed functor $\mathbf{U}$, and in some situations to be completely explicit we could speak of $\mathbf{U}$-refinement, $\mathbf{U}$-typing judgments, and so on. In fig. 1, we give a graphical illustration of the definitions relative to a few miniature examples.

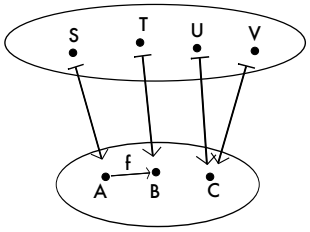
Along with these definitions, we introduce some notation (also appearing in fig. 1) and conventions inspired from logic and proof theory:

1. We write $S \sqsubset A$ to indicate that S refines A (i.e., $\mathbf{U}(S) = A$). In general, we refer to objects of $\mathcal{T}$ as *types*, to objects of $\mathcal{D}$ as *refinement types*, and to morphisms of $\mathcal{T}$ as *terms*.
2. We write
$$S \underset{f}{\Longrightarrow} T$$
to indicate that (S, f, T) is a typing judgment in the sense of Defn. 4 (i.e., $\mathbf{U}(S) = \mathrm{dom}(f)$ and $\mathbf{U}(T) = \mathrm{cod}(f)$), and
$$U \leq V$$
to indicate that (U, id, V) is a subtyping judgment (i.e., $\mathbf{U}(U) = \mathbf{U}(V)$). Since subtyping is just a special case of typing, the two judgments
$$U \leq V \quad \text{and} \quad U \underset{\mathrm{id}}{\Longrightarrow} V$$
have precisely the same meaning.
3. We write
$$S \overset{\alpha}{\underset{f}{\Longrightarrow}} T$$
to indicate that α is a derivation of the typing judgment (S, f, T) in the sense of Defn. 5 (i.e., $\alpha : S \to T$ and $\mathbf{U}(\alpha) = f$). We also write
$$\vdash S \underset{f}{\Longrightarrow} T$$
to simply indicate that such a derivation *exists* (without naming it), or
$$\nvdash S \underset{f}{\Longrightarrow} T$$
to indicate the absence of such a derivation, in which case we say that the typing judgment is *valid* or *invalid*, respectively.



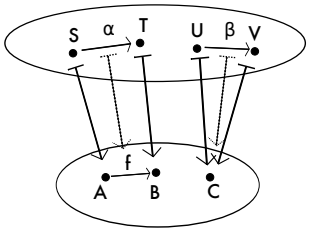
$S, T \sqsubset A$
$U \sqsubset C$

(a) Type refinement



$S \underset{f}{\Longrightarrow} T \quad U \leq V$

(b) Typing and subtyping judgments



$S \overset{\alpha}{\underset{f}{\Longrightarrow}} T \quad U \overset{\beta}{\leq} V$

(c) Derivations of typing and subtyping judgments

Figure 1: An illustration of various type-theoretic concepts associated to a functor.

4. More generally, we say that a *typing rule*

$$\frac{S_1 \underset{f_1}{\Longrightarrow} T_1 \quad \dots \quad S_n \underset{f_n}{\Longrightarrow} T_n}{S \underset{f}{\Longrightarrow} T}$$

is valid if, given derivations of the premises, we can construct a derivation of the conclusion. We will sometimes give an explicit name to a typing rule (as a way of referring to the corresponding construction on derivations), by placing it to the right of the horizontal bar. For example, any derivation can be treated as a valid typing rule with no premises:

$$S \overset{\alpha}{\underset{f}{\Longrightarrow}} T \quad \Longleftrightarrow \quad \frac{}{S \underset{f}{\Longrightarrow} T}\ \alpha$$

Proposition 6. *The following typing rules are always valid:*

$$\frac{S \underset{f}{\Longrightarrow} T \quad T \underset{g}{\Longrightarrow} U}{S \underset{f;g}{\Longrightarrow} U}\ ; \qquad \frac{}{S \underset{\mathrm{id}}{\Longrightarrow} S}\ \mathrm{id}$$

Proof. These are immediate consequences of the functoriality of $\mathbf{U}$. For example, suppose α is a derivation of (S, f, T) and β is a derivation of (T, g, U). By definition, this means that $\alpha : S \to T$ and $\mathbf{U}(\alpha) = f$, and $\beta : T \to U$ and $\mathbf{U}(\beta) = g$. But then $(\alpha; \beta)$ is a derivation of $(S, (f; g), U)$, since $(\alpha; \beta) : S \to U$ and $\mathbf{U}(\alpha; \beta) = (\mathbf{U}(\alpha); \mathbf{U}(\beta)) = (f; g)$. □

Proposition 7. *Subtyping is reflexive and transitive, and admits rules of covariant and contravariant subsumption:*

$$\frac{}{S \leq S} \qquad \frac{S \leq T \quad T \leq U}{S \leq U} \qquad \frac{S \underset{f}{\Longrightarrow} T \quad T \leq U}{S \underset{f}{\Longrightarrow} U} \qquad \frac{S \leq T \quad T \underset{g}{\Longrightarrow} U}{S \underset{g}{\Longrightarrow} U}$$

Proof. Reflexivity of subtyping is by definition just another way of writing the id typing rule of Prop. 6, while transitivity and subsumption are all special cases of ";" with one or both of the terms (i.e., morphisms of $\mathcal{T}$) f and g set to the identity term id. □

As the proof of Prop. 7 illustrates, sometimes constructing a typing derivation involves reasoning about equality of terms (i.e., morphisms of $\mathcal{T}$). In general, we allow ourselves to work modulo this equality, but for clarity it is sometimes useful to make the move between equal terms explicit by indicating a *conversion step*:

$$\frac{\dfrac{S \underset{f}{\Longrightarrow} T}{S \underset{g}{\Longrightarrow} T}\ \sim}{\vdots}$$

For example, the covariant subsumption rule of Prop. 7 can be more explicitly derived as follows:

$$\frac{\dfrac{\dfrac{S \underset{f}{\Longrightarrow} T \quad T \leq U}{S \underset{f;\mathrm{id}}{\Longrightarrow} U}\ ;}{S \underset{f}{\Longrightarrow} U}\ \sim}{\vdots}$$

Finally, we should mention that the categorical axioms also imply various equations on typing derivations. For example, the associativity axioms imply that the derivation named by

$$\frac{\dfrac{S \overset{\alpha}{\underset{f}{\Longrightarrow}} T \quad T \overset{\beta}{\underset{g}{\Longrightarrow}} U}{S \underset{f;g}{\Longrightarrow} U}\ ; \quad U \overset{\gamma}{\underset{h}{\Longrightarrow}} V}{S \underset{(f;g);h}{\Longrightarrow} V}\ ;$$

is equal to the derivation named by

$$\frac{S \overset{\alpha}{\underset{f}{\Longrightarrow}} T \quad \dfrac{T \overset{\beta}{\underset{g}{\Longrightarrow}} U \quad U \overset{\gamma}{\underset{h}{\Longrightarrow}} V}{T \underset{g;h}{\Longrightarrow} V}\ ;}{S \underset{f;(g;h)}{\Longrightarrow} V}\ ;$$

while the unit laws imply that

$$\frac{\dfrac{S \overset{\alpha}{\underset{f}{\Longrightarrow}} T \quad T \overset{\beta}{\underset{g}{\Longrightarrow}} U}{S \underset{f;g}{\Longrightarrow} U}\ ; \quad \dfrac{}{U \underset{\mathrm{id}}{\Longrightarrow} U}\ \mathrm{id}}{S \underset{(f;g);\mathrm{id}}{\Longrightarrow} U}\ ; \qquad = \qquad \frac{S \overset{\alpha}{\underset{f}{\Longrightarrow}} T \quad T \overset{\beta}{\underset{g}{\Longrightarrow}} U}{S \underset{f;g}{\Longrightarrow} U}\ ;$$

$$\frac{\dfrac{}{T \underset{\mathrm{id}}{\Longrightarrow} T}\ \mathrm{id} \quad \dfrac{T \overset{\beta}{\underset{g}{\Longrightarrow}} U \quad U \overset{\gamma}{\underset{h}{\Longrightarrow}} V}{T \underset{g;h}{\Longrightarrow} V}\ ;}{T \underset{\mathrm{id};(g;h)}{\Longrightarrow} V}\ ; \qquad = \qquad \frac{T \overset{\beta}{\underset{g}{\Longrightarrow}} U \quad U \overset{\gamma}{\underset{h}{\Longrightarrow}} V}{T \underset{g;h}{\Longrightarrow} V}\ ;$$

These various simple observations motivate our adopting the following definition:

Definition 8. A **refinement system** is a functor $\mathbf{U} : \mathcal{D} \to \mathcal{T}$.

Example 1. To try to provide a bit of intuition for this way of reading functors, we will consider a simple and naive example, which is indeed perhaps the "folk model" of refinement types. For $\mathcal{T}$ we take the category **Set** of sets and functions, while for $\mathcal{D}$ we take the category **SubSet** of *subsets and image inclusions*. An object of **SubSet** is a pair (A, S) of a set A and a subset of that set $S \subseteq A$, while a morphism

$$(A, S) \to (B, T)$$

is a function between the underlying sets

$$f : A \to B$$

such that the image of the first subset is included in the second

$$\forall a.a \in S \Rightarrow f(a) \in T$$

As the functor $\mathbf{U} : \mathbf{SubSet} \to \mathbf{Set}$ we take the first projection, sending a subset to its underlying set.

Putting aside formal questions of what exactly "sets" are (e.g., whether axiomatized by ZFC, etc.), by most interpretations, the category **Set** is already quite rich with types. For example we can probably suppose it contains types of natural numbers, integers, sequences of integers,

$$\mathbb{N}, \mathbb{Z}, \mathbb{Z}^{\mathbb{N}}$$

and many more besides. But if one could attribute a "philosophy" to type refinement, it is that rather than trying to say everything at once in the language of types, it is sometimes better to start from a rough statement, and then explore ways of making it more precise *while keeping the original statement*. So, for instance, we might consider the refinement types of *odd*, *even*, or *prime* natural numbers,[2]

$$\mathit{odd} \stackrel{\text{def}}{=} \{\, n \mid \exists k.n = 2k + 1 \,\} \sqsubset \mathbb{N}$$

$$\mathit{even} \stackrel{\text{def}}{=} \{\, n \mid \exists k.n = 2k \,\} \sqsubset \mathbb{N}$$

$$\mathit{prime} \stackrel{\text{def}}{=} \{\, n \mid n > 1 \wedge \forall k.(k > 1 \wedge k \mid n) \Rightarrow k = n \,\} \sqsubset \mathbb{N}$$

of *non-zero* or *non-negative* integers,

$$\mathit{nonzero} \stackrel{\text{def}}{=} \{\, x \mid x \neq 0 \,\} \sqsubset \mathbb{Z}$$

$$\mathit{nonneg} \stackrel{\text{def}}{=} \{\, x \mid x \geq 0 \,\} \sqsubset \mathbb{Z}$$

of *linear* or *bounded* sequences,

$$\mathit{linear} \stackrel{\text{def}}{=} \{\, f \mid \exists a, b \forall n.f(n) = a \cdot n + b \,\} \sqsubset \mathbb{Z}^{\mathbb{N}}$$

$$\mathit{bounded} \stackrel{\text{def}}{=} \{\, f \mid \exists x \forall n.f(n) \leq x \,\} \sqsubset \mathbb{Z}^{\mathbb{N}}$$

and so on. The point of the functor $\mathbf{U} : \mathbf{SubSet} \to \mathbf{Set}$ is that these refinement types (in **SubSet**) will always be considered with respect to the original types (in **Set**) they refine.

