A Formalization of Operads in Coq

Zachary Flores ✉
High Assurance Soluions, Two Six Technologies

Angelo Taranto ✉
High Assurance Solutions, Two Six Technologies

Eric Bond ✉
High Assurance Solutions, Two Six Technologies

Yakir Forman ✉
High Assurance Solutions, Two Six Technologies

──── **Abstract** ────

What provides the highest level of assurance for correctness of execution within a programming language? One answer, and our solution in particular, to this problem is to provide a formalization for, if it exists, the denotational semantics of a programming language. Achieving such a formalization provides a gold standard for ensuring a programming language is correct-by-construction. In our effort on the DARPA V-SPELLS program, we worked to provide a foundation for the denotational semantics of a meta-language using a mathematical object known as an operad. This object has compositional properties which are vital to building languages from smaller pieces. In this paper, we discuss our formalization of an operad in the proof assistant Coq. Moreover, our definition within Coq is capable of providing proofs that objects specified within Coq are operads. This work within Coq provides a formal mathematical basis for our meta-language development within V-SPELLS. Our work also provides, to our knowledge, the first known formalization of operads within a proof assistant that has significant automation, as well as a model that can be replicated without knowledge of Homotopy Type Theory.

1 Introduction

The DARPA V-SPELLS (Verified Security and Performance Enhancement of Large Legacy Software) program aims to create developer-accessible capability for piece-by-piece enhancement of software components for large legacy codebases with new verified code that is *safely composable* with the rest of the system.

In our approach with the Johns Hopkins Applied Physics Laboratory to solving the problems posed by V-SPELLS, our tool in development, called LUMOS, begins by applying

methods from static analysis, natural language processing, and dynamic analysis to the legacy source code in order to generate high-level abstractions of these DSLs (Domain Specific Languages) that we call domain-specific semantic models (DSSMs) from the DSLs that comprise the source code. These DSSMs will be generated in a language we refer to as the *meta-DSL*, and in order to provide the patches to the legacy code requested in V-SPELLS, these DSSMs will have to be composed in very specific ways. In order to ensure correctness of composition, as is required in V-SPELLS, we are providing verification via an algebraic framework using several ideas from category theory in which the key structure to our modeling is called an *operad*. Operads have begun to play an increasingly important role within applied mathematics (see [6, 3, 1, 2, 4]), and we find they provide an excellent mathematical model for our verification needs on V-SPELLS.

To be more precise about our modeling, when a DSSM is written in the meta-DSL, we will use an operad to represent the DSSM in the meta-DSL, and composition of DSSMs within the meta-DSL will be modeled via a "gluing" operation. Mathematically, we are providing denotational semantics for a key portion of the language of the meta-DSL. To ensure the highest level of correctness on composition between DSSMs in the meta-DSL, we aim provide a formalization of the denotational semantics of the meta-DSL. In particular, we need to provide a formalization for the foundation for the denotational semantics of the meta-DSL: operads. We provide this formalization within the proof assistant Coq, and this is the focus of our paper.

In Section 2, we discuss the informal definition of operads; Section 3 discusses the technicalities we faced and our solutions to defining operads within Coq; in Section 4, we discuss our construction of the equivalent of an *operad of sets* within Coq (namely, an *operad of types*), and discuss our proof in Coq that this is an operad according to our specification in Section 3; and lastly, in Section 5, we compare our formalization to the only other formalization of operads we are aware of [5]. We

2   Informally Defining Operads

While there does not seem to be an agreed-upon definition for a *symmetric colored operad*, we note we are following the definition of a *symmetric colored operad* in [7]. However, we remark the definition in [7] does not include what is called the *equivariance axiom* in [8]; we too omit this axiom, since it is not relevant to what we want to accomplish in our work on V-SPELLS. Regardless of these distinctions, we use *operad* to mean *symmetric colored operad* or *colored operad* in the sequel.

As our aim was to fully formalize the definition of an operad within Coq, we require precision, so we provide the full informal definition of an operad below in two parts. The first part consists of the objects that comprise an operad.

