# Theory Presentation Combinators

Jacques Carette
and
Russell O'Connor
Department of Computing and Software, McMaster University
Hamilton, Ontario, Canada
carette@mcmaster.ca,roconnor@theorem.ca.

---

To build a scalable library of mathematics, we need a method which takes advantage of the inherent structure of mathematical theories. Here we argue that *theory presentation combinators* are a helpful tool towards that quest. We motivate our choice of combinators, and give them precise semantics. We observe that the *category of contexts* plays a fundamental rôle (explicitly or otherwise) in all such developments, so we will examine its structure carefully. In particular, as it is a fibered category, cartesian liftings are pervasive. While our original work was based on experience and intuition, this work is firmly grounded in categorical semantics, and has resulted in a much cleaner and more powerful set of theory presentation combinators.

---

## 1. INTRODUCTION

A mechanized mathematics system, to be useful, must possess a large library of mathematical knowledge, on top of sound foundations. While sound foundations contain many interesting intellectual challenges, building a large library seems a daunting task simply because of its sheer volume. However, as has been well-documented [CFJ+11, CK11, GS10], there is a tremendous amount of redundancy in existing libraries. Thus there is some hope that by designing the "right" meta-language, guided by parsimony principles [Vel07], we can reduce the effort needed to build a library of mathematics.

Our aim is to build tools that allow library developers to take advantage of commonalities in mathematics so as to build a large, rich library for end-users, whilst expending much less actual development effort than in the past. In other words, we continue with our approach of developing *High Level Theories* [CF08] through building a network of theories, by putting our previous experiments [CFJ+11] on a sound theoretical basis.

### 1.1 The Problem

The problem we wish to solve is easy to state: we want to shorten the development time of large mathematical libraries. But why would mathematical libraries be any different than other software, where the quest for time-saving techniques has been long but vain [Bro95]? Because we have known since Whitehead's 1898 text "A treatise on universal algebra" [Whi98] that significant parts of mathematics have a lot of structure, structure which we can take advantage of. The flat list of 342 structures gathered by Peter Jipsen [Jip] is both impressively large, and could easily be greatly extended. Another beautiful source of structure in a theory graph is that of *modal logics*; John Halleck's web pages on Logic System Interrelationships [Hal] is quite eye opening.

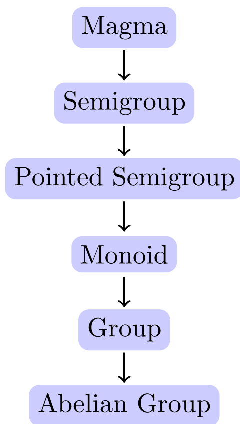



Fig. 1. Theories

Figure 1 shows what we are talking about: The *presentation* of the theory `Semigroup` strictly contains that of the theory `Magma`, and so on[1]. It is therefore pointless for a human to enter this information multiple times – *if* it is actually possible to take advantage of this structure. Strict inclusions at the level of presentations is only part of the structure: for example, we know that a `Ring` actually contains two isomorphic copies of `Monoid`, where the isomorphism is given by a simple *renaming*. There are further commonalities to take advantage of, which we will explain later in this paper.

Another question that arises naturally: is there sufficient structure outside of the traditional realm of universal algebra, in other words, beyond single-sorted equational theories, to make it worthwhile to develop significant infrastructure to leverage that structure? Luckily for us, it turns out that there is.

We will also require tools to selectively hide (and reveal) this structure from end-users. This latter requirement stems from the observation [CF08] that *in practice*, when mathematicians are *using* theories rather than developing new ones, they tend to work in a rather "flat" namespace. An analogy: someone working in Group Theory will unconsciously assume the availability of all concepts from a standard textbook, with their usual names and meanings. As their goal is to get some work done, whatever structure system builders have decided to use to construct their system should not *leak* into the application domain. They may not be aware of the existence of pointed semigroups, nor should that awareness be forced upon them. On the other hand, some application domains do rely on the "structure of theories", so we cannot unilaterally hide this structure from all users either.

## 1.2 Contributions

We previously explained our core ideas in [CO12], where a variant of the *category of contexts* was presented as our setting for theory presentations. There we presented a simple term language for building theories, along with two (compatible) categorical semantics – one in terms of objects, another in terms of arrows. By using "tiny theories", this allowed reuse and modularity. We emphasized names, as the objects we are dealing with are syntactic and ultimately meant for human consumption. We also emphasized arrows: while this is categorically obvious, nevertheless the current literature on this topic is very object-centric. Put another way: most of the emphasis in other work is on operational issues, or evolved from operational thinking, while our approach is unabashedly denotational, whilst still taking names seriously.

We leverage that basis here, and extend our work in multiple ways[2]. First, we

[1] We are not concerned with *models*, whose inclusion go in the opposite direction.
[2] We provide a summary of the contributions here to guide the reader who wishes to focus on the new ideas, even though much of the terminology used in this paragraph is only defined later.

enhanced contexts with definitions. We treat these as first-class citizens, so that names introduced by definitions are dealt with in the same way as all other names. The categorical semantics is extended to a fibration of generalized extensions over contexts. This is not straightforward: taking names seriously prevents us from having a cloven fibration without a renaming policy. But once this machinery is in place, this allows us to build presentations by lifting views over extensions, a very powerful mechanism for defining new presentations. There are obstacles to taking the "obvious" categorical solutions: for example, having all pullbacks would require that the underlying type theory have subset types, which is something we do not want to force. Furthermore, equivalence of terms needs to be checked when constructing mediating arrows, which in some settings may have implications for the decidability of typechecking.

While the core ideas of [CO12] remain, the text has been almost completely rewritten. We have also made the choice to generally put the links to the related work and its relationship to our work on its own section (8) rather than in the text.

### 1.3 Plan of paper

We motivate our work with concrete examples in section 2. Section 3 lays out the basic (operational) theory, with concrete algorithms. The theoretical foundations of our work, the fibered category of contexts, is presented in full detail in section 4. This allow us in section 5 to formalize a combinator language for theory presentation combinators. We close with some discussion, related work and conclusions in sections 7–10.

## 2. MOTIVATION

We review, informally, the motivation for introducing a variety of combinators for creating new theory presentations from old. We use an informal syntax which should be readily understandable to anyone with a reasonable background in mathematics and type theory; section 5 will give a formal syntax and its semantics. Note that the "intuitive" combinators that we present here are purely motivational, as the semantics of some of these turn out to be awkward and/or contrived. We will thus have to build our formal language (almost) from scratch, based on the semantics we develop in sections 3 and 4. As we go, we will also comment on the problems which need to be overcome to obtain a reasonable solution.

It is important to remember, throughout this section, that our principal perspective is that of *system builders*. Our task is to form a bridge (via software) between tasks that end-users of a mechanized mathematics system may wish to perform, and the underlying (semantic) theory concerned. This bridge is necessarily syntactic, as syntax is the only entity which can be symbolically manipulated by computers. More importantly, we must respect the syntactic choices of users, even when these choices are not necessarily semantically relevant.

### 2.1 Extension

The simplest situation is where the presentation of one theory is included, verbatim, in another. Concretely, consider `Monoid` and `CommutativeMonoid`.

$$\texttt{Monoid} \triangleq \left\{ \begin{array}{rcl} U & : & \texttt{Type} \\ (\cdot) & : & (U,U) \to U \\ e & : & U \\ \texttt{right identity} & : & \forall x : U . x \cdot e = x \\ \texttt{left identity} & : & \forall x : U . e \cdot x = x \\ \texttt{associative} & : & \forall x,y,z : U . (x \cdot y) \cdot z = x \cdot (y \cdot z) \end{array} \right\}$$

$$\texttt{CommutativeMonoid} \triangleq \left\{ \begin{array}{rcl} U & : & \texttt{Type} \\ (\cdot) & : & (U,U) \to U \\ e & : & U \\ \texttt{right identity} & : & \forall x : U . x \cdot e = x \\ \texttt{left identity} & : & \forall x : U . e \cdot x = x \\ \texttt{associative} & : & \forall x,y,z : U . (x \cdot y) \cdot z = x \cdot (y \cdot z) \\ \texttt{commutative} & : & \forall x,y : U . x \cdot y = y \cdot x \end{array} \right\}$$

As expected, the only difference is that `CommutativeMonoid` adds a `commutative` axiom. Thus, given `Monoid`, it would be much more economical to define

$$\texttt{CommutativeMonoid} \triangleq \texttt{Monoid}\ \textbf{extended by}\ \{\texttt{commutative} : \forall x, y : U . x \cdot y = y \cdot x\}$$

### 2.2 Renaming

From an end-user perspective, our `CommutativeMonoid` has one flaw: such monoids are frequently written *additively* rather than multiplicatively. Let us call a commutative monoid written additively an *abelian monoid*, as we do with groups. Thus it would be convenient to be able to say

$$\texttt{AbelianMonoid} \triangleq \texttt{CommutativeMonoid}[\ (\cdot) \mapsto +,\ e \mapsto 0]$$

Immediately, one is led to ask: how are `AbelianMonoid` and `CommutativeMonoid` related? Traditionally, these are regarded as *equal*, for semantic reasons. However, since we are dealing with presentations, as syntax, we wish to regard them as *isomorphic* rather than equal[3] . In other words, we take a nominal rather than structural approach, since we are dealing with syntax. While working up to isomorphism is a minor inconvenience for the semantics, this enables us to respect user choices in names.

### 2.3 Combination

But even with these features, given `Group`, we would find ourselves writing

$$\texttt{CommutativeGroup} \triangleq \texttt{Group}\ \textbf{extended by}\ \{\texttt{commutative} : \forall a, b : U . a \cdot b = a \cdot b\}$$

which is problematic: we lose the relationship that every commutative group is a commutative monoid. In other words, we reduce our ability to transport results "for

[3] Univalent Foundations[Uni13] does not change this, as we **can** distinguish the two, *as presentations.*

free" to other theories, and must prove that these results transport, even though the morphism involved is (essentially) the identity. Thus it is natural to further extend our language with a facility that expresses this sharing. Taking a cue from previous work, we might want to say

$$\texttt{CommutativeGroup} \triangleq \textbf{combine}\ \texttt{CommutativeMonoid}, \texttt{Group}\ \textbf{over}\ \texttt{Monoid}$$

Informally, this can be read as saying that `Group` and `CommutativeMonoid` are both "extensions" of `Monoid`, and `CommutativeGroup` is formed by the union (amalgamated sum) of those extensions. In other words, by **over**, we mean to have a single copy of `Monoid`, to which we add the extensions necessary for obtaining `CommutativeMonoid` and `Group`. This implicitly assumes that our two `Monoid` extensions are meant to be orthogonal, in some suitable sense.

Unfortunately, while this "works" to build a sizeable library (say of the order of 500 concepts) in a fairly economical way, it is nevertheless brittle. Let us examine why this is the case. It should be clear that by `combine`, we really mean *pushout*. But a pushout is a 5-ary operation on 3 objects and 2 arrows; our syntax gives the 3 objects and leaves the arrows implicit. In other words, they have to be inferred. This is a very serious mistake: these arrows are (in general) impossible to infer, especially in the presence of renaming. As mentioned previously, there are two distinct arrows from `Monoid` to `Ring`, with neither arrow being "better" or somehow more canonical than the other. Furthermore, we know that pushouts can also be regarded as a 2-ary operation on compatible arrows. In other words, even though our goal is to produce *theory presentations*, using pushouts as a fundamental building block, gives us no choice but to *take arrows seriously*.