For example, the question whether "every prime number is odd" may be sensibly posed as a subtyping problem,

$$\mathit{prime} \leq \mathit{odd}$$

whose answer happens to be negative (i.e., the judgment is invalid). On the other hand, the question of whether "every linear sequence is odd" is not really sensible without resort to some encoding, and the corresponding subtyping judgment

$$* \; \mathit{linear} \leq \mathit{odd}$$

is not well-formed, since the two sides refine different types. As another example, if we take

$$\lambda x.x^2 : \mathbb{Z} \to \mathbb{Z}$$

to be the squaring function on the integers, then the following three typing judgments are respectively valid, invalid, and ill-formed:

$$\vdash \mathit{nonzero} \Longrightarrow_{\lambda x.x^2} \mathit{nonneg}$$

$$\nvdash \mathit{nonneg} \Longrightarrow_{\lambda x.x^2} \mathit{nonzero}$$

$$* \; \mathit{nonneg} \Longrightarrow_{\lambda x.x^2} \mathit{bounded}$$

3. Monoidal and logical refinement systems

In the previous section, we described how an *arbitrary* functor gives rise to a "refinement system", in the sense of an abstract notion of typing judgment, typing rules, etc. Of course, if all we could say were restricted to such generalities, then we could not say very much. But we have several strategies for carving out richer *classes* of refinement systems $\mathbf{U} : \mathcal{D} \to \mathcal{T}$:

1. By asking for additional structure on $\mathcal{D}$ and $\mathcal{T}$, and that it is preserved by $\mathbf{U}$.
2. By asking for additional properties of $\mathbf{U}$ (like for instance that it is a fibration).
3. By considering specific (refinement) type signatures, under assumption of some existing structure and properties.

In this section we will pursue the first strategy (the others will be considered later on). We begin by recalling the standard definition of a *monoidal* category:

Definition 9. A **monoidal structure** on a category $\mathcal{D}$ consists of a functor (called the tensor product)

$$- \bullet - : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$$

together with an object $I \in \mathcal{D}$, satisfying associativity and unity axioms up to natural isomorphism,

$$(A \bullet B) \bullet C \equiv A \bullet (B \bullet C) \qquad A \bullet I \equiv A \equiv I \bullet A$$

Moreover, these natural isomorphisms have to satisfy certain "coherence laws" which we omit here (see [21]). A **monoidal category** is a category equipped with a monoidal structure. ■

There are many examples of monoidal categories, and often the tensor product satisfies additional properties, such as being *symmetric* or *cartesian* [21]. However, here we just want to start from the most basic situation.

Definition 10. A **monoidal refinement system** is a functor $\mathbf{U} : \mathcal{D} \to \mathcal{T}$ between monoidal categories, preserving the monoidal structure in the strict sense that we have a pair of commutative squares:

$$\begin{array}{ccc} \mathcal{D} \times \mathcal{D} & \xrightarrow{\bullet_{\mathcal{D}}} & \mathcal{D} \\ {\scriptstyle \mathbf{U} \times \mathbf{U}}\downarrow & & \downarrow{\scriptstyle \mathbf{U}} \\ \mathcal{T} \times \mathcal{T} & \xrightarrow[\bullet_{\mathcal{T}}]{} & \mathcal{T} \end{array} \qquad \begin{array}{ccc} 1 & \xrightarrow{I_{\mathcal{D}}} & \mathcal{D} \\ \| & & \downarrow{\scriptstyle \mathbf{U}} \\ 1 & \xrightarrow[I_{\mathcal{T}}]{} & \mathcal{T} \end{array}$$

■

To read these conditions in type-theoretic language, we first allow ourselves to introduce another natural convention: we say that a *refinement rule*

$$\frac{S_1 \sqsubset A_1 \quad \ldots \quad S_n \sqsubset A_n}{S \sqsubset A}$$

[2] Here we allow ourselves the slight abuse of writing $S \sqsubset A$, although strictly speaking the pair (A, S) is the object of **SubSet**.

is valid if $\mathbf{U}(S_1) = A_1, \ldots, \mathbf{U}(S_n) = A_n$ implies that $\mathbf{U}(S) = A$. Then the commutative squares of defn. 10 translate straightforwardly to the following proposition (we omit subscripts on the monoidal operations, since they are always clear from context).

Proposition 11. *In any monoidal refinement system, the following refinement rules and typing rules are valid:*

$$\frac{S_1 \sqsubset A_1 \quad S_2 \sqsubset A_2}{S_1 \bullet S_2 \sqsubset A_1 \bullet A_2} \qquad \frac{}{I \sqsubset I} \qquad \frac{S_1 \underset{f_1}{\Longrightarrow} T_1 \quad S_2 \underset{f_2}{\Longrightarrow} T_2}{S_1 \bullet S_2 \underset{f_1 \bullet f_2}{\Longrightarrow} T_1 \bullet T_2}\ \bullet \qquad \frac{}{I \underset{I}{\Longrightarrow} I}\ I$$

Likewise, the axioms of monoidal categories translate to various equations on derivations constructed using the typing rules. We elide these here, and instead move on to considering what we call *logical* refinement systems. First we recall more standard material on category theory.

Definition 12. Let A and C be two objects of a monoidal category. A **left residual of** C **by** A is an object B' equipped with a *left-evaluation map*

$$A \bullet B' \xrightarrow{\ \oslash\ } C$$

and a transformation $\lambda[-]$ from maps

$$A \bullet B \xrightarrow{\ f\ } C \tag{1}$$

(where B is any object) to maps

$$B \xrightarrow{\ \lambda[f]\ } B' \tag{2}$$

called *left-currying*, such that for any $f : A \bullet B \to C$ and $g : B \to B'$ we have equations

$$((\mathrm{id} \bullet \lambda[f]); \oslash) = f \qquad g = \lambda[(\mathrm{id} \bullet g); \oslash]$$

These equations ensure that there is a one-to-one correspondence between maps of the form (1) and maps of the form (2). Similarly, for any two objects B and C, a **right residual of** C **by** B is an object A' equipped with a *right-evaluation map*

$$A' \bullet B \xrightarrow{\ \oslash\ } C$$

and a transformation $\rho[-]$ from maps

$$A \bullet B \xrightarrow{\ f\ } C$$

(where A is any object) to maps

$$A \xrightarrow{\ \rho[f]\ } A'$$

called *right-currying*, such that for any $f : A \bullet B \to C$ and $g : A \to A'$ we have

$$((\rho[f] \bullet \mathrm{id}); \oslash) = f \qquad g = \rho[(g \bullet \mathrm{id}); \oslash]$$

Proposition 13. *Residuation is determined up to isomorphism, i.e., if B and B' are two left residuals of C by A, then $B \cong B'$, and if A and A' are two right residuals of C by B, then $A \cong A'$.*

Because of this proposition, we allow ourselves to speak of *the* left residual of C by A whenever it exists, writing $A \setminus C$ to denote it—and similarly C / B for the right residual.

Definition 14. A **logical refinement system** is a monoidal refinement system $\mathbf{U} : \mathcal{D} \to \mathcal{T}$ which preserves left and right residuals. ∎

It is worth mentioning that the definition of a logical refinement system in our sense does not require that *all* left and right residuals exist in $\mathcal{D}$ and $\mathcal{T}$ (i.e., that the categories are *closed*), but only that $\mathbf{U}$ *preserves* any which exist in $\mathcal{D}$.

$$\frac{S \sqsubset A \quad U \sqsubset C}{S \setminus U \sqsubset A \setminus C} \qquad \frac{}{S \bullet S \setminus U \underset{\oslash}{\Longrightarrow} U}\ \oslash \qquad \frac{S \bullet T \underset{f}{\Longrightarrow} U}{T \underset{\lambda[f]}{\Longrightarrow} S \setminus U}\ \lambda$$

$$\frac{\dfrac{}{S \underset{\mathrm{id}}{\Longrightarrow} S}\ \mathrm{id} \qquad \dfrac{S \bullet T \overset{\beta}{\underset{f}{\Longrightarrow}} U}{T \underset{\lambda[f]}{\Longrightarrow} S \setminus U}\ \lambda}{\dfrac{S \bullet T \underset{\mathrm{id} \bullet \lambda[f]}{\Longrightarrow} S \bullet S \setminus U \qquad S \bullet S \setminus U \underset{\oslash}{\Longrightarrow} U}{S \bullet T \underset{(\mathrm{id} \bullet \lambda[f]); \oslash}{\Longrightarrow} U}\ ;}\ \bullet \quad = \quad S \bullet T \overset{\beta}{\underset{f}{\Longrightarrow}} U$$

$$T \overset{\eta}{\underset{g}{\Longrightarrow}} S \setminus U \quad = \quad \dfrac{\dfrac{\dfrac{}{S \leq S}\ \mathrm{id} \qquad T \overset{\eta}{\underset{g}{\Longrightarrow}} S \setminus U}{S \bullet T \underset{\mathrm{id} \bullet g}{\Longrightarrow} S \bullet S \setminus U}\ \bullet \qquad \dfrac{}{S \bullet S \setminus U \underset{\oslash}{\Longrightarrow} U}\ \oslash}{S \bullet T \underset{(\mathrm{id} \bullet g); \oslash}{\Longrightarrow} U}\ ;}{T \underset{\lambda[(\mathrm{id} \bullet g); \oslash]}{\Longrightarrow} S \setminus U}\ \lambda$$

Figure 2: The defining rules of a logical refinement system (restricted to the rules involving left residuals).

In fig. 2, we illustrate how defn. 14 (limited to the part involving left residuals) may be equivalently formulated in the language of type theory, using the appropriate refinement rules, typing rules, and equations. These rules are actually quite standard in the literature on refinement types (see, for example, the system of "simple sorts" described by Pfenning [27, §6]), except for our use of the notation of the Lambek calculus [16] (justified by the fact that we are working in a general monoidal rather than a cartesian setting). Perhaps one rule from fig. 2 that bears emphasizing is the refinement rule:

$$\frac{S \sqsubset A \quad U \sqsubset C}{S \setminus U \sqsubset A \setminus C}$$

Under the conventions we have established, the rule simply restates the condition that the functor $\mathbf{U}$ preserves left residuals. In particular, the refinement rule should *not* be confused with the familiar rule of subtyping for function types, which mixes contravariance in the domain with covariance in the codomain:

Proposition 15. *The following subtyping rules are valid in any logical refinement system, if the corresponding residuals exist:*

$$\frac{S_2 \leq S_1 \quad U_1 \leq U_2}{S_1 \setminus U_1 \leq S_2 \setminus U_2} \qquad \frac{U_1 \leq U_2 \quad T_2 \leq T_1}{U_1 \,/\, T_1 \leq U_2 \,/\, T_2}$$

Proof. We can derive the rule for left residuals as follows (the case of right residuals is symmetric):

$$\dfrac{\dfrac{\dfrac{S_2 \leq S_1 \qquad \dfrac{}{S_1 \setminus U_1 \leq S_1 \setminus U_1}\ \mathrm{id}}{S_2 \bullet S_1 \setminus U_1 \leq S_1 \bullet S_1 \setminus U_1}\ \bullet \qquad \dfrac{}{S_1 \bullet S_1 \setminus U_1 \underset{\oslash}{\Longrightarrow} U_1}\ \oslash}{S_2 \bullet S_1 \setminus U_1 \underset{\oslash}{\Longrightarrow} U_1}\ ; \qquad U_1 \leq U_2}{\dfrac{S_2 \bullet S_1 \setminus U_1 \underset{\oslash}{\Longrightarrow} U_2}{\dfrac{S_1 \setminus U_1 \underset{\lambda[\oslash]}{\Longrightarrow} S_2 \setminus U_2}{S_1 \setminus U_1 \leq S_2 \setminus U_2}\ \sim}\ \lambda}\ ;$$

□

Example 2. The refinement system **SubSet** → **Set** considered in Section 2 extends to a logical refinement system. The monoidal structure on **Set** is the usual cartesian structure,

$$A \bullet B \stackrel{\text{def}}{=} A \times B \qquad I \stackrel{\text{def}}{=} 1$$

which also lifts to a (cartesian) monoidal structure on **SubSet**:

$$(S \sqsubset A) \bullet (T \sqsubset B) \stackrel{\text{def}}{=} \{\, (a,b) \mid a \in S, b \in T \,\} \sqsubset A \times B$$
$$(I \sqsubset I) \stackrel{\text{def}}{=} \{\, * \,\} \sqsubset 1$$

Both categories are also closed, with left and right residuals both defined in terms of the function space (we describe only the underlying sets/subsets, not evaluation and currying):

$$A \setminus C \stackrel{\text{def}}{=} C^A \qquad C \,/\, B \stackrel{\text{def}}{=} C^B$$
$$(S \sqsubset A) \setminus (U \sqsubset C) \stackrel{\text{def}}{=} \{\, f \mid \forall a.a \in S \Rightarrow f(a) \in U \,\} \sqsubset C^A$$
$$(U \sqsubset C) \,/\, (T \sqsubset B) \stackrel{\text{def}}{=} \{\, f \mid \forall b.b \in T \Rightarrow f(b) \in U \,\} \sqsubset C^B$$

The forgetful functor **SubSet** → **Set** evidently sends products and residuals in **SubSet** to products and residuals in **Set**, and thus defines a logical refinement system.