▶ **Definition 1** (Data for an Operad)**.** *An **operad**, $\mathcal{O}$, consists of a collection of types, which we will denote by T, and for each $n \geq 1$, $d \in T, \underline{c} := c_0, \ldots, c_{n-1}$ a sequence of types in T, a collection of terms $\mathcal{O}\binom{d}{\underline{c}}$ such that,*

1. *for each $c \in T$, we designate an element $\mathbb{1}_c \in \mathcal{O}\binom{c}{c}$ called the c-**colored unit**;*
2. *if σ is a permutation on n letters, and $\underline{c}\sigma := c_{\sigma(0)}, \ldots, c_{\sigma(n-1)}$, then there is a bijection between $\mathcal{O}\binom{d}{\underline{c}}$ and $\mathcal{O}\binom{d}{\underline{c}\sigma}$;*

---

3. *for any sequence $\underline{b}$ of types in T, if we denote by $\underline{c} \bullet_i \underline{b}$ the sequence given by*

$$\underbrace{c_0, \ldots, c_{i-1}}_{\emptyset \ if \ i = 0}, \underline{b}, \underbrace{c_{i+1}, \ldots, c_{n-1}}_{\emptyset \ if \ i = n-1},$$

then we require the existence of a function:

$$\circ_i : \mathcal{O}\binom{d}{\underline{c}} \times \mathcal{O}\binom{c_i}{\underline{b}} \to \mathcal{O}\binom{d}{\underline{c} \bullet_i \underline{b}}.$$

*We typically refer to the function $\circ_i$ as **multi-composition**.*

▶ **Example 2.** For quick example of what the type signature of a multi-composition function looks like, let $\underline{c} = c_0, c_1, c_2$, $\underline{b} = b_0, b_1$, and $i = 1$, then $\circ_1$ has type signature:

$$\mathcal{O}\binom{d}{c_0, c_1, c_2} \times \mathcal{O}\binom{c_1}{b_0, b_1} \to \mathcal{O}\binom{d}{c_0, b_0, b_1, c_2}.$$

Now the data for an operad $\mathcal{O}$ in Definition 1 is subject to certain axiomatic constraints, and this forms the second half of our definition for an operad.

▶ **Definition 3** (Axioms for an Operad). *Let $\underline{c} := c_0, \ldots, c_{n-1}, \underline{b} := b_0, \ldots, b_{m-1}, \underline{a} = a_0, \ldots, a_{\ell-1}$ be sequences from a collection of types T. The axioms that the data for an operad $\mathcal{O}$ must follow are given below.*

1. *The **horizontal associativity axiom**: Suppose $n \geq 2$ and $0 \leq i < j \leq n-1$, then for $(\alpha, \beta, \gamma) \in \mathcal{O}\binom{d}{\underline{c}} \times \mathcal{O}\binom{c_i}{\underline{a}} \times \mathcal{O}\binom{c_j}{\underline{b}}$,*

$$(\alpha \circ_i \beta) \circ_{\ell-1+j} \gamma = (\alpha \circ_j \gamma) \circ_i \beta$$

To give a visual description of this axiom, we are requiring commutativity of the following diagram:

$$
\begin{array}{ccc}
\mathcal{O}\binom{d}{\underline{c}} \times \mathcal{O}\binom{c_i}{\underline{a}} \times \mathcal{O}\binom{c_j}{\underline{b}} & \xrightarrow{(\circ_i, id)} & \mathcal{O}\binom{d}{\underline{c} \bullet_i \underline{a}} \times \mathcal{O}\binom{c_j}{\underline{b}} \\
\cong \downarrow swap & & \downarrow \circ_{\ell-1+j} \\
\mathcal{O}\binom{d}{\underline{c}} \times \mathcal{O}\binom{c_j}{\underline{b}} \times \mathcal{O}\binom{c_i}{\underline{a}} & & \mathcal{O}\binom{d}{(\underline{c} \bullet_i \underline{a}) \bullet_{\ell-1+j} \underline{b}} \\
(\circ_j, id) \downarrow & & \| \\
\mathcal{O}\binom{d}{\underline{c} \bullet_j \underline{b}} \times \mathcal{O}\binom{c_i}{\underline{a}} & \xrightarrow{\circ_i} & \mathcal{O}\binom{d}{(\underline{c} \bullet_j \underline{b}) \bullet_i \underline{a}}.
\end{array}
$$