### 2.4 Arrows

If we revisit the extension and renaming operations, it is easy to see that these operations not only create a new presentation, they also create a map from the source presentation into the target presentation. For extensions, this is an injective map. In other words,

$$\texttt{CommutativeMonoid} \triangleq \texttt{Monoid}\,\textbf{extended by}\,\{\texttt{commutative} : \forall x, y : U . x \cdot y = y \cdot x\}$$

creates more than just `CommutativeMonoid`, it also creates a *morphism* from `Monoid` to `CommutativeMonoid`. These can be written explicitly, and in this case this would be

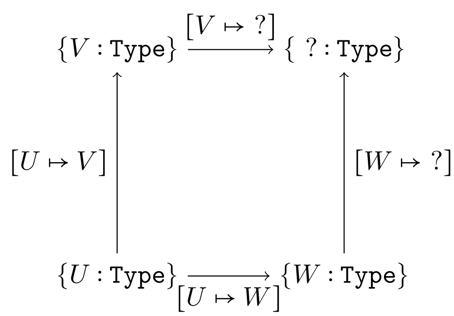
$$\begin{aligned}\texttt{MtoCM} \triangleq [&\texttt{U} := \texttt{U}, (\cdot) := (\cdot), \texttt{e} := \texttt{e}, \texttt{right identity} := \texttt{right identity},\\ &\texttt{left identity} := \texttt{left identity}, \texttt{associative} := \texttt{associative},\\ &\texttt{commutative} := \forall x, y : U . x \cdot y = y \cdot x] : \texttt{Monoid} \Rightarrow \texttt{CommutativeMonoid}\end{aligned}$$

where we use $\Rightarrow$ to indicate that this is a construction, and $:=$ to mean that this is an assignment of terms of `Monoid` to names of `CommutativeMonoid`. Clearly this would be very tedious to write out for larger theories. In *concrete syntax*, we would prefer to write just the non-identity parts, so that for this case we would prefer

$$\texttt{MtoCM} \triangleq [\texttt{commutative} := \forall x, y : U . x \cdot y = y \cdot x] : \texttt{Monoid} \Rightarrow \texttt{CommutativeMonoid}$$

which is easily seen to be isomorphic to the definition we started with. Thus it would be better to simply infer these morphisms when we can. We will however not make inferability a requirement.

For renaming, it is natural to require that the map on names causes no collisions, as that would rename multiple concepts to the same name. While this is a potentially interesting operation on presentations, this is not the operation that users have in mind for *renaming*. Collision-free renamings also induce an injective map.




Pushouts do create arrows as well, but unfortunately renamings are a problem: there are simple situations where there is no canonical name for some of the objects in the result. For example, take the presentation of `Carrier`, aka $\{U : \texttt{Type}\}$ and the arrows induced by the renamings $U \mapsto V$ and $U \mapsto W$; while the result will necessarily be isomorphic to `Carrier`, there is no canonical choice of name for the end result. This is one problem we must solve. The Figure above left illustrates the issue. It also illustrates that we really do compute *amalgamated sums* and not simply syntactic union.

In general, a map from one presentation to another will be called a *view*. For example, one can witness that the additive naturals form a monoid with a statement such as

$$\textsf{view } \texttt{Nat} \textsf{ as } \texttt{Monoid} \textsf{ via } [U := \mathbb{N}, (\cdot) := +_{\mathbb{N}}, e := 0, \ldots] \tag{1}$$

where we elide the names of the proofs. The right hand side of an assignment in a view does not need to be a symbol, it can be any well-typed term. For example, we can have a view from `Magma` to itself which maps the binary operation to its opposite:

$$\left\{ \begin{array}{lcl} U & : & \texttt{Type} \\ (\cdot) & : & (U,U) \to U \end{array} \right\} \xrightarrow{[U := \; U, \cdot := \; \textsf{flip} \; \cdot]} \left\{ \begin{array}{lcl} U & : & \texttt{Type} \\ (\cdot) & : & (U,U) \to U \end{array} \right\} \tag{2}$$

## 2.5 Little Theories

One important observation is that *contexts* of a type theory (or a logic) contain the same information as a *theory presentation*. Given a context, theorems about specific structures can be constructed by transport along views [FGT92]. For example, in the context of the definition of `Monoid` (2.1), we can prove that the identity element, $e$, is unique.

$$\forall e' : U. \left( (\forall x. e' \cdot x = x) \vee (\forall x. x \cdot e' = x) \right) \to e' = e$$

In order to apply this theorem in other contexts, we can provide a view from one theory presentation to another. For example, consider the theory presentation of semi-rings.

$$\texttt{Semiring} \triangleq \left\{ \begin{array}{rcl} U & : & \texttt{Type} \\ (+) & : & (U,U) \to U \\ (\times) & : & (U,U) \to U \\ 0 & : & U \\ 1 & : & U \\ \texttt{additive associative} & : & \forall x,y,z:U.\,(x+y)+z = x+(y+z) \\ \texttt{additive commutative} & : & \forall x,y:U.x+y = y+x \\ \texttt{additive left identity} & : & \forall x:U.0+x = x \\ \texttt{additive right identity} & : & \forall x:U.x+0 = x \\ \texttt{multiplicative associative} & : & \forall x,y,z:U.\,(x\times y)\times z = x\times(y\times z) \\ \texttt{multiplicative left identity} & : & \forall x:U.1\times x = x \\ \texttt{multiplicative right identity} & : & \forall x:U.x\times 1 = x \\ \texttt{left distributive} & : & \forall x,y,z:U.x\times(y+z) = x\times y + x\times z \\ \texttt{rightt distributive} & : & \forall x,y,z:U.\,(y+z)\times x = y\times x + z\times x \end{array} \right\}$$

There are two naturally induced views from Monoid to Semiring, one assigning $\cdot$ to $\times$ and $e$ to 1, and another assigning $\cdot$ to $+$ and $e$ to 0 (with the views also assigning the monoid axioms to their respective axioms). Each of these two views can be used to transport our example theorem to prove that 0 and 1 are unique with respect to their associated binary operations.

But these are not the only views from Monoid to Semiring. We do not have to restrict to assigning constants to constants – we could map constants to arbitrary terms (in the underlying language). For example we could send $\times$ to $\lambda x, y : U.y \times x$.

Which leads to the inevitable conclusion that, in general, we need an explicit language for defining views. But we have to proceed with care, otherwise we risk making simple situations complicated. For example, if we required explicit identity views for extensions, this would be semantically correct but painfully verbose in practice, as was pointed out earlier.

### 2.6 Models

It is important to remember that models are contravariant: while there is a presentation view from Monoid to CommutativeMonoid, the model morphisms are from $[\![\texttt{CommutativeMonoid}]\!]$ to $[\![\texttt{Monoid}]\!]$. Theorems are also contravariant with respect to *model* morphisms, so that they travel in the same direction as presentation views.

In this way a view *to* the empty theory presentation provides models of presentations by assigning every constant to a closed term. It is worthwhile noting that these models are internal to the underlying logic, rather than necessarily being Set-models. For example, if our underlying logic can express the existence of a type of natural numbers, $\mathbb{N}$, then the view given by (1) can be used to transport our example theorem to prove that 0 is the unique identity element for $+_{\mathbb{N}}$.

### 2.7 Tiny Theories

We noticed in our experiments [CFJ+11] that for ease of extension, it was best to use *tiny* theories, in other words presentations which add in a single concept at a time. This is useful both for defining pure signatures (presentations with no axioms) as well as when defining properties such as commutativity. Typically one proceeds by first defining the smallest typing context in which the property can be stated. For commutativity, `Magma` is the smallest such context – which also turns out to be a signature. We can then obtain the structures we are actually interested in via a "mixin" of the necessary properties over a base signature.

An example might make this clearer. Suppose we want to construct the presentation of `CommutativeSemiring` by adding the commutativity property to `Semiring` (see §2.5). As commutativity is defined as an extension to `Magma`, we need a view from `Magma` to `Semiring`. This view will tell us (exactly!) which binary operation we want to make commutative. Here we would pick the view that maps $U$ to $U$ and $(\cdot)$ to $(\times)$ We can then combine that view with the injection from `Magma` to `CommutativeMagma` to produce a `CommutativeSemiring` presentation.

$$\texttt{CommutativeSemiring} \triangleq \left\{ \begin{array}{rcl} U & : & \texttt{Type} \\ (+) & : & U \to U \to U \\ (\times) & : & U \to U \\ 0 & : & U \\ 1 & : & U \\ & \cdots & \\ \texttt{multiplicative commutative} & : & \forall xy : U . x \times y = y \times x \\ & \cdots & \end{array} \right\}$$

We see that this operation also requires that we provide a renaming that maps the axiom name "commutative" to "multipliciative commutative" in order to avoid the possibility of name collision (as addition was already commutative in `Semiring`).

### 2.8 Constructions

It is worthwhile noticing that there is nothing specific to `CommutativeGroup` in the renaming $\cdot \mapsto +, e \mapsto 0$, this can be applied to any theory where the pairs $(\cdot, +)$ and $(e, 0)$ have compatible signatures (including the case where they are not present). Similarly, `extend` really defines a "construction" which can be applied to any presentation whenever all the symbols used in the extension are defined. In other words, a reasonable semantics should associate a whole class of arrows[4] to these operations. While it is tempting to think that these operations will induce some endofunctors on presentations, this is not quite the case: name clashes will prevent that.

### 2.9 Problems

Clearly we need to have a setting in which extensions, renamings and combinations (or mixins) make sense. We will need to play close attention to names, both to allow pleasant names and prevent accidental collisions. In other words, to be able to maintain human-readable names for all concepts, we will put the burden on the *library developers* to come up with a reasonable naming scheme, rather than to push that issue onto end users. Another way to see this is that symbol choice

[4]We are again being deliberately vague here, section 4 will make this precise.

carries a lot of intentional, as well as contextual, information which is commonly used in mathematical practice.

Views will need to be formally defined, as well as a convenient language for dealing with them. While in some situations, it is imperative to be explicit about views, at other times they are obvious or easily inferred; in those latter situations, usability dictates that we should let the system do the heavy lifting for us.

Furthermore, we do want to use both the little theories and tiny theories method, so our language (and semantics) needs to allow, even promote, that style. We will see that, semantically, not all views have the same compositional properties. We will thus want to single out, syntactically, as large a subset of well-behaved views as possible, even though we know we can't be complete.

Our earlier attempt used an explicit base for **combine**, which only works for medium-scale libraries: we need to work more directly with views themselves. A common solution uses long names, which automatically generates (new, long) names to uniquely identify common names. But this has the effect of leaking the details of how a presentation was constructed into the names of the constants of the new presentation. This essentially prevents later refinements, as all these names would change. As far as we can tell, any automatic naming policy will suffer from this problem, which is why we insist on having the library developers explicitly deal with name clashes. We can then check that this has been done consistently. In practice few renamings are needed, so allowing the empty renaming annotation to denote the identity renaming scheme makes our design choice lightweight.

## 3. BASIC SEMANTICS

In this section we present the necessary definitions from (dependent) type theory and category theory which will form the basis of our theory presentation combinators. First we formally describe theory presentations and views, then we describe the semantics of our combinators.

Our presentations depend on a background type theory, but is otherwise agnostic as to many of the internal details of that theory. From this type theory we require the following:

—An infinite set of variable names $\mathbb{V}$.

—A typing judgement for terms $s$ of type $\sigma$ in a context $\Gamma$ which we write $\Gamma \vdash s : \sigma$.

—A kinding judgement of types $\sigma$ of kind $\kappa$ in a context $\Gamma$ which we write $\Gamma \vdash \sigma : \kappa : \Box$. We further assume that the set of valid kinds $\kappa : \Box$ is given and fixed.

—A definitional equality (a.k.a. convertibility) judgement of terms $s_1$ of type $\sigma_1$ and $s_2$ of type $\sigma_2$ in a context $\Gamma$, which we write $\Gamma \vdash s_1 : \sigma_1 \equiv s_2 : \sigma_2$. We will write $\Gamma \vdash s_1 \equiv s_2 : \sigma$ to denote $\Gamma \vdash s_1 : \sigma \equiv s_2 : \sigma$.

—A notion of substitution on terms. Given a list of variable assignments $[x_i \to s_{a_i}]_{i<n}$ and an expression $e$ we write $e\,[x_i := s_{a_i}]_{i<n}$ for the term $e$ after simultaneous substitution of variables $\{x_i\}_{i<n}$ by the corresponding term in the assignment.

We will often denote an assignment by $v$, and its application to a term $e$ by $e\,[v]$.