As an example, writing $+ : \mathbb{N} \bullet \mathbb{N} \to \mathbb{N}$ for addition of natural numbers, we can state various easy arithmetic facts and non-facts as valid and invalid judgments:

$$\vdash \mathit{odd} \underset{\lambda[+]}{\Longrightarrow} \mathit{odd} \setminus \mathit{even}$$
$$\vdash \mathit{odd} \underset{\lambda[+]}{\Longrightarrow} \mathit{even} \setminus \mathit{odd}$$
$$\nvdash \mathit{even} \underset{\lambda[+]}{\Longrightarrow} \mathit{prime} \setminus \mathit{odd}$$

4. Reading Grothendieck in translation

In this section we will pursue the second strategy mentioned at the beginning of Section 3, and begin by recalling the definition of when a functor $\mathbf{U} : \mathcal{D} \to \mathcal{T}$ is a *fibration* à la Grothendieck [13].

Definition 16. A morphism $\alpha : T' \to T$ in $\mathcal{D}$ is said to be **cartesian** if for every object $S \in \mathcal{D}$ and every pair of morphisms $\beta : S \to T$ and $g : \mathbf{U}(S) \to \mathbf{U}(T')$ such that $\mathbf{U}(\beta) = g; \mathbf{U}(\alpha)$, there is a unique morphism $\beta' : S \to T'$ such that $\beta = \alpha; \beta'$ and $\mathbf{U}(\beta') = g$.

Definition 17. Let $f : A \to B$ be a morphism in $\mathcal{T}$, and T be an object of $\mathcal{D}$ such that $\mathbf{U}(T) = B$. A morphism α in $\mathcal{D}$ is said to be a **cartesian lifting** of f to T if $\mathbf{U}(\alpha) = f$, $\mathrm{cod}(\alpha) = T$, and α is cartesian.

Definition 18. A functor $\mathbf{U} : \mathcal{D} \to \mathcal{T}$ is said to be a **fibration** if for every morphism $f : A \to B$ in $\mathcal{T}$ and object $T \in \mathcal{D}$ such that $\mathbf{U}(T) = B$, f has a cartesian lifting to T.

The definition of fibration plays a fundamental role in category theory as well as in the semantics of dependent types, and we may thus wonder whether we can understand it from our point of view on refinement type systems. Again, we assume $\mathbf{U} : \mathcal{D} \to \mathcal{T}$ is fixed, with the notational and terminological conventions of Section 2 (we do not assume any other structure on $\mathcal{D}$ and $\mathcal{T}$).

Definition 19. Let $f : A \to B$ and $T \sqsubset B$. A **pullback** (or "inverse image") **of** T **along** f is a refinement type $T' \sqsubset A$ equipped with a pair of valid typing rules

$$\frac{}{T' \underset{f}{\Longrightarrow} T}\, L \qquad \frac{S \underset{g;f}{\Longrightarrow} T}{S \underset{g}{\Longrightarrow} T'}\, R$$

such that for all derivations

$$S \underset{g;f}{\overset{\beta}{\Longrightarrow}} T \quad \text{and} \quad S \underset{g}{\overset{\eta}{\Longrightarrow}} T'$$

we have a pair of equalities

$$\frac{\dfrac{S \underset{g;f}{\overset{\beta}{\Longrightarrow}} T}{S \underset{g}{\Longrightarrow} T'}\, R \qquad \dfrac{}{T' \underset{f}{\Longrightarrow} T}\, L}{S \underset{g;f}{\Longrightarrow} T}\, ; \quad = \quad S \underset{g;f}{\overset{\beta}{\Longrightarrow}} T$$

and

$$S \underset{g}{\overset{\eta}{\Longrightarrow}} T' \quad = \quad \frac{S \underset{g}{\overset{\eta}{\Longrightarrow}} T' \qquad \dfrac{}{T' \underset{f}{\Longrightarrow} T}\, L}{S \underset{g;f}{\Longrightarrow} T}\, ; \qquad \frac{}{S \underset{g}{\Longrightarrow} T'}\, R$$

■

Now, it is essentially immediate by unwinding the definitions that we have just redubbed Grothendieck to a type-theoretic soundtrack:

Proposition 20. *$\alpha : T' \to T$ is a cartesian lifting of f to T if and only if the triple (T', L_α, R_α) is a pullback of T along f, where $L_\alpha = \alpha$, and where R_α is defined using the universal property of α.*

Proposition 21. **U** *is a fibration iff for every $f : A \to B$ and $T \sqsubset B$, there exists a pullback of T along f.*

Somewhat remarkably, many standard facts about fibrations can be derived quite mechanically under this translation, reminiscent of proofs in the sequent calculus. We begin by showing in this proof-theoretic style that pullbacks are determined up to *vertical isomorphism*.

Definition 22. Let $S, T \sqsubset A$ be two refinements of a common type. We say that S and T are **vertically isomorphic** (written $S \equiv T$) when there exist a pair of subtyping derivations

$$S \overset{\alpha}{\leq} T \qquad T \overset{\beta}{\leq} S$$

which compose to the identity

$$\frac{S \overset{\alpha}{\leq} T \quad T \overset{\beta}{\leq} S}{S \leq S}\, ; \quad = \quad \frac{}{S \leq S}\,\mathrm{id} \qquad \frac{T \overset{\beta}{\leq} S \quad S \overset{\alpha}{\leq} T}{T \leq T}\, ; \quad = \quad \frac{}{T \leq T}\,\mathrm{id}$$

Proposition 23. *Any two pullbacks of T along f are vertically isomorphic.*

Proof. Let T' and T'' both be pullbacks of T along f, equipped with corresponding valid typing rules

$$\frac{}{T' \underset{f}{\Longrightarrow} T}\, LT' \qquad \frac{S \underset{g;f}{\Longrightarrow} T}{S \underset{g}{\Longrightarrow} T'}\, RT'$$

and

$$\frac{}{T'' \underset{f}{\Longrightarrow} T}\, LT'' \qquad \frac{S \underset{g;f}{\Longrightarrow} T}{S \underset{g}{\Longrightarrow} T''}\, RT''$$

Then we can build derivations of $T' \leq T''$ and $T'' \leq T'$ by

$$\frac{\dfrac{}{T' \underset{f}{\Longrightarrow} T}\, LT' \qquad \dfrac{}{T'' \underset{f}{\Longrightarrow} T}\, LT''}{T' \leq T''}\, RT'' \qquad \frac{\dfrac{}{T'' \underset{f}{\Longrightarrow} T}\, LT'' \qquad \dfrac{}{T' \underset{f}{\Longrightarrow} T}\, LT'}{T'' \leq T'}\, RT'$$

$$\frac{f : A \to B \quad T \sqsubset B}{f^* T \sqsubset A} \qquad \frac{}{f^* T \underset{f}{\Longrightarrow} T}\, Lf^* \qquad \frac{S \underset{g;f}{\Longrightarrow} T}{S \underset{g}{\Longrightarrow} f^* T}\, Rf^*$$

$$\frac{\dfrac{S \underset{g;f}{\overset{\beta}{\Longrightarrow}} T}{S \underset{g}{\Longrightarrow} f^* T}\, Rf^* \quad \dfrac{}{f^* T \underset{f}{\Longrightarrow} T}\, Lf^*}{S \underset{g;f}{\Longrightarrow} T}\, ; \quad = \quad S \underset{g;f}{\overset{\beta}{\Longrightarrow}} T \quad S \underset{g}{\overset{\eta}{\Longrightarrow}} f^* T \quad = \quad \frac{S \underset{g}{\overset{\eta}{\Longrightarrow}} f^* T \quad \dfrac{}{f^* T \underset{f}{\Longrightarrow} T}\, Lf^*}{S \underset{g;f}{\Longrightarrow} T}\, ; \quad \frac{}{S \underset{g}{\Longrightarrow} f^* T}\, Rf^*$$

Figure 3: The defining rules of pullback refinements.

$$\frac{S \sqsubset A \quad f : A \to B}{f\,S \sqsubset B} \qquad \frac{S \underset{f;g}{\Longrightarrow} T}{f\,S \underset{g}{\Longrightarrow} T}\, Lf \qquad \frac{}{S \underset{f}{\Longrightarrow} f\,S}\, Rf$$

$$\frac{\dfrac{S \underset{f;g}{\overset{\beta}{\Longrightarrow}} T}{S \underset{f}{\Longrightarrow} f\,S}\, Rf \quad \dfrac{f\,S \underset{g}{\Longrightarrow} T}{f\,S \underset{g}{\Longrightarrow} T}\, Lf}{S \underset{f;g}{\Longrightarrow} T}\, ; \quad = \quad S \underset{f;g}{\overset{\beta}{\Longrightarrow}} T \quad f\,S \underset{g}{\overset{\eta}{\Longrightarrow}} T \quad = \quad \frac{\dfrac{}{S \underset{f}{\Longrightarrow} f\,S}\, Rf \quad f\,S \underset{g}{\overset{\eta}{\Longrightarrow}} T}{S \underset{f;g}{\Longrightarrow} T}\, ; \quad \frac{S \underset{f;g}{\Longrightarrow} T}{f\,S \underset{g}{\Longrightarrow} T}\, Lf$$

Figure 4: The defining rules of pushforward refinements.

and easily verify from the axioms that these two derivations compose to the identity. □

Because pullbacks are determined up to vertical isomorphism, we allow ourselves to speak of *the* pullback of T along f whenever one exists, writing $f^* T$ for the refinement type and Lf^* and Rf^* for the corresponding rules (see fig. 3).