2. *The **vertical associativity axiom**: Suppose $m, n \geq 1$, $0 \leq i \leq n-1$, and $0 \leq j \leq m-1$. Then for $(\alpha, \beta, \gamma) \in \mathcal{O}\binom{d}{\underline{c}} \times \mathcal{O}\binom{c_i}{\underline{b}} \times \mathcal{O}\binom{b_j}{\underline{a}}$,*

$$(\alpha \circ_i \beta) \circ_{i+j} \gamma = \alpha \circ_i (\beta \circ_j \gamma)$$

That is, we are requiring commutativity of the following diagram:

$$\mathcal{O}\binom{d}{\underline{c}} \times \mathcal{O}\binom{c_i}{\underline{b}} \times \mathcal{O}\binom{b_j}{\underline{a}} \xrightarrow{(id, \circ_j)} \mathcal{O}\binom{d}{\underline{c}} \times \mathcal{O}\binom{c_i}{\underline{b} \bullet_j \underline{a}}$$

$$(\circ_i, id) \downarrow \qquad\qquad\qquad\qquad \downarrow \circ_i$$

$$\mathcal{O}\binom{d}{c \bullet_i (\underline{b} \bullet_j \underline{a})}$$

$$\|$$

$$\mathcal{O}\binom{d}{\underline{c} \bullet_i \underline{b}} \times \mathcal{O}\binom{b_j}{\underline{a}} \xrightarrow{\circ_{i+j}} \mathcal{O}\binom{d}{(\underline{c} \bullet_i \underline{b}) \bullet_{i+j} \underline{a}}$$

3. The **left unity axiom** requires that for $\alpha \in \mathcal{O}\binom{d}{\underline{c}}$ with $n \geq 1$, $\mathbb{1}_d \circ_1 \alpha = \alpha$.

4. The **right unity axiom** requires that for $n \geq 1$, $0 \leq i \leq n-1$, and $\alpha \in \mathcal{O}\binom{d}{\underline{c}}$, $\alpha \circ_i \mathbb{1}_{c_i} = \alpha$.

Before we give an example, some comments are in order about Definition 3.

▶ Remark 4.
We want to give some sanity checks of the associativity axioms. First notice the following equality occurs in the right-hand corner of the diagram for the horizontal associativity axiom (1 of Definition 3):

$$\mathcal{O}\binom{d}{(\underline{c} \bullet_i \underline{a}) \bullet_{\ell-1+j} \underline{b}} = \mathcal{O}\binom{d}{(\underline{c} \bullet_j \underline{b}) \bullet_i \underline{a}} \tag{1}$$

This equality arises from an equality of the following sequences:

$$
\begin{aligned}
(\underline{c} \bullet_i \underline{a}) \bullet_{\ell-1+j} \underline{b} &= (c_0, \ldots, c_{i-1}, \underline{a}, c_{i+1}, \ldots, c_{n-1}) \bullet_{\ell-1+j} \underline{b} \\
&= c_0, \ldots, c_{i-1}, \underline{a}, c_{i+1}, \ldots c_{j-1}, \underline{b}, c_{j+1}, \ldots, c_{n-1} \\
&= (\underline{c} \bullet_j \underline{b}) \bullet_i \underline{a}
\end{aligned}
\tag{2}
$$

In particular, in providing a specification in Coq for operads, we need to provide a proof that (2) holds for such sequences in T.

A similar equality of sequences is required to define the vertical associativity diagram:

$$
\begin{aligned}
\underline{c} \bullet_i (\underline{b} \bullet_j \underline{a}) &= c_0, \ldots, c_{i-1}, (\underline{b} \bullet_j \underline{a}), c_{i+1}, \ldots, c_{n-1} \\
&= c_0, \ldots, c_{i-1}, b_0, \ldots, b_{j-1}, \underline{a}, b_{j+1}, \ldots, b_{m-1}, c_{i+1}, \ldots, c_{n-1} \\
&= (\underline{c} \bullet_i \underline{b}) \bullet_{i+j} \underline{a}
\end{aligned}
\tag{3}
$$

While our definition seems extraordinarily abstract, the next example helps clarify the roots of the abstraction found in Definition 1 and Definition 3. Moreover, the next example will serve as the first application of our formal definition of operads, as we will prove in Coq that our realization of this example is an operad according to our specification.