## 3.1 Theory Presentations

A theory presentation is a well-typed list of declarations and definitions. More formally, Figure 2 gives the formation rules. In this definition, we use $|\Gamma|$ to denote the set of variables of a well-formed context $\Gamma$. Explicitly, it is given by

$$|\varnothing| = \varnothing \qquad |\Gamma\ ;\ x:\sigma| = |\Gamma| \cup \{x\} \qquad |\Gamma\ ;\ x:\sigma := s| = |\Gamma| \cup \{x\}$$

Here $x : \sigma := s$ denotes the declaration of a new synonym $x$ for term $s$ of type $\sigma$. It is possible to develop this theory without declarations, however including them appears to make both the theory and practical implementations easier.

$$\frac{}{\varnothing\ \texttt{ctx}} \qquad \frac{\Gamma\ \texttt{ctx} \quad x \notin |\Gamma| \quad \Gamma \vdash \sigma : \kappa : \Box}{(\Gamma\ ;\ x:\sigma)\ \texttt{ctx}} \qquad \frac{\Gamma\ \texttt{ctx} \quad x \notin |\Gamma| \quad \Gamma \vdash s : \sigma}{(\Gamma\ ;\ x:\sigma := s)\ \texttt{ctx}}$$

Fig. 2. Formation rules for contexts

## 3.2 Views

A view from a theory presentation $\Gamma$ to a theory presentation $\Delta$ is an assignment of well-typed expressions in $\Delta$ to declarations of $\Gamma$. The assigments transport well-typed terms in the context $\Gamma$ to well-typed terms in $\Delta$, by substitution. More formally,

$$\frac{\Delta\ \texttt{ctx}}{[\,] : \varnothing \to \Delta} \qquad \frac{(\Gamma\ ;\ x:\sigma)\ \texttt{ctx} \quad [v] : \Gamma \to \Delta \quad \Delta \vdash r : \sigma[v]}{[v, x := r] : (\Gamma\ ;\ x:\sigma) \to \Delta}$$

$$\frac{(\Gamma\ ;\ x:\sigma) \quad [v] : \Gamma \to \Delta \quad \Delta \vdash r \equiv s[v] : \sigma[v]}{[v, x := r] : (\Gamma; x:\sigma := s) \to \Delta}$$

Fig. 3. Formation rules for views.

There is a subtle but important distinction between assignments, $[v]$ and views, $[v] : \Gamma \to \Delta$. A view is made up of 3 components: an assignment, a source presentation and a target presentation. In particular, the same assignment can occur in different views.

3.2.1 *Extensions and Inclusions.* An *extension* is a special type of view, which we denote

$$[a \mapsto r_a]_{a \in |\Gamma|} : \Gamma \to \Delta$$

where each expression $r_a$ is a unique variable name from $\Delta$. An inclusion is a special type of extension of the form

$$[a \mapsto a]_{a \in |\Gamma|} : \Gamma \to \Delta.$$

Inclusions have the nice property that there is a most one inclusion between any two theory presentations, and that inclusions form a poset of presentations. However this nice property is also a limitation. As we have hinted at before, `Ring` is an extension of `Monoid` in two different ways, and hence both extensions cannot be inclusions. We do not give inclusions any special status (unlike extensions); we draw attention to them here as many other systems make inclusions play a very special rôle.

3.2.2 *Composition of Views.* Given two views: $[v] : \Gamma \to \Delta$ and $[w] : \Delta \to \Phi$, we can compose them to create a view $[v] ; [w] : \Gamma \to \Phi$. If $[v] = [a := r_a]_{a \in |\Gamma|}$ then the composite view is

$$[v] ; [w] \triangleq [a := r_a [w]]_{a \in |\Gamma|} .$$

That this gives a well-defined notion of composition, and that it is associative is standard [Car86, Jac99, Tay99].

3.2.3 *Equivalence of Views.* Two views with the same domain and codomain, $[u], [v] : \Gamma \to \Delta$ are *equivalent* if $\Delta \vdash r_a : (\sigma_a [u]) \equiv s_a : (\sigma_a [v])$ where

$$\begin{aligned} \Gamma &\triangleq [a : \sigma_a]_{a \in |\Gamma|} \\ [u] &\triangleq [a := r_a]_{a \in |\Gamma|} \\ [v] &\triangleq [a := s_a]_{a \in |\Gamma|} \end{aligned}$$

3.2.4 *The category of theory presentations.* We now have all the necessary ingredients to define the *category of theory presentations* $\mathbb{P}$ with theory presentations as objects, and *views* as morphisms. The identity inclusions are the identity morphisms, and views act on views by substitution, which is associative and respects the identity.

Note that in [CO12], we worked with $\mathbb{C} = \mathbb{P}^{op}$, which is traditionally called the *category of contexts*, which is more often used in categorical logic [Car86, Jac99, Tay99, Pit00]. But in our setting, and as is common in the context of specifications (see for example [BG77, Smi93, CoF04] amongst many others), we prefer to take our intuition from *textual inclusion* rather than *models*. Nevertheless, when it will be time to define the *semantics*, we will revert to using $\mathbb{C}$, as this not only simplifies certain arguments, it also makes our work easier to compare to that in categorical logic.

### 3.3 Combinators

Having defined theory presentations and views (including extensions), we can now define presentation and view combinators. In fact, all combinators in this section will end up working in tandem on presentations and views. They allow us, as with most combinators, to create new presentations/views from old, in a much more convenient manner than building everything by hand.

The combinators are: extend, rename, combine and mixin. This list should be unsurprising given §2. Although we expect the majority of theory presentations and views will be constructed with these combinators, a few complex views will need to be defined directly. The reader may have noticed the absence of combinators such as delete or hide: this is quite purposeful on our part. While the operational semantics on theory presentations for these is "obvious", the denotational semantics in terms of theory morphisms is backwards, and has distasteful properties.

We give the full details of the constructions, which are completely deterministic. These can serve as a direct design for an implementation. In other words, this section gives an operational semantics for the combinators. In the next section, we will give them a categorical semantics; we make a few inline remarks here to help the reader understand why we choose a particular construction.

3.3.1 *Renaming.* Given a presentation $\Gamma$ and an injective renaming function $\pi : |\Gamma| \to \mathbb{V}$ we can construct a new theory presentation $\Delta \triangleq \Gamma\left[a \mapsto \pi\left(a\right)\right]_{a\in|\Gamma|}$ by renaming $\Gamma$'s symbols: we will denote this action of $\pi$ on $\Gamma$ by $\pi \cdot \Gamma$. We also construct a extension from $\left[a \to \pi\left(a\right)\right]_{a\in|\Gamma|} : \Gamma \to \pi \cdot \Gamma$ which provides a translation from $\Gamma$ to the constructed presentation $\pi \cdot \Gamma$; we denote this extension by $v_\pi$. For this construction as a whole, we use the notation

$$\mathfrak{R}\left(\Gamma, \pi : |\Gamma| \to \mathbb{V}\right) \triangleq \left\{ \begin{matrix} \texttt{pres} = \pi \cdot \Gamma \\ \texttt{extend} = v_\pi : \Gamma \to \pi \cdot \Gamma \end{matrix} \right\}$$

3.3.2 *Extend.* Given a theory presentation $\Gamma$, a fresh name $a$ and a well formed type $\sigma$ of some kind $\kappa$, (i.e. $\Gamma \vdash \sigma : \kappa : \Box$) we can construct a new theory presentation $\Delta \triangleq \Gamma; a : \sigma$ and the extension (an inclusion in this case) $\left[b \mapsto b\right]_{b\in|\Gamma|} : \Gamma \to \Delta$. More generally, given a sequence of fresh names, types and kinds, $\left\{a_i\right\}_{i<n}$, $\left\{\sigma_i\right\}_{i<n}$, and $\left\{\kappa_i\right\}_{i<n}$ we can define a sequence of theory presentations $\Gamma_0 \triangleq \Gamma$ and $\Gamma_{i+1} \triangleq \Gamma_i; a_i : \sigma_i$ so long as $\Gamma_i \vdash \sigma_i : \kappa_i : \Box$. Given such as sequence we construct a new theory presentation $\Delta \triangleq \Gamma_n$ with the extension (which is still an inclusion) $\left[b \mapsto b\right]_{b\in|\Gamma|} : \Gamma \to \Delta$. Of course $\Delta$ is the concatenation of $\Gamma$ with $\left\{a_i : \sigma_i : \kappa_i\right\}_{i<n}$. We will thus use $\Gamma \rtimes \Delta^+$ to denote the target of this view whenever the components of $\Delta^+$ are clear from context. However $\Delta^+$ is in general *not* a valid presentation, as it may depend on $\Gamma$. This is why we use an asymmetric symbol $\rtimes$.

It is worthwhile noting that general extensions $[u] : \Gamma \to \Delta$ as defined in §3.2.1 can be decomposed into a renaming composed with an $\rtimes$, in other words $[u] : \Gamma \to \Delta = [u] : \Gamma \to (\pi \cdot \Gamma \rtimes \Delta^+)$, where $\pi$ is defined by the action of the extension $[u]$ on $\Gamma$, namely $\pi \cdot \Gamma = \Gamma[u]$. We will use the notation $\Gamma[u] \rtimes \Delta^+$ as it makes the dependence on the extension clearer.

Extensions which are inclusions are traditionally called *display maps* in $\mathbb{C} = \mathbb{P}^{op}$, and our $\left[b \mapsto b\right]_{b\in|\Gamma|} : \Gamma \to (\Gamma; a : \sigma)$ in $\mathbb{P}$ is denoted by $\hat{a} : (\Gamma; a : \sigma) \to \Gamma$ in $\mathbb{C}$ [Tay99], and $\delta_a$ in [Jac99].

For notational convenience, we can encode the construction above as an explicit function from the inputs as given above, to a *record* containing two fields, `pres` (for presentation) and `extend` (for extension).

$$\mathfrak{E}\left(\Gamma, \Delta^+\right) \triangleq \left\{ \begin{matrix} \texttt{pres} = \Gamma \rtimes \Delta^+ \\ \texttt{extend} = \left[b \mapsto b\right]_{b\in|\Gamma|} : \Gamma \to \Gamma \rtimes \Delta^+ \end{matrix} \right\}$$

where $\Delta^+ = \left\{a_i : \sigma_i : \kappa_i\right\}_{i<n}$.

3.3.3 *Combine.* Given two extensions $\left[u_\Delta\right] : \Gamma \to \Delta$ and $\left[u_\Phi\right] : \Gamma \to \Phi$ and two injective renaming functions $\pi_\Delta : |\Delta| \to \mathbb{V}$ and $\pi_\Phi : |\Phi| \to \mathbb{V}$, we can combine them and generate a new theory presentation $\Xi$. We require that

$$\pi_\Delta\left(x\right) = \pi_\Phi\left(y\right) \Leftrightarrow \exists z \in |\Gamma| .\ x = z\left[u_\Delta\right] \wedge y = z\left[u_\Phi\right].$$

Say that the two extensions decompose as $\Delta = \Gamma\left[u_\Delta\right] \rtimes \Delta^+$ and $\Phi = \Gamma\left[u_\Phi\right] \rtimes \Phi^+$. Then we define $\Xi \triangleq \Xi_0 \rtimes \left(\Xi_\Delta \cup \Xi_\Phi\right)$ where $\Xi_0 \triangleq \Gamma\left[z \mapsto \pi_\Delta\left(z\left[u_\Delta\right]\right)\right]_{z\in|\Gamma|}$ (or equivalently $\Xi_0 \triangleq \Gamma\left[z \mapsto \pi_\Phi\left(z\left[u_\Phi\right]\right)\right]_{z\in|\Gamma|}$), $\Xi_\Delta \triangleq \Delta^+\left[x \mapsto \pi_\Delta\left(x\right)\right]_{x\in|\Delta|}$, and $\Xi_\Phi \triangleq \Phi^+\left[y \mapsto \pi_\Phi\left(y\right)\right]_{y\in|\Phi|}$. Note that, by construction, $\Xi_0 \rtimes \left(\Xi_\Delta \rtimes \Xi_\Phi\right)$ is equivalent to $\Xi_0 \rtimes \left(\Xi_\Phi \rtimes \Xi_\Delta\right)$; we denote this equivalence class[5] of views by $\Xi_0 \rtimes \left(\Xi_\Delta \cup \Xi_\Phi\right)$.

The combination operation also provides the two extensions $\left[v_\Delta\right] : \Delta \to \Xi$ and $\left[v_\Phi\right] : \Phi \to \Xi$ where

$$\begin{aligned}\left[v_\Delta\right] &\triangleq \left[x \mapsto \pi_\Delta\left(x\right)\right]_{x\in|\Delta|}\\ \left[v_\Phi\right] &\triangleq \left[y \mapsto \pi_\Phi\left(y\right)\right]_{y\in|\Phi|}\end{aligned}$$

A quick calculation shows that $\left[u_\Delta\right];\left[v_\Delta\right]$ is equal to $\left[u_\Phi\right];\left[v_\Phi\right]$ (and not just equivalent); we denote this joint arrow $\left[uv\right] : \Gamma \to \Xi$. Furthermore, combine provides a set of mediating views from the constructed theory presentation $\Xi$. Suppose we are given views $\left[w_\Delta\right] : \Delta \to \Omega$ and $\left[w_\Phi\right] : \Phi \to \Omega$ such that the the composed views $\left[u_\Delta\right];\left[w_\Delta\right] : \Gamma \to \Omega$ and $\left[u_\Phi\right];\left[w_\Phi\right] : \Gamma \to \Omega$ are equivalent. We can combine $\left[w_\Delta\right]$ and $\left[w_\Phi\right]$ into a mediating view $\left[w_\Xi\right] : \Xi \to \Omega$ where

$$\left[w_\Xi\right] \triangleq \left[\pi_\Delta\left(x\right) := x\left[w_\Delta\right]\right]_{x\in|\Delta|} \cup \left[\pi_\Phi\left(y\right) := y\left[w_\Phi\right]\right]_{y\in|\Phi|}.$$

This union is well defined since if $\pi_\Delta\left(x\right) = \pi_\Phi\left(y\right)$ then there exists $z$ such that $x = z\left[u_\Delta\right]$ and $y = z\left[u_\Phi\right]$, in which case $x\left[w_\Delta\right] = z\left[u_\Delta\right]\left[w_\Delta\right]$ and $y\left[w_\Phi\right] = z\left[u_\Phi\right]\left[w_\Phi\right]$ are equivalent since by assumption $\left[u_\Delta\right];\left[w_\Delta\right]$ and $\left[u_\Phi\right];\left[w_\Phi\right]$ are equivalent. It is also worthwhile noticing that this construction is symmetric in $\Delta$ and $\Phi$.