We can now mechanically establish the following facts about pullbacks, which, in categorical jargon, go into showing that any fibration determines a pseudofunctor $\mathcal{T}^{\mathrm{op}} \to \mathbf{Cat}$:

Proposition 24. *Whenever the corresponding pullbacks exist:*

1. *the following subtyping rule is valid:*
$$\frac{T_1 \le T_2}{f^* T_1 \le f^* T_2}$$
2. *we have vertical isomorphisms*
$$(g;f)^* T \equiv g^* f^* T \qquad \mathrm{id}^* T \equiv T$$

Proof. 1.
$$\frac{\dfrac{}{f^* T_1 \underset{f}{\Longrightarrow} T_1}\, Lf^* \quad T_1 \le T_2}{\dfrac{f^* T_1 \underset{f;\mathrm{id}}{\Longrightarrow} T_2}{\dfrac{f^* T_1 \underset{\mathrm{id};f}{\Longrightarrow} T_2}{f^* T_1 \le f^* T_2}\, Rf^*}\, \sim}\, ;$$

2. For the left equation, we construct subtyping derivations in both directions by
$$\frac{\dfrac{\dfrac{}{(g;f)^* T \underset{g;f}{\Longrightarrow} T}\, L(g;f)^*}{(g;f)^* T \underset{g}{\Longrightarrow} f^* T}\, Rf^*}{(g;f)^* T \le g^* f^* T}\, Rg^* \qquad \frac{\dfrac{\dfrac{}{g^* f^* T \underset{g}{\Longrightarrow} f^* T}\, Lg^* \quad \dfrac{}{f^* T \underset{f}{\Longrightarrow} T}\, Lf^*}{g^* f^* T \underset{g;f}{\Longrightarrow} T}\, ;}{g^* f^* T \le (g;f)^* T}\, R(g;f)^*$$
and again by an easy calculation, we can show that these two derivations compose to the identity. The right equation $\mathrm{id}^* T \equiv T$ is essentially immediate (which also means that pullbacks along the identity always exist). □

Next, we give an analogous reconstruction of the dual concept of an *opfibration*.

Definition 25. Let $S \sqsubset A$ and $f : A \to B$. A **pushforward** (or "image") **of** S **along** f is a refinement type $S' \sqsubset B$ equipped with a pair of valid typing rules
$$\frac{S \underset{f;g}{\Longrightarrow} T}{S' \underset{g}{\Longrightarrow} T}\, L \qquad \frac{}{S \underset{f}{\Longrightarrow} S'}\, R$$
such that for all derivations
$$S \underset{f;g}{\overset{\beta}{\Longrightarrow}} T \quad \text{and} \quad S' \underset{g}{\overset{\eta}{\Longrightarrow}} T$$
we have equalities
$$\frac{\dfrac{}{S \underset{f}{\Longrightarrow} S'}\, R \quad \dfrac{S \underset{f;g}{\overset{\beta}{\Longrightarrow}} T}{S' \underset{g}{\Longrightarrow} T}\, L}{S \underset{f;g}{\Longrightarrow} T}\, ; \quad = \quad S \underset{f;g}{\overset{\beta}{\Longrightarrow}} T$$
and
$$S' \underset{g}{\overset{\eta}{\Longrightarrow}} T \quad = \quad \frac{\dfrac{}{S \underset{f}{\Longrightarrow} S'}\, R \quad S' \underset{g}{\overset{\eta}{\Longrightarrow}} T}{S \underset{f;g}{\Longrightarrow} T}\, ; \quad \frac{S \underset{f;g}{\Longrightarrow} T}{S' \underset{g}{\Longrightarrow} T}\, L$$

Proposition 26. **U** *is a Grothendieck opfibration iff for every* $S \sqsubset A$ *and* $f : A \to B$*, there exists a pushforward of* S *along* f*.*

Since pushforwards are determined up to vertical isomorphism, we speak of *the* pushforward of S along f, writing $f\,S$ for the refinement type and Lf and Rf for the corresponding rules (see fig. 4). Again, we can mechanically establish some basic facts about pushforwards (which go into showing that any opfibration determines a pseudofunctor $\mathcal{T} \to \mathbf{Cat}$):

Proposition 27. *Whenever the corresponding pushforwards exist:*

1. *the following subtyping rule is valid:*
$$\frac{S_1 \le S_2}{f\,S_1 \le f\,S_2}$$
2. *we have vertical isomorphisms*
$$(f;g)\,S \equiv g\,f\,S \qquad \mathrm{id}\,S \equiv S$$

Proposition 28. *Whenever the respective pushforwards and pullbacks exist, we have a three-way correspondence of interderivability,*
$$\vdash f\,S \le T \quad \textit{iff} \quad \vdash S \underset{f}{\Longrightarrow} T \quad \textit{iff} \quad \vdash S \le f^* T$$

Example 3. For the refinement system **SubSet** → **Set**, pushforward and pullback refinements may be constructed as suggested by the notation, via image and inverse image operations on subsets (along any function $f : A \to B$):
$$f\,S \overset{\mathrm{def}}{=} \{\, f(a) \mid a \in S \,\}$$
$$f^* T \overset{\mathrm{def}}{=} \{\, a \mid f(a) \in T \,\}$$

For example, the typing judgment

$$f^* T \underset{f}{\Longrightarrow} T$$

is obviously valid, reading as « f maps anything in the inverse image of T along f to something in T », while

$$S \underset{f}{\Longrightarrow} f\, S$$

reads as « f maps anything in S to something in the image of S along f ». Since these operations are defined for any $f : A \to B$, $S \sqsubset A$, and $T \sqsubset B$, the functor **SubSet** $\to$ **Set** is both a fibration and an opfibration, i.e., a *bifibration*.

Example 4. The general approach of Hoare logic [12] provides a natural class of examples of refinement systems, to a first approximation defined as follows (we will consider a more nuanced view in Section 5):

- Take $\mathcal{T}$ as a category with one object W corresponding to the state space, and with morphisms $c : W \to W$ corresponding to program commands, identified with state transformers.
- Take $\mathcal{D}$ as a category whose objects are predicates ϕ over states, and whose morphisms $\phi \to \psi$ are pairs of a state transformer c together with a verification that c takes any state satisfying ϕ to a state satisfying ψ.
- Let $\mathbf{U} : \mathcal{D} \to \mathcal{T}$ be the evident forgetful functor, mapping every ϕ to W and every verification about c to c itself.

Indeed, the induced notion of typing judgment for the functor $\mathbf{U} : \mathcal{D} \to \mathcal{T}$ corresponds exactly to the classical notion of Hoare triple $\{\phi\}c\{\psi\}$. One easily checks that the usual rules of sequential composition, pre-strengthening and post-weakening are valid by Propositions 6 and 7, and moreover that a pullback of ψ along c is precisely a *weakest precondition*, while a pushforward of ϕ along c is a *strongest postcondition*:

$$wp(c, \psi) = c^*\, \psi$$
$$sp(c, \phi) = c\, \phi$$

On the other hand, it is not necessarily the case that $\mathcal{D} \to \mathcal{T}$ is a fibration and/or opfibration: whether weakest preconditions/strongest postconditions exist for *all* predicates and state transformers depends on the specifics of the class of predicates and the class of state transformers.

Example 5. The example of **SubSet** $\to$ **Set** can be generalized in terms of *enriched category theory* [15]. Let $(\mathcal{V}, \otimes_{\mathcal{V}}, I_{\mathcal{V}}, \multimap_{\mathcal{V}})$ be a symmetric monoidal closed category, let $\mathcal{V}$**Cat** be the (bi)category of $\mathcal{V}$-enriched categories, and let $\mathcal{V}$**Psh** the category of $\mathcal{V}$-presheaves, i.e., the category whose objects are $\mathcal{V}$-valued functors $S : A \to \mathcal{V}$ out of $\mathcal{V}$-enriched categories, and where a morphism from $S : A \to \mathcal{V}$ to $T : B \to \mathcal{V}$ is a pair of a ($\mathcal{V}$-)functor $f : A \to B$ together with a natural transformation $\alpha : S \Rightarrow (f; T)$. Then the refinement system given by the domain functor dom : $\mathcal{V}$**Psh** $\to$ $\mathcal{V}$**Cat** is a bifibration, with pullbacks simply defined by precomposition, and pushforwards computed as *coends*:

$$f^*\, T \stackrel{\text{def}}{=} a \mapsto T(fa)$$
$$f\, S \stackrel{\text{def}}{=} b \mapsto \int^a B(fa, b) \otimes_{\mathcal{V}} T(a)$$

Note that this is also an example of a logical refinement system—the (symmetric) closed monoidal structure on $\mathcal{V}$**Cat** is defined by constructing tensor product categories and functor categories

$$A \bullet B \stackrel{\text{def}}{=} A \otimes B \qquad A \setminus C \stackrel{\text{def}}{=} [A, C] \qquad C \,/\, B \stackrel{\text{def}}{=} [B, C]$$

while the tensor product of two presheaves is defined as their *external tensor product*,

$$S \bullet T : A \otimes B \to \mathcal{V}$$
$$S \bullet T \stackrel{\text{def}}{=} (a, b) \mapsto S(a) \otimes_{\mathcal{V}} T(b)$$

and the left and right residuals defined as *ends*:

$$S \setminus U : [A, C] \to \mathcal{V}$$
$$S \setminus U \stackrel{\text{def}}{=} f \mapsto \int_a S(a) \multimap_{\mathcal{V}} U(fa)$$
$$U \,/\, T : [B, C] \to \mathcal{V}$$
$$U \,/\, T \stackrel{\text{def}}{=} g \mapsto \int_b T(b) \multimap_{\mathcal{V}} U(gb)$$

Example 6. A trivial example of a bifibration is the unique functor $! : \mathcal{D} \to 1$ from any category $\mathcal{D}$ to the terminal category 1. Since there is only the identity arrow in 1, all pushforwards and pullbacks exist trivially. (If $\mathcal{D}$ is monoidal, this is also trivially a logical refinement system.) ■

5. Separation Logic and the Frame Rule

We have seen how a lot of general type theory can be reconstructed as a "theory of functors". In many ways, though, the really interesting phenomena arise by taking the various type constructors as building blocks, and using them to define specific *type signatures*. Effectively, this is a way of viewing refinement systems as a "logical framework", using them both to define theories and to construct models. In this section we will give some basic examples, describing how some aspects of Reynolds and O'Hearn's *separation logic* [32] can be usefully explained in terms of refinement systems.

Recall (Example 4) that Hoare logic may be considered as a refinement system where terms $c : W \to W \in \mathcal{T}$ are commands (state-transformers), refinements $\phi, \psi \sqsubset W$ are predicates over the state space, and where derivations

$$\phi \underset{c}{\overset{\alpha}{\Longrightarrow}} \psi$$

are proofs that the command c will take any state satisfying ϕ to a state satisfying ψ. Although this description suggests that $\mathcal{T}$ is a one-object category, such a restriction is not really necessary, and it turns out to be useful to work more generally.

In particular, suppose we know that $\mathcal{T}$ is a monoidal category and that W is a *monoid object* in $\mathcal{T}$, i.e., that it is equipped with operations

$$\circledast : W \bullet W \to W$$
$$e : 1 \to W$$

satisfying the monoid axioms. Then for any pair of refinements $\phi, \psi \sqsubset W$, we can define their *separating conjunction* $\phi * \psi \sqsubset W$ as a pushforward (along $\circledast$) of a tensor product:

$$\phi * \psi \stackrel{\text{def}}{=} \circledast\, (\phi \bullet \psi)$$

We similarly define the unit of the separating conjunction $emp \sqsubset W$ as a pushforward (along e) of the tensor unit:

$$emp \stackrel{\text{def}}{=} e\, I$$

Finally, for any $\phi, \tau \sqsubset W$ we define "magic wand" $\phi \mathrel{-\!\!*} \tau \sqsubset W$ as a pullback (along the currying of $\circledast$) of a residual:[3]

$$\phi \mathrel{-\!\!*} \tau \stackrel{\text{def}}{=} \lambda[\circledast]^*\, (\phi \setminus \tau)$$

[3] Incidentally, these kinds of definitions—where in order to define some logical structure of interest we rely on a similar structure in the

Now, interpreting this signature in the refinement system **SubSet** → **Set** (Examples 1 to 3) yields the basic set-theoretic semantics of the separation logic connectives:

$$\phi * \psi = \{\, w_1 \circledast w_2 \mid w_1 \in \phi, w_2 \in \psi \,\}$$
$$emp = \{\, e \,\}$$
$$\phi \mathrel{-\!\!*} \tau = \{\, w \mid \forall w'.\ w' \in \phi \Rightarrow w' \circledast w \in \tau \,\}$$

On the other hand, we can see that the abstract definition in terms of refinement systems is much more general. For example, interpreting the signature in $\mathcal{V}$**Psh** → $\mathcal{V}$**Cat** (Example 5) recovers the well-known *Day construction* for lifting a monoidal structure on a category to a closed monoidal structure on its category of presheaves. The next proposition describes the situation more abstractly.