▶ **Example 5.** If we let T be a collection of types for which T is closed under finite products, we can define an operad $\mathbf{Sets}_T$ by setting

$$\mathbf{Sets}_T \begin{pmatrix} d \\ c_0, \ldots, c_{n-1} \end{pmatrix} := \mathrm{Hom}(c_0 \times \cdots \times c_{n-1}, d),$$

where the hom-set on the right is the collection of all functions from the product of sets $c_0 \times \cdots \times c_{n-1}$ to the set d. Given $c \in T$, the identity function on c operates as the c-colored unit in $\mathbf{Sets}_T\binom{c}{c} = \mathrm{Hom}(c, c)$. In this setting, we can explicitly define multi-composition $\circ_i$ from Definition 1 which returns, given $f \in \mathrm{Hom}(c_0 \times \cdots \times c_{n-1}, d)$ and $g \in \mathrm{Hom}(b_0 \times \cdots \times b_{m-1}, c_i)$, the function $f \circ_i g$ which acts on the $(n + m - 1)$-tuple $(x_0, \ldots, x_{i-1}, \underline{y}, x_{i+1}, \ldots, x_{n-1})$ as

$$(f \circ_i g)(x_0, \ldots, x_{i-1}, \underline{y}, x_{i+1}, \ldots, x_{n-1}) = f(x_0, \ldots, x_{i-1}, g(\underline{y}), x_{i+1}, \ldots, x_{n-1}).$$

All other pieces of Definition 1 and 3 not mentioned above can be proved for $\mathbf{Sets}_T$ using everything defined above and basic facts in set theory.

3 Formally Modeling Operads in Coq

In defining the collection of terms $\mathcal{O}\binom{d}{c_0, \ldots, c_{n-1}}$ in Coq, Definition 1 requires that d, c_i come from the collection T. Throughout our specification in this paper, we will replace T with one of Coq's in-house universes: **Type**. In practice, we do need a proper subset of **Type**, but for simplicity in our paper, we use **Type**. In the event we need a restriction to a subset of **Type**, we briefly discuss how to use Tarski universes to do this after the description of our formal model in Coq.

3.1 Encoding an Operad in Coq

The first goal to tackle in defining an operad is giving a formal definition of $\mathcal{O}\binom{d}{c}$.

▶ **Note 6** (A Definition for $\mathcal{O}\binom{d}{c}$ in Coq). Informally, part of an operad $\mathcal{O}$ is a collection of sets indexed by pair $d :$ **Type** and $\underline{c} := c_0, \ldots, c_{n-1} :$ **list Type**. Since this is a *collection of sets*, it would be natural to use a *record* in Coq to make this definition. To do so, we create a record in Coq, which we denote as **Operad**, whose single field is given by a function with type signature: **Type** $\to$ **list Type** $\to$ **Type**. An instantiation of **Operad** will yield a function $\mathcal{O} :$ **list Type** $\to$ **Type** $\to$ **Type**, so that $\mathcal{O}\binom{d}{c}$ yields our desired collection of terms.

We give an example of our definition from Note 6.

▶ **Example 7.** Our goal in Section 4 is to provide a version of $\mathbf{Sets}_T$ in Example 5 in Coq for $T = $ **Type**; we will denote this operad by **Type**. In Coq, if $\underline{c} = c_0, \ldots, c_{n-1} :$ **list Type** and $d :$ **Type**, then the following is definable in Coq via recursion:

$$\mathbf{Type}\begin{pmatrix} d \\ \underline{c} \end{pmatrix} := c_0 \to \cdots \to c_{n-1} \to d.$$

In particular, *terms* of type $\mathbf{Type}\binom{d}{c}$ are n-ary functions with codomain defined by $\underline{c}$, and with return type d.

In the rest of our model in Coq, we also use a record to denote the data that comprises the operad (as in Definition 1) or the constraints the data is subject to (as in Definition 3). Each piece in Definition 1 and 3 is a proposition that must be satisfied. We first detail how the the data from Definition 1 will be encoded as propositions within Coq.