For this construction, we use the following notation, where we use the symbols as defined above (omitting type information for notational clarity)

$$\mathfrak{C}\left(u_\Delta, u_\Phi, \pi_\Delta, \pi_\Phi\right) \triangleq \left\{\begin{aligned}\texttt{pres} &= \Xi_0 \rtimes \left(\Xi_\Delta \cup \Xi_\Phi\right)\\ \texttt{extend}_\Delta &= \left[v_\Delta\right] : \Delta \to \Xi\\ \texttt{extend}_\Phi &= \left[v_\Phi\right] : \Phi \to \Xi\\ \texttt{diag} &= \left[uv\right] : \Gamma \to \Xi\\ \texttt{mediate} &= \lambda\ w_\Delta\ w_\Phi\ .\ w_\Xi\end{aligned}\right\}$$

The attentive reader will have noticed that we have painstakingly constructed an explicit *pushout* in $\mathbb{P}$. There are two reasons to do this: first, we need to be this explicit if we wish to be able to implement such an operation. And second, we do not want an arbitrary pushout, because we do not wish to work up to isomorphism as that would "mess up" the names. This is why we need user-provided injective renamings $\pi_\Delta$ and $\pi_\Phi$ to deal with potential name clashes. If we worked up to isomorphism, these renamings would not be needed, as they can always be manufactured by the system – but then these are no longer necessarily related to the users' names. Alternatively, if we use *long names* based on the (names of the) views, the method used to construct the presentations and views "leaks" into the names of the results, which we also consider undesirable.

3.3.4 *Mixin.* Given a view $\left[u_\Delta\right] : \Gamma \to \Delta$, an extension $\left[u_\Phi\right] : \Gamma \to \Phi$ and two disjoint injective renaming functions $\pi_\Delta : |\Delta| \to \mathbb{V}$ and $\pi_{\Phi^+} : |\Phi^+| \to \mathbb{V}$, where

[5] In practice, theory presentations are rendered (printed, serialized) using a topological sort where ties are broken alphabetically, so as to be construction-order indedepent.

the extension $\Phi$ decomposes as $\Phi = \Gamma[u_\Phi] \rtimes \Phi^+$, we can mixin the view into the extension, constructing a new theory presentation $\Xi$. We define $\Xi \triangleq \Xi_1 \rtimes \Xi_2$ where

$$\begin{aligned}
\Xi_1 &\triangleq \Delta\left[x \mapsto \pi_\Delta(x)\right]_{x\in|\Delta|}\\
\Xi_2 &\triangleq \Phi^+\left[y := \pi'_{\Phi^+}(y)\right]_{y\in|\Phi|}\\
\pi'_{\Phi^+}(y) &\triangleq \begin{cases} z\left[u_\Delta\right]\left[x \mapsto \pi_\Delta(x)\right]_{x\in|\Delta|} & \text{when there is a } z \in |\Gamma| \text{ such that } z\left[u_\Phi\right] = y\\ \pi_{\Phi^+}(y) & \text{when } y \in |\Phi^+| \end{cases}
\end{aligned}$$

The mixin also provides an extension $[v_\Delta] : \Delta \to \Xi$ and a view $[v_\Phi] : \Phi \to \Xi$, defined as

$$\begin{aligned}
\left[v_\Delta\right] &\triangleq \left[x \mapsto \pi_\Delta(x)\right]_{x\in|\Delta|}\\
\left[v_\Phi\right] &\triangleq \left[y := \pi'_{\Phi^+}(y)\right]_{y\in|\Phi|}
\end{aligned}$$

By definition of extension, there is no $z \in |\Gamma|$ that is mapped into $\Phi^+$ by $[u_\Phi]$. The definition of $\pi'_{\Phi^+}$ is arranged such that $[u_\Delta];[v_\Delta]$ is equal to $[u_\Phi];[v_\Phi]$ (and not just equivalent); so we can denote this joint arrow by $[uv] : \Gamma \to \Xi$. In other words, in a mixin, by only allowing renaming of the *new components* in $\Phi^+$, we insure commutativity *on the nose* rather than just up to isomorphism.

Mixins also provide a set of mediating views from the constructed theory presentaiton $\Xi$. Suppose we are given a view $[w_\Delta] : \Delta \to \Omega$ and view $[w_\Phi] : \Phi \to \Omega$ such that the composed views $[u_\Delta];[w_\Delta] : \Gamma \to \Omega$ and $[u_\Phi];[w_\Phi] : \Gamma \to \Omega$ are equivalent. We can combine $[w_\Delta]$ and $[w_\Phi]$ into the mediating view $[w_\Xi] : \Xi \to \Omega$ defined as

$$\left[w_\Xi\right] \triangleq \left[\pi_\Delta(x) := x\left[w_\Delta\right]\right]_{x\in|\Delta|} \cup \left[\pi_{\Phi^+}(y) := y\left[w_\Phi\right]\right]_{y\in|\Phi^+|}.$$

For mixin, again using the symbols as above, we denote the construction results as

$$\mathfrak{M}(u_\Delta, u_\Phi, \pi_\Delta, \pi_\Phi) \triangleq \left\{\begin{aligned}
\texttt{pres} &= \Xi_1 \rtimes \Xi_2\\
\texttt{extend}_\Delta &= [v_\Delta] : \Delta \to \Xi\\
\texttt{view}_\Phi &= [v_\Phi] : \Phi \to \Xi\\
\texttt{diag} &= [uv] : \Gamma \to \Xi\\
\texttt{mediate} &= \lambda\ w_\Delta\ w_\Phi\ .\ w_\Xi
\end{aligned}\right\}$$

Symbolically the above is very similar to what was done in combine, and indeed we are constructing all of the data for a specific pushout. However in this case the results are not symmetric, as seen from the details of the construction of $\Xi_1$ and $\Xi_2$, which stems from the fact that in this case $[v_\Phi]$ is an arbitrary view rather than an extension.

3.3.5 *Reification of views.* Although we (will) have a syntax and semantics for views, there are times when we wish to take views and treat them as first-class objects. For example, if we want to show that the set of all (small) groups and group homomorpisms forms a category, we need to be able to have a "theory" of group homomorphisms. But we can think of an even simpler example: we would like to talk about the *theory* of opposite magmas (see the view 2 in §2.4). To do this, we need to somehow internalize (reify) this view: this is a further reason to add declarations to our presentations (§3.1).

Suppose we are given a view $[v] : \Gamma \to \Delta$ We want to define a new presentation which internalizes $[v]$. A priori, this would require copying all of $\Gamma$ and $\Delta$ into a new presentation, and then define relations between the terms of $\Gamma$ and $\Delta$ via $[v]$. However, $\Delta$ might share some names with $\Gamma$, with some sharing of names, both on purpose and accidental. While we could rename everything in $\Delta$ and use $[v]$ to recover sharing, this is wasteful. In this case, we will only ask for a renaming for those names of $\Delta$ which introduce definitions.

Given an injective renaming function $\pi : |\Delta| \to \mathbb{V}$, we can define a new presentation $\Xi \triangleq \Gamma \rtimes \pi \cdot \Delta \rtimes w$ where $w \triangleq [\pi(z) : \sigma[v] := z[v]]_{z:\sigma\in\Gamma}$. Note how we have used the convertibility axiom from the formation rules of views (Fig. 3) in the definition of $w$. Naturally, we also get extensions $u_\Gamma : \Gamma \hookrightarrow \Xi = [b \mapsto b]_{b\in|\Gamma|}$ and $u_\Delta : \Delta \hookrightarrow \Xi = [b \mapsto \pi b]_{b\in|\Delta|}$.

For internalization, using the symbols as above, we denote results of the construction as

$$\mathfrak{I}(v, \pi) \triangleq \left\{ \begin{array}{r} \texttt{pres} = \Xi \\ \texttt{extend}_\Gamma = u_\Gamma \\ \texttt{extend}_\Delta = u_\Delta \end{array} \right\}$$

## 4. THE CATEGORICAL THEORY OF SEMANTICS

At first glance, the definitions of combine and mixin may appear ad hoc and overly complicated. This is because, in practice, the renaming functions $\pi_\Delta$ and $\pi_\Phi$ are frequently the *identity*. The main reason for this is that mathematical vernacular uses a lot of rigid conventions, such as usually naming an associative, commutative, invertible operator which possesses a unit +, and the unit 0, backward composition is $\circ$, forward composition is ;, and so on. But the usual notation of lattices is different than that of semirings, even though they share a similar ancestry – renamings are clearly necessary at some point.

The details of the combinators combine and mixin can be motivated by giving them a categorical specification. When we do, we find out that the mixin operation is a Cartesian lifting in a suitable fibration, and the combine operation is a special case of mixin.

While our primary interest is in theory presentations, the bulk of the categorical work in this area has been done on the category of contexts, which is the opposite category. To be consistent with the existing literature, we will give our semantics in terms of $\mathbb{B} = \mathbb{P}^{op}$. Thus if $[v] : \Gamma \to \Delta$ is a view from the theory presentation $\Gamma$ to the theory presentation $\Delta$, then $[v]$ is an arrow from $\Delta$, considered as a context, to $\Gamma$, considered as a context. We will write such arrows as $[v] : \Delta \leftarrow \Gamma$ as an arrow from $\Delta$ to $\Gamma$ when we are considering the category of contexts. Composition of two arrows is simply the composition of views.

### 4.1 Semantics

The category of contexts forms the base category for a fibration. The fibered category $\mathbb{E}$ is the category of theory extensions. The objects of $\mathbb{E}$ are extensions of theory presentations. We write such objects as $[u] : \Gamma \hookrightarrow \Delta$ where $\Gamma$ is the base of the extension and $\Delta$ is the extended theory presentation. The notation is to remind the reader that the underlying arrows are in fact monos. Arrows between two extensions is a pair of views forming a commutative square with the extensions. Thus given extensions $[u_2] : \Gamma_2 \hookrightarrow \Delta_2$ and $[u_1] : \Gamma_1 \hookrightarrow \Delta_1$, then an arrow between these consists of two arrows $[v_\Delta] : \Delta_2 \leftarrow \Delta_1$ and $[v_\Gamma] : \Gamma_2 \leftarrow \Gamma_1$ from $\mathbb{B}$ such that $[u_1] \circ [v_\Delta] = [v_\Gamma] \circ [u_2] : \Delta_2 \leftarrow \Gamma_1$. When we need to be very precise, we write such an arrow as $[u_2] \begin{array}{ccc} & [v_\Delta] & \\ \Delta_2 & \leftarrow & \Delta_1 \\ \uparrow & & \uparrow \\ \Gamma_2 & \leftarrow & \Gamma_1 \\ & [v_\Gamma] & \end{array} [u_1]$. We will write $v_\Gamma^\Delta : u_2 \leftleftarrows u_1$ whenever the rest of the information can be inferred from context. When given a specific arrow in $\mathbb{E}$, we will use the notation $e^{\leftleftarrows}$ for its name.

A fibration of $\mathbb{E}$ over $\mathbb{B}$ is defined by giving a suitable functor from $\mathbb{E}$ to $\mathbb{B}$. Our "base" functor sends an extension $[e] : \Gamma \hookrightarrow \Delta$ to $\Gamma$ and sends an arrow $v_\Gamma^\Delta : u_2 \leftleftarrows u_1$ to its base arrow $[v_\Gamma] : \Gamma_2 \leftarrow \Gamma_1$.