Proposition 29. *Whenever the operations $\phi * -$ and $\phi \mathrel{-\!\!*} -$ are defined (i.e., when the corresponding pushforwards, pullbacks, and residuals exist), they are functorial in the sense that subtyping rules*

$$\frac{\psi_1 \le \psi_2}{\phi * \psi_1 \le \phi * \psi_2} \qquad \frac{\tau_1 \le \tau_2}{\phi \mathrel{-\!\!*} \tau_1 \le \phi \mathrel{-\!\!*} \tau_2}$$

are valid, and adjoint in the sense that the subtyping rule

$$\frac{\phi * \psi \le \tau}{\psi \le \phi \mathrel{-\!\!*} \tau}$$

is both valid and invertible.

This proposition is actually independent of whether $\circledast$ and e satisfy the monoid axioms, and can even be adapted for a binary operation of arbitrary type $\circledast : A \bullet B \to C$.

The functoriality of $\phi * -$ expressed in Prop. 29 is a trivial instance of O'Hearn's *frame rule*, which can be expressed as follows for a general command c:

$$\frac{\psi_1 \underset{c}{\Longrightarrow} \psi_2}{\phi * \psi_1 \underset{c}{\Longrightarrow} \phi * \psi_2}\ \mathit{frame}$$

There is no reason why the frame rule should be valid in an arbitrary refinement system, and it is impossible to derive it from the above axioms. On the other hand, in the presence of pullbacks or pushforwards, it is not difficult to see that the frame rule is equivalent to either one of two simple algebraic conditions on the command c:

$$\phi * c^*\,\psi \le c^*\,(\phi * \psi) \qquad c\,(\phi * \psi) \le \phi * c\,\psi$$

The correspondence between the frame rule and the left-hand side property was noticed already by O'Hearn and Yang [24]. Here, we would like to emphasize that the left-hand side property says that pullback along c, seen as an endofunctor

$$c^* : \mathcal{D}_W \to \mathcal{D}_W$$

on the fiber of W, is strong with respect to the action $\phi * -$ of the monoidal category $\mathcal{D}_W$ on itself. This correspondence connects the frame rule to the traditional description of locality as a monadic strength in the semantics of effects.

6. Reconstructing "The Meaning of Types"

As we mentioned in the introduction, John Reynolds wrote eloquently on the distinction between the "intrinsic" and "extrinsic" views of typing (as he did on many topics), and considered the relationship between these two views carefully in his paper on "The Meaning of Types" [31]. To conclude our paper, we want to revisit Reynolds' analysis, and describe how much of it may be expressed quite naturally in the language of refinement systems. The following Sections 6.1 to 6.5 track Sections 1–5 of [31].

6.1 Syntax and Typing Rules

We define a category $\mathcal{T}$ and a functor $\mathbf{U} : \mathcal{D} \to \mathcal{T}$, representing the syntax and typing rules of a small language.

The language Reynolds considers in the paper is a simple extension of the lambda calculus including primitive boolean and arithmetic operations, records, recursion, and subtyping. Since the language is meant to be illustrative rather than interesting of itself, we will further simplify here by getting rid of records, recursion, and a few of the primitive operations, in order to focus on the treatment of subtyping. We will also follow the LF approach [10] and use higher-order abstract syntax (rather than explicit identifiers) to describe binding operations, since this leads to an elegant analysis in terms of *cartesian* logical refinement systems.

Definition 30. A **cartesian logical refinement system** is a logical refinement system $\mathbf{U} : \mathcal{D} \to \mathcal{T}$ in which the monoidal structures on $\mathcal{D}$ and $\mathcal{T}$ are cartesian [21]. We write $X \times Y$ and Y^X to denote cartesian products and exponentials.

The reason we must consider cartesian rather than arbitrary logical refinement systems is that in Reynolds' language variables can be used any number of times and in any order. Otherwise, though, whether one works with cartesian logical refinement systems or in the general monoidal setting, the structure of the analysis is essentially the same.

So, we begin by considering the category $\mathcal{T}$ of terms as a cartesian category freely generated from a single type P of *phrases*, a pair of morphisms

$$lam : P^P \to P$$
$$app : P \to P^P$$

representing lambda and application, and morphisms

$$add, sub, lt : P \times P \to P$$
$$not : P \to P$$
$$zero, one, true, false : 1 \to P$$

representing the operations of addition, subtraction, comparison, negation, and the numerical and boolean constants.

We next define a functor $\mathbf{U} : \mathcal{D} \to \mathcal{T}$, as a cartesian logical refinement system freely generated from the following refinement rules

$$\frac{}{int \sqsubset P} \qquad \frac{}{nat \sqsubset P} \qquad \frac{}{bool \sqsubset P} \qquad \frac{\theta_1 \sqsubset P \quad \theta_2 \sqsubset P}{fn[\theta_1, \theta_2] \sqsubset P}$$

the following subtyping rules

$$\frac{}{nat \le int}\ NI \qquad \frac{}{nat \le bool}\ NB \qquad \frac{\theta_1' \le \theta_1 \quad \theta_2 \le \theta_2'}{fn[\theta_1, \theta_2] \le fn[\theta_1', \theta_2']}\ Fn$$

a pair of typing rules for lambda and application

$$\frac{}{{\theta_2}^{\theta_1} \underset{lam}{\Longrightarrow} fn[\theta_1, \theta_2]}\ Lam \qquad \frac{}{fn[\theta_1, \theta_2] \underset{app}{\Longrightarrow} {\theta_2}^{\theta_1}}\ App$$

and a collection of typing rules for the primitive operations

$$\frac{}{nat \times nat \underset{add}{\Longrightarrow} nat}\ N{+} \qquad \frac{}{int \times int \underset{add}{\Longrightarrow} int}\ I{+} \qquad \frac{}{bool \times bool \underset{add}{\Longrightarrow} bool}\ B{+}$$

$$\frac{}{int \times int \underset{sub}{\Longrightarrow} int}\ I{-} \qquad \frac{}{int \times int \underset{lt}{\Longrightarrow} bool}\ I{<} \qquad \frac{}{bool \underset{not}{\Longrightarrow} bool}\ B{\neg}$$

logical framework—are a recurring pattern, and an example of the "microcosm principle" in the sense of Baez and Dolan. In Section 6.4 we will see this pattern again in the definition of the logical relation.

$$\frac{}{1 \underset{zero}{\Longrightarrow} nat}\ N0 \qquad \frac{}{1 \underset{one}{\Longrightarrow} nat}\ N1 \qquad \frac{}{1 \underset{true}{\Longrightarrow} bool}\ Bt \qquad \frac{}{1 \underset{false}{\Longrightarrow} bool}\ Bf$$

We emphasize that this signature should be read as specifying conditions on the category $\mathcal{D}$ and the functor $\mathbf{U}$. For example, the *NI* subtyping rule indicates that $\mathcal{D}$ contains a morphism $NI : nat \to int$ and that $\mathbf{U}$ maps NI to id_P, while the *App* rule says that $\mathcal{D}$ contains a morphism $App : fn[\theta_1, \theta_2] \to {\theta_2}^{\theta_1}$ (for every $\theta_1, \theta_2 \sqsubset P$) and that $\mathbf{U}$ maps *App* to *app*.

In the spirit of the LF approach (but now at the level of refinements, cf. [20, 26]), there is a lot that we *don't* have to mention in this signature. For example, we don't have to explicitly include Reynolds' subsumption rule, since it is valid in any refinement system (Prop. 7). On the other hand, observe that the *Fn* subtyping rule is *not* an instance of Prop. 15, since a priori we do not identify $fn[\theta_1, \theta_2]$ with the exponential object ${\theta_2}^{\theta_1}$ (the former refines P, the latter P^P).

Example 7. The function $dec = lam(\lambda[(\mathrm{id} \bullet one); sub])$ subtracting one from an integer may be typed by the following derivation (at the bottom we write $; Lam$ for the typing rule defined by post-composing with *Lam*):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{int \underset{\mathrm{id}}{\Longrightarrow} int}\ \mathrm{id} \quad \dfrac{\dfrac{}{1 \underset{one}{\Longrightarrow} nat}\ N1 \quad \dfrac{}{nat \leq int}\ NI}{1 \underset{one}{\Longrightarrow} int}\ ;}{int \underset{\mathrm{id} \bullet one}{\Longrightarrow} int \times int}\ \bullet \qquad \dfrac{}{int \times int \underset{sub}{\Longrightarrow} int}\ I-}{int \underset{(\mathrm{id} \bullet one); sub}{\Longrightarrow} int}\ ;}{1 \underset{\lambda[(\mathrm{id} \bullet one); sub]}{\Longrightarrow} int^{int}}\ \lambda}{1 \underset{dec}{\Longrightarrow} fn[int, int]}\ ; Lam$$

Example 8. It is possible to have more than one typing derivation of the same typing judgment. For example, the typing judgment

$$nat \times nat \underset{add}{\Longrightarrow} bool$$

may be derived either as

$$\dfrac{\dfrac{}{nat \times nat \underset{add}{\Longrightarrow} nat}\ N+ \quad \dfrac{}{nat \leq bool}\ NB}{nat \times nat \underset{add}{\Longrightarrow} bool}\ ;$$

or as

$$\dfrac{\dfrac{\dfrac{}{nat \leq bool}\ NB \quad \dfrac{}{nat \leq bool}\ NB}{nat \times nat \leq bool \times bool}\ \bullet \qquad \dfrac{}{bool \times bool \underset{add}{\Longrightarrow} bool}\ B+}{nat \times nat \underset{add}{\Longrightarrow} bool}\ ;$$

6.2 An Intrinsic Semantics

We define a cartesian closed functor $[\![-]\!]_D : \mathcal{D} \to \mathbf{Dom}$ from derivations to domains.

In order to define the intrinsic semantics we assume domains $\mathbb{N}_\perp, \mathbb{Z}_\perp, \mathbb{B}_\perp$ of naturals, integers, and booleans, as well as operations

$$+_{\mathbb{N}} : \mathbb{N}_\perp \times \mathbb{N}_\perp \to \mathbb{N}_\perp \quad +_{\mathbb{Z}} : \mathbb{Z}_\perp \times \mathbb{Z}_\perp \to \mathbb{Z}_\perp \quad \vee_{\mathbb{B}} : \mathbb{B}_\perp \times \mathbb{B}_\perp \to \mathbb{B}_\perp$$
$$-_{\mathbb{Z}} : \mathbb{Z}_\perp \times \mathbb{Z}_\perp \to \mathbb{Z}_\perp \quad <_{\mathbb{Z}} : \mathbb{Z}_\perp \times \mathbb{Z}_\perp \to \mathbb{B}_\perp \quad \neg_{\mathbb{B}} : \mathbb{B}_\perp \to \mathbb{B}_\perp$$
$$0, 1 : 1 \to \mathbb{N}_\perp \quad tt, ff : 1 \to \mathbb{B}_\perp$$

standing for the *componentwise-strict* extensions of the evident operations on naturals, integers, and booleans. We also write $i : \mathbb{N}_\perp \to \mathbb{Z}_\perp$ for the evident inclusion, and $j : \mathbb{N}_\perp \to \mathbb{B}_\perp$ for the strict extension of the following map:

$$n \mapsto \begin{cases} tt & \text{if } n > 0 \\ ff & \text{if } n = 0 \end{cases}$$

Then we define the functor $[\![-]\!]_D : \mathcal{D} \to \mathbf{Dom}$ by:

$$[\![nat]\!] = \mathbb{N}_\perp \quad [\![int]\!] = \mathbb{Z}_\perp \quad [\![bool]\!] = \mathbb{B}_\perp \quad [\![fn[S,T]]\!] = [\![T]\!]^{[\![S]\!]}$$
$$[\![NI]\!] = i \quad [\![NB]\!] = j \quad [\![Fn[\alpha,\beta]]\!] = (f \mapsto [\![\alpha]\!]; f; [\![\beta]\!])$$
$$[\![Lam]\!] = \mathrm{id} \quad [\![App]\!] = \mathrm{id}$$
$$[\![N+]\!] = +_{\mathbb{N}} \quad [\![I+]\!] = +_{\mathbb{Z}} \quad [\![B+]\!] = \vee_{\mathbb{B}}$$
$$[\![I-]\!] = -_{\mathbb{Z}} \quad [\![I<]\!] = <_{\mathbb{Z}} \quad [\![B\neg]\!] = \neg_{\mathbb{B}}$$
$$[\![N0]\!] = 0 \quad [\![N1]\!] = 1 \quad [\![Bt]\!] = tt \quad [\![Bf]\!] = ff$$

Note that this data suffices for defining the cartesian closed functor $[\![-]\!]_D$, by the requirement that it preserves cartesian products and exponentials.

6.3 An Untyped Semantics

We define a cartesian closed functor $[\![-]\!]_T : \mathcal{T} \to \mathbf{Dom}$ from terms to domains.

In order to define the untyped semantics we assume a domain U equipped with operations

$$\mathbb{Z}_\perp \underset{\Psi_{\mathrm{p}}}{\overset{\Phi_{\mathrm{p}}}{\rightleftarrows}} U \qquad U^U \underset{\Psi_{\mathrm{f}}}{\overset{\Phi_{\mathrm{f}}}{\rightleftarrows}} U$$

such that $\Phi_{\mathrm{p}}; \Psi_{\mathrm{p}} = \mathrm{id}$ and $\Phi_{\mathrm{f}}; \Psi_{\mathrm{f}} = \mathrm{id}$ (i.e., embedding-retraction pairs). We also suppose operations

$$tt_{\mathbb{Z}} : 1 \to \mathbb{Z}_\perp \quad <_{\mathbb{Z}} : \mathbb{Z}_\perp \times \mathbb{Z}_\perp \to \mathbb{Z}_\perp \quad \neg_{\mathbb{Z}} : \mathbb{Z}_\perp \to \mathbb{Z}_\perp$$

which must at minimum satisfy

$$\begin{aligned} tt_{\mathbb{Z}} &> 0 \\ (x <_{\mathbb{Z}} y) &> 0 \quad \text{when } x < y \\ (x <_{\mathbb{Z}} y) &= 0 \quad \text{when } x \geq y \\ (\neg_{\mathbb{Z}} x) &> 0 \quad \text{when } x = 0 \\ (\neg_{\mathbb{Z}} x) &= 0 \quad \text{when } x > 0 \end{aligned}$$

Then $[\![-]\!]_T$ is defined by:

$$[\![P]\!] = U \qquad [\![lam]\!] = \Phi_{\mathrm{f}} \quad [\![app]\!] = \Psi_{\mathrm{f}}$$
$$[\![add]\!] = (\Psi_{\mathrm{p}} \bullet \Psi_{\mathrm{p}}); +_{\mathbb{Z}}; \Phi_{\mathrm{p}} \quad [\![sub]\!] = (\Psi_{\mathrm{p}} \bullet \Psi_{\mathrm{p}}); -_{\mathbb{Z}}; \Phi_{\mathrm{p}}$$
$$[\![lt]\!] = (\Psi_{\mathrm{p}} \bullet \Psi_{\mathrm{p}}); <_{\mathbb{Z}}; \Phi_{\mathrm{p}} \quad [\![not]\!] = (\Psi_{\mathrm{p}} \bullet \Psi_{\mathrm{p}}); \neg_{\mathbb{Z}}; \Phi_{\mathrm{p}}$$
$$[\![zero]\!] = [\![false]\!] = 0; i; \Phi_{\mathrm{p}} \quad [\![one]\!] = 1; i; \Phi_{\mathrm{p}} \quad [\![true]\!] = tt_{\mathbb{Z}}; \Phi_{\mathrm{p}}$$

As in Section 6.2, this data suffices for defining $[\![-]\!]_T$ as a cartesian closed functor.

6.4 Logical Relations

We formalize Reynolds' logical relations theorem, connecting the intrinsic semantics to the untyped semantics.

The logical relations theorem boils down to defining a cartesian closed functor $\rho : \mathcal{D} \to \mathbf{DRel}$ satisfying a pair of commuting squares:

$$\begin{array}{ccc} \mathcal{D} & \xrightarrow{\rho} & \mathbf{DRel} \\ \| & (0) & \downarrow{\scriptstyle \partial_0} \\ \mathcal{D} & \xrightarrow[[\![-]\!]_D]{} & \mathbf{Dom} \end{array} \qquad \begin{array}{ccc} \mathcal{D} & \xrightarrow{\rho} & \mathbf{DRel} \\ \mathbf{U} \downarrow & (1) & \downarrow{\scriptstyle \partial_1} \\ \mathcal{T} & \xrightarrow[[\![-]\!]_T]{} & \mathbf{Dom} \end{array}$$

Here **DRel** is the category whose objects are chain-complete relations over domains (hence *domain-theoretic relations*) and whose morphisms are pairs of functions mapping related arguments to related results, while the ∂_i are the evident projection functors. To define ρ, we take advantage of the fact that the pairing of the projection functors $(\partial_0, \partial_1) :$ **DRel** $\to$

Dom × **Dom** has the structure of a logical refinement system (with all pullbacks and some pushforwards)—so let us take a moment to describe this structure (cf. [14, Example 3(2)]).

A refinement $S \sqsubset (A, B)$ corresponds to a chain-complete relation $S \subseteq A \times B$ between domains, while a derivation of

$$S \underset{(f,g)}{\Longrightarrow} T$$

corresponds to a proof that

$$\forall a, b.\ a \sim_S b \Rightarrow f(a) \sim_T g(b)$$

Products, residuals, pullbacks and pushforwards are defined as follows on relations:

$$\begin{aligned}
(a_1, a_2) \sim_{S\times T} (b_1, b_2) \quad &\text{iff} \quad a_1 \sim_S b_1 \wedge a_2 \sim_T b_2 \\
f \sim_{T^S} g \quad &\text{iff} \quad \forall a, b.\ a \sim_S b \Rightarrow f(a) \sim_T g(b) \\
a \sim_{(f,g)^* T} b \quad &\text{iff} \quad f(a) \sim_T g(b) \\
c \sim_{(f,g)\, S} d \quad &\text{iff} \quad \exists a, b.\ c = f(a) \wedge d = g(b) \wedge a \sim_S b
\end{aligned}$$

Note that pushforward is a partial operation, because the relation $(f, g)\, S$ is not necessarily chain-complete. However, the pushforward along a pair of functions (f, g) with flat codomains (or more generally, with codomains where every element is compact) is always defined.

Now, in order to satisfy the squares (0) and (1), the object part of the functor $\rho : \mathcal{D} \to$ **DRel** must assign to each $\theta \sqsubset P$ a relation $\rho[\theta] \sqsubset (\llbracket\theta\rrbracket, U)$. Let Δ : **Dom** $\to$ **DRel** be the (cartesian closed) functor assigning the identity relation to any domain. We define $\rho[\theta]$ as follows (by induction on θ):

$$\begin{aligned}
\rho[int] &= (\mathrm{id}, \Psi_\mathrm{p})^*\, \Delta[\mathbb{Z}_\perp] \\
\rho[nat] &= (\mathrm{id}, \Psi_\mathrm{p})^*\, (\mathrm{id}, i)\, \Delta[\mathbb{N}_\perp] \\
\rho[bool] &= (\mathrm{id}, \Psi_\mathrm{p})^*\, (\mathrm{id}, i)\, (\mathrm{id}, j)^*\, \Delta[\mathbb{B}_\perp] \\
\rho[fn[\theta_1, \theta_2]] &= (\mathrm{id}, \Psi_\mathrm{f})^*\, \rho[\theta_2]^{\rho[\theta_2]}
\end{aligned}$$

By unwinding the interpretation of the logical refinement system **DRel** $\to$ **Dom** × **Dom**, it is easy to check that these definitions agree with Reynolds' Definitions 4.1 and 4.2. For example, we have

$$\begin{aligned}
b \sim_{\rho[bool]} p \quad &\text{iff} \quad b \sim_{(\mathrm{id},i)\,(\mathrm{id},j)^*\,\Delta[\mathbb{B}]} \Psi_\mathrm{p}(p) \\
&\text{iff} \quad \exists n.\, \Psi_\mathrm{p}(p) = i(n) \wedge b \sim_{(\mathrm{id},j)^*\,\Delta[\mathbb{B}]} n \\
&\text{iff} \quad \exists n.\, \Psi_\mathrm{p}(p) = i(n) \wedge b = j(n)
\end{aligned}$$

We can now prove that ρ extends to a functor and that the two squares (0) and (1) commute, which modulo our treatment of variables (using higher-order abstract syntax rather than explicit environments) is an exact transcription of Reynolds' Logical Relations Theorem (4.8).

Theorem 31. *If* $\theta_1 \underset{p}{\overset{\alpha}{\Longrightarrow}} \theta_2$ *then* $\vdash \rho[\theta_1] \underset{(\llbracket\alpha\rrbracket,\llbracket p\rrbracket)}{\Longrightarrow} \rho[\theta_2]$.

Proof. By induction on α. The content of the proof is basically identical with the proof in [31], but the trip through refinement systems gives the proof considerably more structure. We illustrate with a few cases:

- $\alpha = NI$:

$$\dfrac{\dfrac{\dfrac{}{\Delta[\mathbb{N}_\perp] \underset{(i,i)}{\Longrightarrow} \Delta[\mathbb{Z}_\perp]}\ \Delta[i]}{(\mathrm{id}, i)\, \Delta[\mathbb{N}_\perp] \underset{(i,\mathrm{id})}{\Longrightarrow} \Delta[\mathbb{Z}_\perp]}\ L(\mathrm{id}, i)}{(\mathrm{id}, \Psi_\mathrm{p})^*\, (\mathrm{id}, i)\, \Delta[\mathbb{N}_\perp] \underset{(i,\mathrm{id})}{\Longrightarrow} (\mathrm{id}, \Psi_\mathrm{p})^*\, \Delta[\mathbb{Z}_\perp]}\ (fun)$$
$$\overline{\qquad\qquad}$$
$$\rho[nat] \underset{(\llbracket NI\rrbracket,\llbracket \mathrm{id}\rrbracket)}{\Longrightarrow} \rho[int]$$

  where we write a double line for expansion of definitions, (*fun*) for functoriality of pullbacks (Prop. 24), and $\Delta[i]$ for the application of the functor Δ to the map $i : \mathbb{N}_\perp \to \mathbb{Z}_\perp$.