THEOREM 4.1. *This base fibration is a Cartesian fibration.*

This theorem, in slightly different form, can be found in [Jac99] and [Tay99]. We give a full proof here because we want to make the link with our mixin construction explicit. We use the results of §3.3 directly.

PROOF. Suppose $[u_\Delta] : \Delta \leftarrow \Gamma$ is an arrow in $\mathbb{B}$ (a view), and $[u_\Phi] : \Gamma \hookrightarrow \Phi$ is an object of $\mathbb{E}$ in the fiber of $\Gamma$ (i.e. an extension). We need to construct a Cartesian lifting of $[u_\Delta]$, which is a Cartesian arrow of $\mathbb{E}$ over $[u_\Delta]$. The components of the mixin construction are exactly the ingredients we need to create this Cartesian lifting. Let $\pi_\Delta : |\Delta| \to \mathbb{V}$ and $\pi_{\Phi^+} : |\Phi^+| \to \mathbb{V}$ be two disjoint injective renaming functions. Note that such $\pi_\Delta$ and $\pi_{\Phi^+}$ always exist because $\mathbb{V}$ is infinite while $|\Delta|$ and $|\Phi^+|$ are finite. Then let

$$\mathfrak{M}(u_\Delta, u_\Phi, \pi_\Delta, \pi_\Phi) \triangleq \left\{ \begin{array}{r} \mathtt{pres} = \Xi \\ \mathtt{extend}_\Delta = [v_\Delta] : \Delta \to \Xi \\ \mathtt{view}_\Phi = [v_\Phi] : \Phi \to \Xi \\ \mathtt{diag} = [uv] : \Gamma \to \Xi \\ \mathtt{mediate} = \lambda\ w_\Delta\ w_\Phi\ .\ w_\Xi \end{array} \right\}$$

Then $e^{\leftleftarrows} \triangleq [v_\Delta] \begin{array}{ccc} & [v_\Phi] & \\ \Xi & \leftarrow & \Phi \\ \uparrow & & \uparrow \\ \Delta & \leftarrow & \Gamma \\ & [u_\Delta] & \end{array} [u_\Phi]$ is an arrow of $\mathbb{E}$ which is a Cartesian lift of $[u_\Delta]$.

Firstly, to see that $e^{\leftleftarrows}$ is in fact an arrow of $\mathbb{E}$, we note that $[v_\Delta] : \Delta \to \Xi$ is an extension, so $[v_\Delta] : \Delta \hookrightarrow \Xi$ is an object of $\mathbb{E}$. Next we need to show that $[v_\Delta] \circ [u_\Delta] = [v_\Phi] \circ [u_\Phi]$. Let $z \in |\Gamma|$. Then $z[u_\Delta][v_\Delta] = z[u_\Delta][x \mapsto \pi_\Delta(x)]$ by definition of $[v_\Delta]$. On the other hand, $z[u_\Phi][v_\Phi] = z[u_\Phi][y := \pi'_{\Phi^+}(y)]_{y \in |\Phi|}$ by definition of $[v_\Phi]$. However, $z[u_\Phi]$ is a variable since $u_\Phi$ is an extension, and by definition $\pi'_{\Phi^+}(z[u_\Phi]) = z[u_\Delta][x \mapsto \pi_\Delta(x)]_{x \in |\Delta|}$ so that we have $z[u_\Delta][v_\Delta] = z[u_\Phi][v_\Phi]$ as required.

Secondly we need to see that $e^{\leftleftarrows}$ is a Cartesian lift of $[u_\Delta] : \Delta \leftarrow \Gamma$. We note that it is plain to see that $[u_\Delta] : \Delta \leftarrow \Gamma$ is the base of $e^{\leftleftarrows}$. What remains is to show that for any other arrow $f^{\leftleftarrows} \triangleq \begin{array}{ccc} & [w_\Phi] & \\ \Omega & \leftarrow & \Phi \\ [w_\Psi]\ \upharpoonleft & & \upharpoonleft\ [u_\Phi] \\ \Psi & \leftarrow & \Gamma \\ & [w_\Gamma] & \end{array}$ from $\mathbb{E}$ and any arrow $[w_0] : \Psi \leftarrow \Delta$ from $\mathbb{B}$ such that $[w_\Gamma] = [u_\Delta] \circ [w_0] : \Psi \leftarrow \Gamma$, there is a unique mediating arrow

$h^{\leftleftarrows} \triangleq \begin{array}{ccc} & [w_\Xi] & \\ \Omega & \leftarrow & \Xi \\ [w_\Psi]\ \upharpoonleft & & \upharpoonleft\ [v_\Delta] \\ \Psi & \leftarrow & \Delta \\ & [w_0] & \end{array}$ from $\mathbb{E}$ such that

$$h^{\leftleftarrows} \ ; \ f^{\leftleftarrows} = e^{\leftleftarrows} \tag{3}$$

To show that such an $h^{\leftleftarrows}$ exists, we only need to construct $[w_\Xi] : \Omega \leftarrow \Xi$ and show that it has the required properties. We will show that the mediating arrow $[w]$ from the mixin construction given $[w_\Phi] : \Phi \rightarrow \Omega$ and $[w_\Delta] \triangleq [w_0] ; [w_\Psi] : \Delta \rightarrow \Omega$ is the required arrow.

First we note that $[u_\Delta] \circ [w_\Delta] = [u_\Phi] \circ [w_\Phi]$ as required by the mixin construction for the mediating arrow since $[u_\Delta] \circ [w_0] = [u_\Delta] \circ [w_0] \circ [w_\Psi] = [w_\Gamma] \circ [w_\Psi] = [u_\Phi] \circ [w_\Phi]$. Now taking $[w_\Xi] \triangleq [w]$ we need to show that $h^{\leftleftarrows}$ is a well defined arrow in $\mathbb{E}$ by showing it forms a commutative square. Suppose $x \in |\Delta|$. Then $x\,[v_\Delta]\,[w_\Xi] = \pi_\Delta(x)\,[w_\Xi] = x\,[w_\Delta] = x\,[w_0]\,[w_\Psi]$ as required. Next we need to show that equation (3) holds. It suffices to show that $[v_\Phi] \circ [w_\Xi] = [w_\Phi]$ since it is already required that $[u_\Delta] \circ [w_0] = [w_\Gamma]$. Suppose $y \in |\Phi|$. There are two possiblities, either $y = z\,[u_\Phi]$ for some $z \in |\Gamma|$, or $y \in |\Phi^+|$ where $\Phi = \Gamma\,[u_\Phi] \rtimes \Phi^+$. If $y \in |\Phi^+|$ then $y\,[v_\Phi]\,[w_\Xi] = \pi_{\Phi^+}(y)\,[w_\Xi] = y\,[w_\Phi]$ as requried. In case $y = z\,[u_\Phi]$, then $y\,[v_\Phi]\,[w_\Xi] = z\,[u_\Phi]\,[v_\Phi]\,[w_\Xi] = z\,[u_\Delta]\,[v_\Delta]\,[w_\Xi] = z\,[u_\Delta]\,[w_0]\,[w_\Psi] = z\,[w_\Gamma]\,[w_\Psi] = z\,[u_\Phi]\,[w_\Phi] = y\,[w_\Phi]$ as requiried.

Lastly we need to show that the mediating arrow $h^{\leftleftarrows}$ is the unique arrow one satisfying equation (3). Let $j^{\leftleftarrows}$ be another arrow of $\mathbb{E}$, where $j^{\leftleftarrows}$ must have the same shape as $h^{\leftleftarrows}$, but with $[w_\Xi]$ replaced with $[w'_\Xi]$. Suppose that

$$j^{\leftleftarrows} \ ; \ f^{\leftleftarrows} = e^{\leftleftarrows}$$

We need to show that $[w'_\Xi] = [w_\Xi]$. Suppose $z \in |\Xi|$. There are two possiblities. Either $z = x\,[v_\Delta]$ for some $x \in |\Delta|$ or $z = y\,[v_\Phi]$ for some $y \in |\Phi^+|$. Suppose $z = x\,[v_\Delta]$. Then $z\,[w'_\Xi] = x\,[v_\Delta]\,[w'_\Xi] = x\,[w_0]\,[w_\Psi] = x\,[v_\Delta]\,[w_\Xi] = z\,[w_\Xi]$ as required. On the other hand, suppose $z = y\,[v_\Phi]$. Then $z\,[w'_\Xi] = y\,[v_\Phi]\,[w'_\Xi] = y\,[w_\Phi] = y\,[v_\Phi]\,[w_\Xi] = z\,[w_\Xi]$ as required. So $[w'_\Xi] = [w_\Xi]$ and hence $j^{\leftleftarrows} = h^{\leftleftarrows}$, as required. □

The above proof illustrates that the mixin operation is characterized by the properties of a Cartesian lifting in the fibration of extensions. Notice that a Cartesian lift is only characterised up to isomorphism. Thus there are potentially many isomorphic choices for a Cartesian lift, and hence there are many possible choices for how to mixin an extension into a view. This is the underlying reason why the mixin construction requires a pair of renaming functions. The renaming functions pick out a particular choice of mixin from the many possibilities. This ability to specify which mixin construction to make is quite important as one cannot simply define a mixin to be “the” Cartesian lift, since “the” Cartesian lift is only defined up to isomorphism. It is important to remember that for *user syntax*, we cannot work up to isomorphism!

Next we will see that combine is a special case of mixin.

THEOREM 4.2. *Given two extensions* $[u_\Delta] : \Gamma \hookrightarrow \Delta$ *and* $[u_\Phi] : \Gamma \hookrightarrow \Phi$ *and renaming functions* $\pi_\Delta : |\Delta| \to \mathbb{V}$ *and* $\pi_\Phi : |\Phi| \to \mathbb{V}$ *sastifiying the requirement of the combine construction, then*

$$\mathfrak{M}(u_\Delta, u_\Phi, \pi_\Delta, \pi_{\Phi^+}) = \mathfrak{C}(u_\Delta, u_\Phi, \pi_\Delta, \pi_\Phi) \qquad (4)$$

*where* $\Phi = \Gamma[u_\Phi] \rtimes \Phi^+$ *and* $\pi_{\Phi^+} = \pi_\Phi \downharpoonleft_{|\Phi^+|}$, *and equation 4 is interpreted componentwise.*

PROOF. Suppose that

$$\mathfrak{C}(u_\Delta, u_\Phi, \pi_\Delta, \pi_\Phi) = \left\{\begin{array}{r} \texttt{pres} = \Xi_0 \rtimes (\Xi_\Delta \cup \Xi_\Phi) \\ \texttt{extend}_\Delta = [v_\Delta] : \Delta \to \Xi \\ \texttt{extend}_\Phi = [v_\Phi] : \Phi \to \Xi \\ \texttt{diag} = [uv] : \Gamma \to \Xi \\ \texttt{mediate} = \lambda\ w_\Delta\ w_\Phi\ .\ w_\Xi \end{array}\right\}$$

and

$$\mathfrak{M}(u_\Delta, u_\Phi, \pi_\Delta, \pi_{\Phi^+}) = \left\{\begin{array}{r} \texttt{pres} = \Xi' \\ \texttt{extend}_\Delta = [v'_\Delta] : \Delta \to \Xi' \\ \texttt{view}_\Phi = [v'_\Phi] : \Phi \to \Xi' \\ \texttt{diag} = [uv'] : \Gamma \to \Xi' \\ \texttt{mediate} = \lambda\ w_\Delta\ w_\Phi\ .\ w_{\Xi'} \end{array}\right\}$$

Recall that $\Xi = \Xi_0 \rtimes (\Xi_\Delta \cup \Xi_\Phi) = \Xi_0 \rtimes (\Xi_\Delta) \rtimes \Xi_\Phi$ where $\Xi_0 \triangleq \Gamma[z \mapsto \pi_\Delta(z[v_\Delta])]_{z\in|\Gamma|}$, $\Xi_\Delta \triangleq \Delta^+[x \mapsto \pi_\Delta(x)]_{x\in|\Delta|}$, and $\Xi_\Phi \triangleq \Phi^+[y \mapsto \pi_\Phi(y)]_{y\in|\Phi|}$. In particular note that $\Xi_0 = \Gamma[v_\Delta][z[v_\Delta] \mapsto \pi_\Delta(z[v_\Delta])]_{z\in|\Gamma|}$. Since $\Delta = \Gamma[v_\Delta] \rtimes \Delta^+$, we have that

$$\begin{aligned} \Xi_0 \rtimes \Xi_\Delta &= \Gamma[v_\Delta][z[v_\Delta] \mapsto \pi_\Delta(z[v_\Delta])]_{z\in|\Gamma|} \rtimes \Delta^+[x \mapsto \pi_\Delta(x)]_{x\in|\Delta|} \\ &= (\Gamma[v_\Delta] \rtimes \Delta^+)[x \mapsto \pi_\Delta(x)]_{x\in|\Delta|} \\ &= \Delta[x \mapsto \pi_\Delta(x)]_{x\in|\Delta|} \end{aligned}$$

Recall also that $\Xi' = \Xi'_1 \rtimes \Xi'_2$ where $\Xi'_1 \triangleq \Delta[x \mapsto \pi_\Delta(x)]_{x\in|\Delta|}$ and $\Xi'_2 \triangleq \Phi^+[y \mapsto \pi'_{\Phi^+}(y)]_{y\in|\Phi|}$. So we see that $\Xi'_1 = \Xi_0 \rtimes \Xi_\Delta$.