- $\alpha = Lam$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\rho[\theta_2]^{\rho[\theta_1]} \underset{(\mathrm{id},\mathrm{id})}{\Longrightarrow} \rho[\theta_2]^{\rho[\theta_1]}}\ \mathrm{id}}{\rho[\theta_2]^{\rho[\theta_1]} \underset{(\mathrm{id},(\Phi_\mathrm{f};\Psi_\mathrm{f}))}{\Longrightarrow} \rho[\theta_2]^{\rho[\theta_1]}}\ \sim}{\rho[\theta_2]^{\rho[\theta_1]} \underset{(\mathrm{id},\Phi_\mathrm{f})}{\Longrightarrow} (\mathrm{id}, \Psi_\mathrm{f})^*\, \rho[\theta_2]^{\rho[\theta_1]}}\ R(\mathrm{id}, \Psi_\mathrm{f})^*}{\rho[\theta_2{}^{\theta_1}] \underset{(\llbracket Lam\rrbracket,\llbracket lam\rrbracket)}{\Longrightarrow} \rho[fn[\theta_1, \theta_2]]}$$

- $\alpha = N+$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta[\mathbb{N}_\perp] \bullet \Delta[\mathbb{N}_\perp] \underset{(+_\mathbb{N},+_\mathbb{N})}{\Longrightarrow} \Delta[\mathbb{N}_\perp]}\ \Delta[+_\mathbb{N}] \quad \dfrac{}{\Delta[\mathbb{N}_\perp] \underset{(\mathrm{id},i)}{\Longrightarrow} (\mathrm{id}, i)\, \Delta[\mathbb{N}_\perp]}\ R(\mathrm{id}, i)}{\Delta[\mathbb{N}_\perp] \bullet \Delta[\mathbb{N}_\perp] \underset{(+_\mathbb{N},(+_\mathbb{N};i))}{\Longrightarrow} (\mathrm{id}, i)\, \Delta[\mathbb{N}_\perp]}\ ;}{\Delta[\mathbb{N}_\perp] \times \Delta[\mathbb{N}_\perp] \underset{(+_\mathbb{N},(i\bullet i;+_\mathbb{Z}))}{\Longrightarrow} (\mathrm{id}, i)\, \Delta[\mathbb{N}_\perp]}\ \sim}{\rho[nat] \times \rho[nat] \underset{(+_\mathbb{N},((\Psi_\mathrm{P}\bullet\Psi_\mathrm{P});+_\mathbb{Z},\Phi_\mathrm{P}))}{\Longrightarrow} \rho[nat]}\ (\dagger)$$
$$\overline{\qquad\qquad}$$
$$\rho[nat \times nat] \underset{(\llbracket N+\rrbracket,\llbracket add\rrbracket)}{\Longrightarrow} \rho[nat]$$

  where at (†) we have elided several easy steps of reasoning, and at ∼ we use the fact that $(+_\mathbb{N}; i) = (i \bullet i; +_\mathbb{Z})$.

□

6.5 Bracketing

We formalize Reynolds' bracketing theorem, which combined with the logical relations theorem yields coherence.

We begin by defining for each domain $\llbracket\theta\rrbracket$ in the image of the intrinsic semantics a pair of functions

$$\llbracket\theta\rrbracket \underset{\psi[\theta]}{\overset{\phi[\theta]}{\rightleftarrows}} U$$

which will turn out to be an embedding-retraction pair. The family is defined as follows (by induction on θ):

$$\phi[int] = \Phi_\mathrm{p} \quad \psi[int] = \Psi_\mathrm{p} \quad \phi[nat] = i; \Phi_\mathrm{p} \quad \psi[nat] = \Psi_\mathrm{p}; i'$$

$$\phi[bool] = j'; i; \Phi_\mathrm{p} \quad \psi[bool] = \Psi_\mathrm{p}; i'; j$$

$$\phi[fn[\theta_1, \theta_2]] = \phi[\theta_2]^{\psi[\theta_1]}; \Phi_\mathrm{f} \quad \psi[fn[\theta_1, \theta_2]] = \Psi_\mathrm{f}; \psi[\theta_2]^{\phi[\theta_1]}$$

Here $g^f \overset{\mathrm{def}}{=} \lambda[f \bullet \mathrm{id}; \otimes; g]$, while $i' : \mathbb{Z}_\perp \to \mathbb{N}_\perp$ is the function sending non-negative integers to naturals and everything else to $\perp$, and $j' : \mathbb{B}_\perp \to \mathbb{N}_\perp$ is the strict extension of the function sending *tt* to 1 and *ff* to 0. Note that this family of pairs may be seen as a pair of transformations between functors

$$\phi : \llbracket - \rrbracket_D \Rightarrow (\mathbf{U}; \llbracket - \rrbracket_T) \qquad \psi : (\mathbf{U}; \llbracket - \rrbracket_T) \Rightarrow \llbracket - \rrbracket_D$$

albeit not a pair of *natural* transformations. Instead, ϕ and ψ are related by Reynolds' *bracketing theorem*.

Theorem 32 (Bracketing)**.** *The two judgments*

$$\Delta\llbracket\theta\rrbracket \underset{(\mathrm{id},\phi_\theta)}{\Longrightarrow} \rho[\theta] \underset{(\mathrm{id},\psi_\theta)}{\Longrightarrow} \Delta\llbracket\theta\rrbracket$$

are derivable for all θ.

Proof. By induction on θ. Once again, the proof is highly structured, and we include it in full in Appendix A. □

Finally, by combining the bracketing theorem with the logical relations theorem, we can show that the intrinsic semantics

is coherent, i.e., that any two derivations of the same judgment have the same interpretation (the following statements correspond to Reynolds' Theorem 5.7 and Corollary 5.8).

Corollary 33 (Coherence)**.** *We have:*

1. *If* $\theta_1 \overset{\alpha}{\underset{p}{\Longrightarrow}} \theta_2$ *then* $[\![\alpha]\!] = \phi_{\theta_1}; [\![p]\!]; \psi_{\theta_2}$.
2. *If* $\theta_1 \overset{\alpha_1}{\underset{p}{\Longrightarrow}} \theta_2$ *and* $\theta_1 \overset{\alpha_2}{\underset{p}{\Longrightarrow}} \theta_2$ *then* $[\![\alpha_1]\!] = [\![\alpha_2]\!]$.

7. Related work and conclusions

Refinement type systems are the higher-order version of Hoare logic, and as such, they are recognized today as a fundamental tool in program analysis and certification. An important contribution to this line of research has been the work by Frank Pfenning and his collaborators [9] who have developed along the years a comprehensive theory of refinement type systems, including a clean account of the relationship between extrinsic and intrinsic typing [27]. Here, by going back to Reynolds [31] we establish a very natural connection between refinement type systems and functorial semantics, based on the idea that every functor defines a refinement type system.

Functorial semantics is an old idea going back to Lawvere in algebra [18] and logic [19], and which plays a central role in the study of imperative languages since Reynolds and Oles [25, 28]. Functorial semantics has also played a defining role in the early development of separation logic [23] as well as in more recent extensions of the logic to higher-order imperative languages [3, 4]. One distinctive feature of the present work is to develop a formal language of typing judgments and derivations reflecting the basic reasoning principles of functorial semantics. This language has been designed in order to be amenable to mechanization and could eventually serve as an intermediate language in a proof assistant. We demonstrated the power of the language in Section 6, by recasting the sophisticated semantic arguments used by Reynolds [31] in a concise and highly structured way.

One of the original motivations for this language was to better understand effect type systems and their fibrational aspects, along the lines of [8, 14]. The idea of using product and implication-preserving fibrations in the study of logical predicates and logical relations may be traced back to Hermida [11], with later developments by Katsumata. Investigating effects and refinement type systems led us to replace fibrations by general functors, and in particular to appreciate the expressive power of closed functors. Together with the existence of *specific* pullbacks and pushforwards, one recovers many of the operations of dependent types but in a more flexible and general setting.

The principle of refining types while paying careful attention to the dual act of "forgetting" also appears in McBride's notion of *ornament* [22], which have been analyzed in fibrational terms [1, 6]. We would like to clarify the connection with our work in the future. Finally, the idea of using closed functors as a logical framework (capable of speaking about both "syntax" and "semantics" in a unified way) is very much in the spirit of de Groote's *abstract categorial grammars* [7], as well as Carette, Kiselyov and Shan's *tagless interpreters* [5].

References

[1] Robert Atkey, Patricia Johann, and Neil Ghani. Refining Inductive Types. *LMCS*, 8:2, 2012.

[2] Jean Bénabou. Distributors at work. Notes from a course at TU Darmstadt in June 2000, taken by Thomas Streicher.

[3] Bodil Biering, Lars Birkedal, and Noah Torp-Smith. BI-Hyperdoctrines, Higher-order Separation Logic, and Abstraction. *ACM Trans. Program. Lang. Syst.*, 5:29, 2007.

[4] Lars Birkedal, Noah Torp-Smith, and Hongseok Yang. Semantics of Separation-Logic Typing and Higher-order Frame Rules for Algol-like languages. *LMCS*, 5:2, 2006.

[5] Jacques Carette, Oleg Kiselyov, and Chung-chieh Shan. Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages. *JFP*, 5:19, 2009.

[6] Pierre-Evariste Dagand and Conor McBride. A Categorical Treatment of Ornaments. LICS 2013.

[7] Philippe de Groote. Towards Abstract Categorial Grammars. In *Assoc. for Computational Linguistics, 39th Annual Meeting*, 2001.

[8] Andrzej Filinski. Monads in Action. POPL 2010.

[9] Tim Freeman and Frank Pfenning. Refinement Types for ML. PLDI 1991.

[10] Robert Harper, Furio Honsell and Gordon Plotkin. A Framework For Defining Logics. *Journal of the ACM*, 40(1):143-184, 1993.

[11] Claudio Hermida. *Fibrations, Logical predicates and indeterminates*, PhD thesis, University of Edinburgh, November 1993.

[12] C.A.R. Hoare. An Axiomatic Basis for Computer Programming, *Communications of the ACM*, 12:10, 1969.

[13] Bart Jacobs. *Categorical Logic and Type Theory.* Studies in Logic and the Foundations of Mathematics 141. North Holland, 1999.

[14] Shin-ya Katsumata. Relating Computational Effects by ⊤⊤-Lifting. ICALP 2011.

[15] Max Kelly. *Basic concepts in enriched category theory*. CUP, 1982.

[16] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:3, 1958.

[17] Joachim Lambek and Philip Scott. *Introduction to Higher-order Categorical Logic*. CUP, 1986.

[18] F. William Lawvere. *Functorial Semantics of Algebraic Theories*, PhD thesis, Columbia University, 1963.

[19] F. William Lawvere. Adjointness in Foundations, *Dialectica* 23, 1969, 281–296.

[20] William Lovas. *Refinement types for logical frameworks*, PhD thesis, Carnegie Mellon University, September 2010.

[21] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1971.

[22] Conor McBride. Ornamental Algebras, Algebraic Ornaments. *JFP* (to appear). 9/8/2010 version available on author's website.

[23] Peter W. O'Hearn and David J. Pym. The Logic of Bunched Implications. *BSL* 5:2, 1999.

[24] Peter W. O'Hearn and Hongseok Yang. A Semantic Basis for Local Reasoning. FOSSACS 2002.

[25] Frank J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages*, PhD thesis, Syracuse University, 1982.

[26] Frank Pfenning. Refinement Types for Logical Frameworks. Workshop on Types for Proofs and Programs, May 1993.

[27] Frank Pfenning. Church and Curry: Combining Intrinsic and Extrinsic Typing. Studies in Logic 17, 2008, 303–338.

[28] John C. Reynolds. The Essence of Algol. *Algorithmic Languages*, 1981, 345–372.

[29] John C. Reynolds. The Coherence of Languages with Intersection Types, TACS 1991.

[30] John C. Reynolds. *Theories of Programming Languages*. CUP, 1998.

[31] John C. Reynolds. The Meaning of Types: from Intrinsic to Extrinsic Semantics. BRICS Report RS-00-32, Aarhus University, December 2000.

[32] John C. Reynolds. Separation logic: A Logic for Shared Mutable Data Structures. LICS 2002.

A. Proof of the Bracketing Theorem

Recall that the bracketing theorem (Thm. 32) says that the pair of judgments

$$\Delta[\![\theta]\!] \underset{(\mathrm{id},\phi[\theta])}{\Longrightarrow} \rho[\theta] \underset{(\mathrm{id},\psi[\theta])}{\Longrightarrow} \Delta[\![\theta]\!]$$

are derivable in the logical refinement system $\mathbf{DRel} \to \mathbf{Dom} \times \mathbf{Dom}$. We prove this by induction on θ.

- (Case $\theta = fn[\theta_1, \theta_2]$).
  In one direction we build the following derivation:

$$\dfrac{\dfrac{\rho[\theta_1] \underset{(\mathrm{id},\psi[\theta_1])}{\overset{i.h.}{\Longrightarrow}} \Delta[\![\theta_1]\!] \quad \dfrac{}{\Delta[\![\theta_1]\!] \bullet \Delta[\![\theta_2]\!]^{\Delta[\![\theta_1]\!]} \underset{(\otimes,\otimes)}{\Longrightarrow} \Delta[\![\theta_2]\!]}\,\Delta[\otimes] \quad \Delta[\![\theta_2]\!] \underset{(\mathrm{id},\phi[\theta_2])}{\overset{i.h.}{\Longrightarrow}} \rho[\theta_2]}{\dfrac{\rho[\theta_1] \bullet \Delta[\![\theta_2]\!]^{\Delta[\![\theta_1]\!]} \underset{(\otimes,(\psi[\theta_1]\bullet\mathrm{id};\otimes;\phi[\theta_2]))}{\Longrightarrow} \rho[\theta_2]}{\Delta[\![\theta_2]\!]^{\Delta[\![\theta_1]\!]} \underset{(\mathrm{id},\phi[\theta_2]^{\psi[\theta_1]})}{\Longrightarrow} \rho[\theta_2]^{\rho[\theta_1]}}\,\lambda}{\dfrac{\Delta[\![\theta_2]\!]^{\Delta[\![\theta_1]\!]} \underset{(\mathrm{id},(\phi[\theta_2]^{\psi[\theta_1]};\Phi_f;\Psi_f))}{\Longrightarrow} \rho[\theta_2]^{\rho[\theta_1]}}{\Delta[\![\theta_2]\!]^{\Delta[\![\theta_1]\!]} \underset{(\mathrm{id},(\phi[\theta_2]^{\psi[\theta_1]};\Phi_f))}{\Longrightarrow} (\mathrm{id},\Psi_f)^*\,\rho[\theta_2]^{\rho[\theta_1]}}\,R(\mathrm{id},\Psi_f)^*}{\Delta[\![fn[\theta_1,\theta_2]]\!] \underset{(\mathrm{id},\phi[fn[\theta_1,\theta_2]])}{\Longrightarrow} \rho[fn[\theta_1,\theta_2]]}\,\sim}\,(\dagger)$$

  where at (†) we apply composition twice (the order is irrelevant by associativity), after multiplying the leftmost premise by the identity on the right to bring it to the appropriate type.
  In the other direction we build the following derivation:

$$\dfrac{\dfrac{}{(\mathrm{id},\Psi_f)^*\,\rho[\theta_2]^{\rho[\theta_1]} \underset{(\mathrm{id},\Psi_f)}{\Longrightarrow} \rho[\theta_2]^{\rho[\theta_1]}}\,L(\mathrm{id},\Psi_f)^* \quad \dfrac{\Delta[\![\theta_1]\!] \underset{(id,\phi[\theta_1])}{\overset{i.h.}{\Longrightarrow}} \rho[\theta_1] \quad \rho[\theta_2] \underset{(id,\psi[\theta_2])}{\overset{i.h.}{\Longrightarrow}} \Delta[\![\theta_1]\!]}{\Delta[\![\theta_1]\!] \bullet \rho[\theta_2]^{\rho[\theta_1]} \underset{(\otimes,(\phi[\theta_1]\bullet\mathrm{id};\otimes;\psi[\theta_2]))}{\Longrightarrow} \Delta[\![\theta_2]\!]}\,L\backslash}{\dfrac{\Delta[\![\theta_1]\!] \bullet (\mathrm{id},\Psi_f)^*\,\rho[\theta_2]^{\rho[\theta_1]} \underset{(\otimes,(\phi[\theta_1]\bullet\Psi_f;\otimes;\psi[\theta_2]))}{\Longrightarrow} \Delta[\![\theta_2]\!]}{(\mathrm{id},\Psi_f)^*\,\rho[\theta_2]^{\rho[\theta_1]} \underset{(\mathrm{id},(\Psi_f;\psi[\theta_2]^{\phi[\theta_1]}))}{\Longrightarrow} \Delta[\![\theta_2]\!]^{\Delta[\![\theta_1]\!]}}\,\lambda}{\rho[fn[\theta_1,\theta_2]] \underset{(\mathrm{id},\psi[fn[\theta_1,\theta_2]])}{\Longrightarrow} \Delta[\![fn[\theta_1,\theta_2]]\!]}\,(\dagger)$$

  where at (†) again we multiply the leftmost premise by the identity on the left before composing, and at $L\backslash$ we apply an instance of the following rule (which is valid in any logical refinement system):

$$\dfrac{S' \underset{v}{\Longrightarrow} S \quad U \underset{k}{\Longrightarrow} U'}{S' \bullet S \setminus U \underset{v\bullet\mathrm{id};\otimes;k}{\Longrightarrow} U'}\,L\backslash$$

- (Case $\theta = int$).

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta[\mathbb{Z}_\perp] \underset{(\mathrm{id},\mathrm{id})}{\Longrightarrow} \Delta[\mathbb{Z}_\perp]}\,\mathrm{id}}{\Delta[\mathbb{Z}_\perp] \underset{(\mathrm{id},(\Phi_p;\Psi_p))}{\Longrightarrow} \Delta[\mathbb{Z}_\perp]}\,\sim}{\Delta[\mathbb{Z}_\perp] \underset{(\mathrm{id},\Phi_p)}{\Longrightarrow} (\mathrm{id},\Psi_p)^*\,\Delta[\mathbb{Z}_\perp]}\,R(\mathrm{id},\Psi_p)^*}{\Delta[\![int]\!] \underset{(\mathrm{id},\phi[int])}{\Longrightarrow} \rho[int]} \qquad \dfrac{}{(\mathrm{id},\Psi_p)^*\,\Delta[\mathbb{Z}_\perp] \underset{(\mathrm{id},\Psi_p)}{\Longrightarrow} \Delta[\mathbb{Z}_\perp]}\,L(\mathrm{id},\Psi_p)^*$$

- (Case $\theta = nat$).

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta[\mathbb{N}_\perp] \underset{(\mathrm{id},i)}{\Longrightarrow} (\mathrm{id},i)\,\Delta[\mathbb{N}_\perp]}\,R(\mathrm{id},i)}{\Delta[\mathbb{N}_\perp] \underset{(\mathrm{id},(i;\Phi_\mathrm{p};\Psi_\mathrm{p}))}{\Longrightarrow} (\mathrm{id},i)\,\Delta[\mathbb{N}_\perp]}\sim}{\Delta[\mathbb{N}_\perp] \underset{(\mathrm{id},(i;\Phi_\mathrm{p}))}{\Longrightarrow} (\mathrm{id},\Psi_\mathrm{p})^*\,(\mathrm{id},i)\,\Delta[\mathbb{N}_\perp]}\,R(\mathrm{id},\Psi_\mathrm{p})^*}{\Delta[\![nat]\!] \underset{(\mathrm{id},\phi[nat])}{\Longrightarrow} \rho[nat]}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta[\mathbb{N}_\perp] \underset{(\mathrm{id},\mathrm{id})}{\Longrightarrow} \Delta[\mathbb{N}_\perp]}\,\mathrm{id}}{\Delta[\mathbb{N}_\perp] \underset{(\mathrm{id},(i;i'))}{\Longrightarrow} \Delta[\mathbb{N}_\perp]}\sim}{(\mathrm{id},i)\,\Delta[\mathbb{N}_\perp] \underset{(\mathrm{id},i')}{\Longrightarrow} \Delta[\mathbb{N}_\perp]}\,L(\mathrm{id},i)}{(\mathrm{id},\Psi_\mathrm{p})^*\,(\mathrm{id},i)\,\Delta[\mathbb{N}_\perp] \underset{(\mathrm{id},(\Psi_\mathrm{p};i'))}{\Longrightarrow} \Delta[\mathbb{N}_\perp]}\,L(\mathrm{id},\Psi_\mathrm{p})^*\,;$$
$$\rho[nat] \underset{(\mathrm{id},\psi[nat])}{\Longrightarrow} \Delta[\![nat]\!]$$

- (Case $\theta = bool$).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},\mathrm{id})}{\Longrightarrow} \Delta[\mathbb{B}_\perp]}\,\mathrm{id}}{\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},(j';j))}{\Longrightarrow} \Delta[\mathbb{B}_\perp]}\sim}{\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},j')}{\Longrightarrow} (\mathrm{id},j)^*\,\Delta[\mathbb{B}_\perp]}\,R(\mathrm{id},j)^*}{\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},(j';i))}{\Longrightarrow} (\mathrm{id},i)\,(\mathrm{id},j)^*\,\Delta[\mathbb{B}_\perp]}\,;R(\mathrm{id},i)}{\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},(j';i;\Phi_\mathrm{p};\Psi_\mathrm{p}))}{\Longrightarrow} (\mathrm{id},i)\,(\mathrm{id},j)^*\,\Delta[\mathbb{B}_\perp]}\sim}{\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},(j';i;\Phi_\mathrm{p}))}{\Longrightarrow} (\mathrm{id},\Psi_\mathrm{p})^*\,(\mathrm{id},i)\,(\mathrm{id},j)^*\,\Delta[\mathbb{B}_\perp]}\,R(\mathrm{id},\Psi_\mathrm{p})^*}{\Delta[\![bool]\!] \underset{(\mathrm{id},\phi[bool])}{\Longrightarrow} \rho[bool]}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\mathrm{id},j)^*\,\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},j)}{\Longrightarrow} \Delta[\mathbb{B}_\perp]}\,L(\mathrm{id},j)^*}{(\mathrm{id},j)^*\,\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},(i;i';j))}{\Longrightarrow} \Delta[\mathbb{B}_\perp]}\sim}{(\mathrm{id},i)\,(\mathrm{id},j)^*\,\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},(i';j))}{\Longrightarrow} \Delta[\mathbb{B}_\perp]}\,L(\mathrm{id},i)}{(\mathrm{id},\Psi_\mathrm{p})^*\,(\mathrm{id},i)\,(\mathrm{id},j)^*\,\Delta[\mathbb{B}_\perp] \underset{(\mathrm{id},(\Psi_\mathrm{p};i';j))}{\Longrightarrow} \Delta[\mathbb{B}_\perp]}\,L(\mathrm{id},\Psi_\mathrm{p})^*\,;$$
$$\rho[bool] \underset{(\mathrm{id},\psi[bool])}{\Longrightarrow} \Delta[\![bool]\!]$$