Next we show that $\pi'_{\Phi^+} = \pi_\Phi$. If $y \in |\Phi|$ then either $y \in |\Phi^+|$ or there is some $z \in \Gamma$ such that $y = z[v_\Phi]$. If $y \in |\Phi^+|$ then $\pi'_{\Phi^+}(y) = \pi_{\Phi^+}(y) = \pi_\Phi(y)$. If $y = z[v_\Phi]$, then $\pi'_{\Phi^+}(y) = z[u_\Delta][x \mapsto \pi_\Delta(x)]_{x\in|\Delta|} = \pi_\Delta(z[u_\Delta]) = \pi_\Phi(z[u_\Phi]) = \pi_\Phi(y)$. Therefore $\Xi'_2 = \Xi_\Phi$ and hence $\Xi' = \Xi$.

Next we need to show that $[v'_\Delta] = [v_\Delta]$ and $[v'_\Phi] = [v_\Phi]$. First we see that $[v'_\Delta]$ and $[v_\Delta]$ are both defined to be $[x \mapsto \pi_\Delta(x)]_{x\in|\Delta|}$, so clearly they are equal. Next we see that $[v_\Phi] \triangleq [y \mapsto \pi_\Phi(y)]_{y\in|\Phi|}$ and $[v'_\Phi] \triangleq [y \mapsto \pi'_{\Phi^+}(y)]_{y\in|\Phi|}$ are equal because $\pi'_{\Phi^+} = \pi_\Phi$. This also gives that $[uv] = [uv']$.

Lastly we show that the mediating arrow of the combine is the same as the mediating arrow of the mixin. Suppose we are given $[w_\Delta] : \Delta \to \Omega$ and $[w_\Phi] : \Phi \to \Omega$ such that $[u_\Delta];[w_\Delta] = [u_\Phi];[w_\Phi] : \Gamma \to \Omega$. To show that the mediating arrow produced by combine, $[w_\Xi] : \Xi \to \Omega$ is the same as the medating arrow produced by the mixin, it suffies to prove that the mediating arrow satifies the universal property of the Cartesian lift, since such an arrow is unique. Thus it suffieces to show that $[v_\Phi];[w_\Xi] = [w_\Phi] : \Phi \to \Omega$ and $[v_\Delta];[w_\Xi] = [w_\Delta] : \Delta \to \Omega$. Let $y \in |\Phi|$. Then $y[v_\Phi][w_\Xi] = \pi_\Phi(y)[w_\Xi] = y[w_\Phi]$. Let $x \in |\Delta|$. Then $x[v_\Delta][w_\Xi] = \pi_\Delta(x)[w_\Xi] = x[w_\Delta]$ as required. □

Combine is rather well-behaved. In particular,

PROPOSITION 4.1. *$\mathfrak{C}(u_\Delta, u_\Phi, \pi_\Delta, \pi_\Phi) = \mathfrak{C}(u_\Phi, u_\Delta, \pi_\Phi, \pi_\Delta)$, i.e. combine is commutative.*

It turns out that combine also satisfies an appropriate notion of associativity. In other words, we can compute limits of cones of extensions.

### 4.2 No Lifting Views over Views

Why do we restrict ourselved to the fibration of extensions? Why not allow mixins of arbitrary views over arbitrary views? If mixins over arbitrary views were allowed, then the notion of a Cartesian lifting reduces to that of a pullback. But to demand that the category of contexts and views be closed under *all* pullbacks would require too much from our type theory: we would have to have all equalizers (as we already have all products). In particular, at the type level, this would force us to have subset types, which is something we are not willing to impose. Thus a restriction is needed, and our proposed restriction of only mixing in extensions into views appears to be quite practical. Taylor [Tay99] is a good source of further reasons for the naturality of restricting to this case.

## 5. SYNTAX AND SEMANTICS OF PRESENTATION COMBINATORS

We are now ready to give a concrete syntax for our presentation and view combinators. This syntax reflects our desire to have a clean semantics, and thus is extracted from the previous section, rather than trying to patch up our intuitive syntax. In other words, we followed the development process of *prototyping to get an idea of what is needed*, *find an elegant denotational semantics*, and *redo everything to match the elegant semantics on the nose*.

We use $A, B$ to denote names at the presentation/view level, $x$ and $y$ to denote symbols, $t$ are terms of the underlying type theory, and $l$ are (raw) contexts from the underlying type theory.

$$\begin{aligned}
\mathrm{tpc} ::= {} & \textsf{Empty} \\
& \mid \textsf{Theory } \{l\} \\
& \mid \textsf{extend } A \textsf{ by } \{l\} \\
& \mid \textsf{combine } A\ r_1,\ B\ r_2 \\
& \mid \textsf{mixin } A\ r_1,\ B\ r_2 \\
& \mid \textsf{view } A \textsf{ as } B \textsf{ via } \mathrm{v} \\
& \mid A\ ;\ B \\
& \mid A\ \mathrm{r}
\end{aligned}
\qquad
\begin{aligned}
\mathrm{r} ::= {} & [\mathrm{ren}] \\
\mathrm{v} ::= {} & [\mathrm{assign}] \\
\mathrm{ren} ::= {} & x \mapsto y \\
& \mid \mathrm{ren},\ x \mapsto y \\
\mathrm{assign} ::= {} & x := t \\
& \mid \mathrm{assign},\ x := t
\end{aligned}$$

Informally, these forms correspond to the empty theory, an explicit theory, a theory extension, combining two extensions, mixing in a view and an extension, explicit view, sequencing views, and renaming.

What might be surprising is that we do not have a separate language for presentations and views. This is because our language does not have a single semantics in terms of presentations, extensions or views, but rather has *several* compatible semantics. In other words, our syntax will yield objects of $\mathbb{B}$, objects of $\mathbb{E}$ (i.e. extensions) and arrows of $\mathbb{B}$ (views).

The semantics is given by defining three partial maps, $[\![-]\!]_{\mathbb{B}} : \textsf{tpc} \rightharpoonup |\mathbb{B}|$, $[\![-]\!]_{\mathbb{E}} : \textsf{tpc} \rightharpoonup |\mathbb{E}|$, $[\![-]\!]_{\mathbb{B}^{\to}} : \textsf{tpc} \rightharpoonup \mathrm{Hom}_{\mathbb{B}}$. This is done by simultaneous structural recusion. We also use $[\![-]\!]_{\pi}$ for the straightforward semantics in $\mathbb{V} \to \mathbb{V}$ of a renaming.

$$\begin{aligned}
[\![-]\!]_{\mathbb{B}} &: \textsf{tpc} \rightharpoonup |\mathbb{B}| \\
[\![\textsf{Empty}]\!]_{\mathbb{B}} &= \varnothing \\
[\![\textsf{Theory } \{l\}]\!]_{\mathbb{B}} &= l \qquad\qquad \text{when } l \ \texttt{ctx} \\
[\![\textsf{extend } A \textsf{ by } \{l\}]\!]_{\mathbb{B}} &= \mathfrak{E}\left([\![A]\!]_{\mathbb{B}}, \mathrm{id}_{|A|}\right).\texttt{pres} \\
[\![\textsf{combine } A_1 r_1, A_2 r_2]\!]_{\mathbb{B}} &= \mathfrak{C}\left([\![A_1]\!]_{\mathbb{E}}, [\![A_2]\!]_{\mathbb{E}}, [\![r_1]\!]_{\pi}, [\![r_2]\!]_{\pi}\right).\texttt{pres} \\
[\![\textsf{mixin } A_1 r_1, A_2 r_2]\!]_{\mathbb{B}} &= \mathfrak{M}\left([\![A_1]\!]_{\mathbb{B}^{\to}}, [\![A_2]\!]_{\mathbb{E}}, [\![r_1]\!]_{\pi}, [\![r_2]\!]_{\pi}\right).\texttt{pres} \\
[\![\textsf{view } A \textsf{ as } B \textsf{ via } \mathrm{v}]\!]_{\mathbb{B}} &= \bot \\
[\![A; B]\!]_{\mathbb{B}} &= \textsf{cod } [\![A; B]\!]_{\mathbb{B}^{\to}} \\
[\![A\ r]\!]_{\mathbb{B}} &= \mathfrak{R}\left([\![A]\!]_{\mathbb{B}}, [\![r]\!]_{\pi}\right).\texttt{pres}
\end{aligned}$$

Recall that objects of $\mathbb{E}$ corresponds to those arrows of $\mathbb{B}$ (i.e. views) which are in fact extensions.

$$
\begin{aligned}
[\![-]\!]_{\mathbb{E}} &: \mathsf{tpc} \rightharpoonup |\mathbb{E}| \\
[\![\mathsf{Empty}]\!]_{\mathbb{E}} &= \mathbb{I}_{\varnothing} \\
[\![\mathsf{Theory}\ \{l\}]\!]_{\mathbb{E}} &= !_l : [\,] \to [\![l]\!]_{\mathbb{B}} \\
[\![\mathsf{extend}\ A\ \mathsf{by}\ \{l\}]\!]_{\mathbb{E}} &= \mathfrak{E}\left([\![A]\!]_{\mathbb{B}}, \mathrm{id}_{|A|}\right)\texttt{.extend} \\
[\![\mathsf{combine}\ A_1 r_1, A_2 r_2]\!]_{\mathbb{E}} &= \mathfrak{C}\left([\![A_1]\!]_{\mathbb{E}}, [\![A_2]\!]_{\mathbb{E}}, [\![r_1]\!]_{\pi}, [\![r_2]\!]_{\pi}\right)\texttt{.diag} \\
[\![\mathsf{mixin}\ A_1 r_1, A_2 r_2]\!]_{\mathbb{E}} &= \bot \\
[\![\mathsf{view}\ A\ \mathsf{as}\ B\ \mathsf{via}\ \mathrm{v}]\!]_{\mathbb{E}} &= \bot \\
[\![A; B]\!]_{\mathbb{E}} &= [\![A]\!]_{\mathbb{E}}; [\![B]\!]_{\mathbb{E}} \\
[\![A\ r]\!]_{\mathbb{E}} &= \mathfrak{R}\left([\![A]\!]_{\mathbb{B}}, [\![r]\!]_{\pi}\right)\texttt{.extend}
\end{aligned}
$$

Lastly, arrows of $\mathbb{B}$ are views.

$$
\begin{aligned}
[\![-]\!]_{\mathbb{B}^{\to}} &: \mathsf{tpc} \rightharpoonup \mathrm{Hom}_{\mathbb{B}} \\
[\![\mathsf{Empty}]\!]_{\mathbb{B}^{\to}} &= \mathbb{I}_{\varnothing} \\
[\![\mathsf{Theory}\ \{l\}]\!]_{\mathbb{B}^{\to}} &= !_l : [\,] \to [\![l]\!]_{\mathbb{B}} \\
[\![\mathsf{extend}\ A\ \mathsf{by}\ \{l\}]\!]_{\mathbb{B}^{\to}} &= \mathfrak{E}\left([\![A]\!]_{\mathbb{B}}, \mathrm{id}_{|A|}\right)\texttt{.extend} \\
[\![\mathsf{combine}\ A_1 r_1, A_2 r_2]\!]_{\mathbb{B}^{\to}} &= \mathfrak{C}\left([\![A_1]\!]_{\mathbb{E}}, [\![A_2]\!]_{\mathbb{E}}, [\![r_1]\!]_{\pi}, [\![r_2]\!]_{\pi}\right)\texttt{.diag} \\
[\![\mathsf{mixin}\ A_1 r_1, A_2 r_2]\!]_{\mathbb{B}^{\to}} &= \mathfrak{C}\left([\![A_1]\!]_{\mathbb{B}^{\to}}, [\![A_2]\!]_{\mathbb{E}}, [\![r_1]\!]_{\pi}, [\![r_2]\!]_{\pi}\right)\texttt{.diag} \\
[\![\mathsf{view}\ A\ \mathsf{as}\ B\ \mathsf{via}\ \mathrm{v}]\!]_{\mathbb{B}^{\to}} &= [v] : [\![A]\!]_{\mathbb{B}} \to [\![B]\!]_{\mathbb{B}} \\
[\![A; B]\!]_{\mathbb{B}^{\to}} &= [\![A]\!]_{\mathbb{B}^{\to}}; [\![B]\!]_{\mathbb{B}^{\to}} \\
[\![A\ r]\!]_{\mathbb{B}^{\to}} &= \mathfrak{R}\left([\![A]\!]_{\mathbb{B}}, [\![r]\!]_{\pi}\right)\texttt{.extend}
\end{aligned}
$$

All rules are strictly compositional except for $[\![A; B]\!]_{\mathbb{B}}$, but this is ok since $[\![A; B]\!]_{\mathbb{B}^{\to}}$ is compositional.

Note that we could have interpreted $[\![\mathsf{view}\ A\ \mathsf{as}\ B\ \mathsf{via}\ \mathrm{v}]\!]_{\mathbb{B}}$ as $\mathsf{cod}[\![\mathsf{view}\ A\ \mathsf{as}\ B\ \mathsf{via}\ \mathrm{v}]\!]_{\mathbb{B}^{\to}}$, rather than as $\bot$, but this is not actually helpful, since this is just $[\![B]\!]_{\mathbb{B}}$, which is not actually what we want. What we would really want is the result of doing the substitution $v$ into $A$, but the resulting presentation may no longer be well-formed. So we chose to interpret the attempt to take the object component of a view as a specification error. Similarly, even though we can give an interpretation as an extension for mixin when $A_1$ turns out to be an extension, and also for an extension $r$ in a view context (i.e. $\mathsf{view}\ A\ \mathsf{as}\ B\ \mathsf{via}\ \mathrm{r}$), we also choose to make these specification errors as well.

We should also note here that in our implementation, we allow raw renamings ([ren]) and assignments ([assign]) to be named, for easier reuse. While renamings can be given a simple categorical semantics (they induce a natural transformation on $\mathbb{B}$), assignments really need to be interpreted contextually since this requires checking that terms $t$ are well-typed.

Furthermore, we add a bit of syntactic sugar: $A||B$ stands for $\mathsf{combine}\ A\ [\,], B\ [\,]$, a rather common situation.

## 6. EXAMPLES

We show some progressively more complex examples, drawn from our library. These are chosen to illustrate the power of the combinators, but also how these do solve the various problems we highlighted in §2. In all the examples below, we are talking solely about presentations of theories; we will nevertheless drop "presentation" and talk about `Group` even though we are not interested in groups themselves, nor even "Group Theory". The reader needs to keep this distinction in mind when reading our examples.

The simplest use of combine comes very quickly in a hierarchy built using *tiny theories*, namely when we construct a pointed magma from a magma and (the theory of) a point.

```
Carrier := extend Empty by { U : type }
Magma := extend Carrier by { * : (U,U) -> U }
Pointed := extend Carrier by { e : U }
PointedMagma := Magma || Pointed
```

where we have used the || sugar for combine. Since $[\![\text{Magma}]\!]_{\mathbb{E}}$ and $[\![\text{Pointed}]\!]_{\mathbb{E}}$ are both arrows from $[\![\text{Carrier}]\!]_{\mathbb{B}}$, these can be combined into another extension $[\![\text{PointedMagma}]\!]_{\mathbb{E}} : [\![\text{Carrier}]\!]_{\mathbb{B}} \to [\![\text{PointedMagma}]\!]_{\mathbb{B}}$.
If we want a theory of *two* points, we need to rename one of them:

```
TwoPointed := combine Pointed [], Pointed [U |-> V]
```

We can just as easily extend by properties:

```
LeftUnital := extend PointedMagma by {
    axiom leftIdentity : forall x:U. e * x = x
}
```

This illustrates a design principle: properties should be defined as extensions of their minimal theory. Such minimal theories are most often *signatures*, in other words property-free theories. By the results of the previous section, this maximizes reusability. Even though signatures have no specific status in our framework, they arise very naturally as "universal base points" for theory development.

LeftUnital of course has a natural dual, RightUnital. While this is easy enough to define explicitly, this should nevertheless give pause, as this is really duplicating information which already exists. This can be solved using the following self-*view*:

```
Flip := view Magma as Magma via [ * |-> fun (x, y). y * x ]
```

Note that there is no interpretation for $[\![\text{Flip}]\!]_{\mathbb{B}}$; if we were to perform the substitution directly, we would obtain

```
Theory { U : type; fun (x,y). y * x : (U,U) -> U }
```

which is ill-defined since it contains the undefined symbol $*$.

One could be tempted to then write

```
RightUnital := mixin Flip [], LeftUnital []
```

but this is incorrect since LeftUnital is an extension from PointedMagma, not Magma. The solution is to write

```
RightUnital := mixin Flip [], (PointedMagma ; LeftUnital) []
```

which gives a correct answer, but with an axiom still called `leftIdentity`; the better solution is to write

```
RightUnital := mixin Flip [],
  (PointedMagma ; LeftUnital) [leftIdentity |-> rightIdentity]
```

which is the `RightUnital` we want. Note that the construction also make available an extension from `Magma` (as if we had done the construction manually) as well as views from `LeftUnital` and `Magma`.

Note that the syntax used above is sub-optimal: the path `PointedMagma`;`LeftUnital` may well be needed again, and should be named. In other words,

```
LeftUnit := PointedMagma ; LeftUnital
```

is a useful intermediate definition.

Note that the previous examples reinforce the importance of signatures, and of arrows from signatures to "interesting" theories as important, separate entities. For example, Monoid as an *extension* is most usefully seen as an arrow from the presentation PointedMagma.

Our machinery also allows one to construct the inverse view, from `LeftUnital` to `RightUnital`. Consider the view `Flip`;`LeftUnital` and the identity view from `LeftUnital` to itself. These are exactly the needed inputs for `mediate`, which returns a (unique) view from `LeftUnital` to `RightUnital`. Furthermore, we obtain (from the construction of the mediating view) that this view composes with the view from `RightUnital` to `LeftUnital` to give the identity. This is illustrated in Figure 6 where the $[\![-]\!]_{\mathbb{B}^\to}$ annotations on nodes are omitted; note that the arrows are in the *semantic category*, which is the opposite of the one for theory presentations. Let

$$RU = \mathfrak{C}\left([\![\texttt{Flip}]\!]_{\mathbb{B}^\to}, [\![\texttt{LeftUnital}]\!]_{\mathbb{E}}, [\![id]\!]_\pi, [\![[\texttt{leftIdentity} \mapsto \texttt{rightIdentity}]]\!]_\pi\right)$$

then $\texttt{Flip}_{RU} = RU.\texttt{view}_{\texttt{LeftUnital}}$ and

$$\texttt{Flip}_{LU} = RU.\texttt{mediate}_{\texttt{LeftUnital}}\left([\![\texttt{LeftUnital}]\!]_{\mathbb{E}}; [\![\texttt{Flip}]\!]_{\mathbb{B}^\to}, [\![id]\!]_{\mathbb{E}}\right)$$

The constrution of `mediate` insures that $\texttt{Flip}_{LU};\texttt{Flip}_{RU} = [\![id]\!]_{\mathbb{E}}$, *provided* that we know that

$$[\![\texttt{Flip}]\!]_{\mathbb{B}^\to}; [\![\texttt{Flip}]\!]_{\mathbb{B}^\to} = [\![id]\!]_{\mathbb{B}^\to} : [\![\text{Magma}]\!]_{\mathbb{B}} \to [\![\text{Magma}]\!]_{\mathbb{B}}.$$

The above identity is not, however, structural, it properly belongs to the underlying type theory: it boils down to asking if

$$\forall x : U.\texttt{flip}\,(\texttt{flip x}) =_{\beta\eta\delta} x$$

or, to use the notation of §3.1,

$$[U : \text{Type}, x : U] \vdash \texttt{flip}\,(\texttt{flip x}) \equiv x : U.$$

## 7. DISCUSSION

It is important to note that we are essentially parametric in the underlying type theory. From a categorical point of view, this is hardly surprising: this is the whole point of contextual categories [Car86]. A lot more features can be added to the

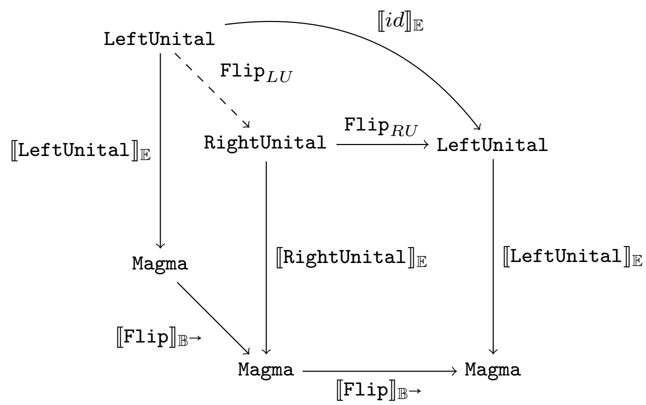



Fig. 4. Construction of `LeftUnital` and `RightUnital`. See the text for the interpretation.

type theory, at no harm to the combinators themselves – see Jacobs [Jac99] and Taylor [Tay99] for many such features.

One of the features we did choose to build in was to allow *definitions* in our contexts. This is especially useful when transporting theorems from one setting to another, as is done when using the "Little Theories" method [FGT92]. In this setting it is frequently beneficial to first build up *towers of conservative extensions* above each of the theories, so as to build up a more convenient common vocabulary, which then makes interpretations easier to build (and use).

Lastly, we have implemented a "flattener" for our semantics, which just turns a presentation $A$ given in our language into a flat presentation $\mathsf{Theory}\{\, l\}$ by computing $\mathsf{cod}\,(\llbracket A \rrbracket_{\mathbb{E}})$. We have been very careful to ensure that all our constructions leave no trace of the construction method in the resulting flattened theory. We strongly believe that users of theories do not wish to be burdened by such details, and we also want developers to have maximal freedom in designing a modular, reusable and maintainable hierarchy without worrying about backwards compatibility of the *hierarchy*, only the end results: the theory presentations.

## 8. RELATED WORK

We have been highly influenced by the early work of Burstall and Goguen [BG77, BG79], Doug Smith's Specware [Smi93, Smi99], and the work of Kapur, Musser and Stepanov on Tecton [KMS82, KM92]. They gave us the basic operational ideas, and some of the semantic tools we needed. But we quickly found out, much to our dismay, that neither of these approaches seemed to scale very well. Later, we were hopeful that the approach of CASL [CoF04] might work, but then found that their own base library was insufficiently factored and full of redundancies. Of the vast algebraic specification literature around this topic, we want to single out the work of Oriat [Ori00] on isomorphism of specification graphs as capturing similar ideas to ours on extreme modularity. And it cannot be emphasized enough how crucial Bart Jacob's book [Jac99] has been to our work.

Another line of influence is through universal algebra [Whi98, BS81], more precisely the *constructions* of universal algebra, rather than its theorems. That we can manipulate signatures as algebraic objects is firmly from that literature. Of course, we must generalize from the single-sorted equational approach of the mathematical literature, to the dependently typed setting. As we eschew all matters dealing with models, the syntactic manipulation aspects of universal algebra generalize quite readily. The syntactic concerns are also why *Lawvere theories* [Law04] are not as important to this work. Sketches [BW90] certainly could have been used, but would have led us too far away from the elegance of using structures already present in the $\lambda$-calculus (namely contexts) quite directly.

*Institutions* might also appear to be an ideal setting for our work. But even as the relation to categorical logic has been worked out [GMdP+07], it remains that these theories are largely semantic, in that they all work up to equivalence. This makes the theory of institutions significantly simpler, however it also makes it largely unusable for user-oriented systems: people really do care what names their symbols have in their theory presentations.

After we had largely finished our work, we found various research threads which had a lot in common with ours. They used different terminology, and frequently provided no implementations.

Proceeding in chronological order, the Harper-Mitchell-Moggi work on Higher-order Modules [HMM89] covers some of the same themes we do: a set of constructions (at the semantic level) similar to ours is developed for ML-style modules. However, they did not seem to realize that these constructions could be turned into an external syntax, with crucial application to structuring a large library of theories (or modules). Nor did they see the use of fibrations, since they avoided such issues "by construction". Moggi returned to this topic [Mog89], and did make use of fibrations as well as *categories with attributes*, a categorical version of contexts. Post-facto, it is possible to recognize some of our ideas as being present in Section 6 of that work; the emphasis is however completely different. In that same vein *Structured theory presentations and logic representations* [HST94] does have a set of combinators. However, it seems flawed: Definition 3.3 of the signature of a presentation requires that both parts of a union must have the same signature (to be well formed) and yet their Example 3.6 on the next page is not well formed! That being said, many parts of the theory-level semantics is the same. However, it is our morphism-level semantics which really allows one to build large hierarchies conveniently.

A rather overlooked 1997 Ph.D. thesis by Sherri Shulman [Shu97] presents a number of interesting combinators. Unfortunately the semantics are unclear, especially in cases where theories have parts in common; there are heavy restrictions on naming, and no renaming, which makes the building of large hierarchies fragile. Nevertheless, there is much kinship here, especially that of extreme modularity. If this work had been implemented in a mainstream tool, it would have saved us a lot of effort.

[Tay99] does worry more about syntax. Although the semantic component is there, there are no algorithms and no notion of building up a library. The categorical tools are presented too, but not in a way to make the connection clear, and lead to an implementable design.

MMT [RK13] is of course closest. But the structuring tools are still not as nice as we'd like — [DHS09] shows some examples. Many of the problems that we have identified as problems for scaling are still present. MMT does have some advantages: it is foundation independent, and possesses some rather nice web-based tools for pretty display. But their extend operation (named `include`) is theory-internal, and its semantics is not given through flattening (which they have yet to implement). The result is that their theory hierarchies explicitly suffer from the "bundling" problem, as lucidly explained in [SvdW11], who introduce type classes in Coq to help alleviate this problem. Furthermore, although theory morphisms are first class, obtaining the "right" ones seems to be entirely manual.

Isabelle's *locales* support *locale expressions*[Bal03], which are also reminiscent of ours. However, we are unaware of a denotational semantics for them; furthermore, neither combine nor mixin are supported. Axiom [JS92] does support theory formation operations, but these are quite restricted, as well as defined purely operationally. They were meant to mimic what mathematicians do informally when operating on theories. No semantics for them has ever been published.

Coq has both *Canonical Structures* and *type classes* [SvdW11], but no combinators to make new ones out of old. Similarly, Lean [dMKA+15] has some (still evolving) structuring mechanisms but not combinators to form new theories from old.

## 9. ACKNOWLEDGMENTS

We wish to thank Michael Kohlhase, Florian Rabe and William M. Farmer for many fruitful conversations on this topic. We also wish to thank the participants of the Dagstuhl Seminar 18341 "Formalization of Mathematics in Type Theory" whose interest in this topic helped insure that this paper got finished.

## 10. CONCLUSION

There has been a lot of work done in mathematics to give structure to mathematical theories, first via universal algebra, then via category theory. But even though a lot of this work started out being somewhat syntactic, very quickly it became mostly semantic within a non-constructive meta-theory, and thus largely useless for the purposes of concrete implementations and full automation.

Here we make the observation that, with a rich enough type theory, we can identify the category of theory presentations with the opposite of the category of contexts. This allows us to draw freely from developments in categorical logic, as well as to continue to be inspired by algebraic specifications. Interestingly, key here is to make the opposite choice as Goguen's (as the main inspiration for the family of OBJ languages) in two ways: our base language is firmly higher-order, as well as dependently typed, while our "module" language is first-order, and we work in the opposite category.

We provide a simple-to-understand term language of "theory expression combinators", along with multiple (categorical) semantics. We have shown that these fit our requirements of allowing to capture mathematical structure, while also allowing this structure to be hidden from users.

The design was firmly driven by its main application: to build a large library of mathematical theories, while capturing the inherent structure known to be present in such a development. To reflect mathematical practice, it is crucial to *take names seriously.* This leads us to ensuring that renamings are not only allowed, but fundamental. Categorical semantics and the desire to capture structure inexorably lead us towards considering *theory morphisms*, aka arrows of our categories of interest, as the primary notion of study — even though our original goal was grounded in the theories themselves. We believe that quite a lot of the related work would have been more successful had they focused on the morphisms rather than the theories; even current work in this domain, which pays lip service to morphisms, is very theory-centric.

Paying close attention to the "conventional wisdom" of category theory led to taking both cartesian liftings and mediating arrows as important concepts. Doing so immediately improved the compositionality of our combinators. Noticing that this puts the focus on fibrations was also helpful. Unfortunately, taking names seriously means that the fibrations are not cloven; we turn this into an opportunity for users to retain control of their names, rather than to force some kind of "naming policy".

A careful reader will have noticed that our combinators are "external", in the sense that they take and produce theories (or morphisms or ...). Many current systems use "internal" combinators, such as `include`, potentially with post facto qualifiers (such as `ocaml`'s `with` for signatures) to "glue" together items that would have been identified in a setting where morphisms, rather than theories, are primary. Furthermore, we are unaware of any system that guarantees that their equivalent to our combine is *commutative*(Proposition 4.1). Lastly, this enables future features, such as limits of diagrams, rather then just just binary combinations/mixins.

A prototype implementation exists, which was used to capture the knowledge for most of the theories on Jipsen's list [Jip] as well as many others from Wikipedia, most of the modal logics on Halleck's list [Hal], as well as two formalizations of basic category theory, once dependently-typed, and another following Lawvere's ETCS approach as presented on the nLab [nLa18]. Totally slightly over 1000 theories in slightly over 2000 lines of code, this demonstrates that our combinators, coupled with the *tiny theories* approach, does seem to work.

Even more promising, our use of very standard categorical constructions points the way to simple generalizations which should allow us to capture even more structure, without having to rewrite our library. Furthermore, as we are independent of the details of the type theory, this structure seems very robust, and our combinators should thus port easily to other systems.

## References

[Bal03] Clemens Ballarin. Locales and locale expressions in isabelle/isar. In *International Workshop on Types for Proofs and Programs*, pages 34–50. Springer, 2003.

[BG77] Rod M. Burstall and Joseph A. Goguen. Putting theories together to make specifications. In *IJCAI*, pages 1045–1058, 1977.

[BG79] Rod M. Burstall and Joseph A. Goguen. The semantics of clear, a specification language. In Dines Bjørner, editor, *Abstract Software Specifications*, volume 86 of *Lecture Notes in Computer Science*, pages 292–332. Springer, 1979.

[Bro95] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition)*. Addison-Wesley Professional, anniversary edition, August 1995.

[BS81] S. Burris and H.P. Sankappanavar. *A course in universal algebra*. Graduate texts in mathematics. Springer-Verlag, 1981. Available free at `http://www.math.uwaterloo.ca/~snburris/htdocs/ualg.html`.

[BW90] Michael Barr and Charles Wells. *Category theory for computing science*, volume 49. Prentice Hall New York, 1990.

[Car86] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209 – 243, 1986.

[CF08] J. Carette and W. M. Farmer. High-level theories. In A. et al. Autexier, editor, *Intelligent Computer Mathematics*, volume 5144 of *Lecture Notes in Computer Science*, pages 232–245. Springer-Verlag, 2008.

[CFJ+11] Jacques Carette, William M. Farmer, F. Jeremic, V. Maccio, R. O'Connor, and Q.M. Tran. The mathscheme library: Some preliminary experiments. Technical report, University of Bologna, Italy, 2011. UBLCS-2011-04.

[CK11] Jacques Carette and Oleg Kiselyov. Multi-stage programming with functors and monads: Eliminating abstraction overhead from generic code. *Sci. Comput. Program.*, 76(5):349–375, 2011.

[CO12] Jacques Carette and Russell O'Connor. Theory presentation combinators. In Johan Jeuring, John a. Campbell, Jacques Carette, Gabriel Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics*, volume 7362 of *Lecture Notes in Computer Science*, pages 202–215. Springer Berlin Heidelberg, 2012.

[CoF04] CoFI (The Common Framework Initiative). CASL *Reference Manual*. LNCS Vol. 2960 (IFIP Series). Springer-Verlag, 2004.

[DHS09] Stefania Dumbrava, Fulya Horozal, and Kristina Sojakova. A case study on formalizing algebra in a module system. In *Proceedings of the 1st Workshop on Modules and Libraries for Proof Assistants*, MLPA '09, pages 11–18, New York, NY, USA, 2009. ACM.

[dMKA+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, pages 378–388, Cham, 2015. Springer International Publishing.

[FGT92] W. M. Farmer, J. D. Guttman, and F. J. Thayer. Little theories. In *CADE-11: Proceedings of the 11th International Conference on Automated Deduction*, pages 567–581, London, UK, 1992. Springer-Verlag.

[GMdP+07] Joseph A. Goguen, Till Mossakowski, Valeria de Paiva, Florian Rabe, and Lutz Schröder. An institutional view on categorical logic. *Int. J. Software and Informatics*, 1(1):129–152, 2007.

[GS10] Adam Grabowski and Christoph Schwarzweller. On duplication in mathematical repositories. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick Ion, Laurence Rideau, Renaud Rioboo, and Alan

Sexton, editors, *Intelligent Computer Mathematics*, volume 6167 of *Lecture Notes in Computer Science*, pages 300–314. Springer Berlin / Heidelberg, 2010.

[Hal] John Halleck. Logic system interrelationships. Accessed: December 14, 2018.

[HMM89] Robert Harper, John C Mitchell, and Eugenio Moggi. Higher-order modules and the phase distinction. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 341–354. ACM, 1989.

[HST94] Robert Harper, Donald Sannella, and Andrzej Tarlecki. Structured theory presentations and logic representations. *Annals of Pure and Applied Logic*, 67(1):113 – 160, 1994.

[Jac99] B. Jacobs. *Categorical Logic and Type Theory.* Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.

[Jip] Peter Jipsen. List of mathematical structures. Accessed: December 14, 2018.

[JS92] Richard D Jenks and Robert S Sutor. *Axiom: the scientific computation system.* Springer, 1992.

[KM92] D. Kapur and D. Musser. Tecton: a framework for specifying and verifying generic system components. Technical report, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, New York, 1992. RPI-92-20.

[KMS82] Deepak Kapur, David R. Musser, and Alexander A. Stepanov. Tecton: A language for manipulating generic objects. In *Program Specification, Proceedings of a Workshop*, pages 402–414, London, UK, UK, 1982. Springer-Verlag.

[Law04] William F. Lawvere. Functorial Semantics of Algebraic Theories. *Reprints in Theory and Applications of Categories*, 4:1–121, 2004.

[Mog89] Eugenio Moggi. A category-theoretic account of program modules. In *Category Theory and Computer Science*, pages 101–117, London, UK, UK, 1989. Springer-Verlag.

[nLa18] nLab authors. fully formal ETCS. `http://ncatlab.org/nlab/show/fully%20formal%20ETCS`, December 2018. Revision 31.

[Ori00] Catherine Oriat. Detecting equivalence of modular specifications with categorical diagrams. *Theor. Comput. Sci.*, 247(1-2):141–190, 2000.

[Pit00] A. M. Pitts. Categorical logic. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, pages 39–128. Oxford University Press, 2000.

[RK13] Florian Rabe and Michael Kohlhase. A scalable module system. *Inf. Comput.*, 230:1–54, September 2013.

[Shu97] Sherri J. Shulman. *A Meta-theory for Structured Presentations in the COC*. PhD thesis, Oregon Graduate Institute of Science & Technology, Beaverton, OR, USA, 1997. UMI Order No. GAX97-24794.

[Smi93] Douglas R. Smith. Constructing specification morphisms. *Journal of Symbolic Computation*, 15:5–6, 1993.

[Smi99] Douglas R. Smith. Mechanizing the development of software. In M. Broy and R. Steinbrueggen, editors, *Calculational System Design, Proceedings of the NATO Advanced Study Institute*, pages 251–292. IOS Press, Amsterdam, 1999.

[SvdW11] Bas Spitters and Eelis van der Weegen. Type classes for mathematics in type theory. *Mathematical Structures in Computer Science*, 21(4):795–825, 2011.

[Tay99] P. Taylor. *Practical Foundations of Mathematics*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1999.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[Vel07] Todd L. Veldhuizen. Parsimony principles for software components and metalanguages. In Charles Consel and Julia L. Lawall, editors, *GPCE*, pages 115–122. ACM, 2007.

[Whi98] A.N. Whitehead. *A treatise on universal algebra: with applications*. Number v. 1 in Cornell University Library historical math monographs. The University Press, 1898.

Received Month Year; revised Month Year; accepted Month Year