▶ Note 8 (Data for an Operad in Coq). **1.** the existence of a c-colored unit in $\mathcal{O}$ (1 from Definition 1): for all $c : \textbf{Type}$, there is a $\mathbb{1}_c \in \mathcal{O}\binom{c}{c}$;

2. the requirement that there is a bijection between $\mathcal{O}\binom{d}{\underline{c}}$ and $\mathcal{O}\binom{d}{\underline{c}\sigma}$ for a permutation σ on n letters (2 from Definition 1): for all $d : \textbf{Type}$, $\underline{c}, \underline{c}' : \textbf{list Type}$ with the length $\underline{c}$ at least 1, and $\underline{c}$ and $\underline{c}'$ are permutations of one another, there is a bijection between $\mathcal{O}\binom{d}{\underline{c}}$ and $\mathcal{O}\binom{d}{\underline{c}'}$;

3. the requirement for the existence of $\circ_i$ (3 from Definition 1); for all $i, n : \mathbb{N}$, $d, c_i : \textbf{Type}$, $\underline{b}, \underline{c} : \textbf{list Type}$, if $\underline{c}$ has length n, $1 \leq n$, $i < n$, and the nth entry of $\underline{c}$ is c_i, there is a function of type $\mathcal{O}\binom{d}{\underline{c}} \times \mathcal{O}\binom{c_i}{\underline{b}} \to \mathcal{O}\binom{d}{\underline{c}\bullet_i\underline{b}}$.

To make our implementation in Coq clear in Note 8, some remarks are in order about how to make the above precise within Coq:

▶ Remark 9. **1.** Any time *bijection* is used in this context, we are referring to a bijection in **Type**. That is, if $t, t' : \textbf{Type}$, then there are functions $f : t \to t'$, $f' : t' \to t$ such that $f \circ f' = \text{id}_{t'}$, and $f' \circ f = \text{id}_t$. This is easily definable in Coq.

2. To create a proposition that two lists, $\underline{c}, \underline{c}'$, are permutations of one another in Coq, we can use Coq's built-in type **Permutation**. This says that **Permutation** $\underline{c}\, \underline{c}' : \textbf{Prop}$ (where **Prop** is the type of all propositions in Coq).

3. The operation $\bullet_i$ on lists can be defined in Coq by taking the first i entries of $\underline{c}$, concatenating the list $\underline{b}$, and then concatenating the the last $n - i - 1$ entries of $\underline{c}$ to the previous concatenation.

4. In 3 of Note 8, we need the use of the nth function within Coq. This function requires a default element as part of its arguments, which means we would need to choose a default element from **Type** to use consistently throughout. The choice we make in Coq is the *unit* type, which is the type used to represent singleton sets.

We encode Definition 3 into Coq in a similar manner using records, denoting this record by **OperadLaws**. However, there is more caution to be had, due in part to the discussion in Remark 4. To demonstrate this caution, we discuss our modeling of of the horizontal associativity axiom within Coq in explicit detail below.

▶ Note 10 (Axioms for an Operad in Coq). The horizontal associativity axiom in an operad (1 in Definition 3) can be defined in Coq by first listing a collection of parameters that we refer to as P:

- $n, m, \ell, i, j : \mathbb{N}$;
- $d, c_i, c_j : \textbf{Type}$;
- $\underline{a}, \underline{b}, \underline{c} : \textbf{list Type}$
- $\alpha : \mathcal{O}\binom{d}{\underline{c}}, \beta : \mathcal{O}\binom{c_i}{\underline{b}}, \gamma : \mathcal{O}\binom{b_j}{\underline{a}}$
- $2 \leq n$, $1 \leq m$, and $1 \leq \ell$;
- $i < j$ and $j < n$;
- $\underline{c}$ has length n, $\underline{b}$ has length m, and $\underline{a}$ has length ℓ;

Distribution Statement A: Approved for Public Release, Distribution Unlimited

- the ith entry of $\underline{c}$ is c_i and the jth entry of $\underline{c}$ is c_j;

Using what is now in P, we can give a proof that the ith entry of $\underline{c} \bullet_j \underline{b}$ is c_i, and a proof that the $(\ell - 1 + j)$th entry of $\underline{c} \bullet_i \underline{a}$ is c_j; we add these proofs to P. With this update to P, we can state our formalization of the horizontal associativity axiom in Coq: for all parameters that comprise P, Equation (2) in Remark 4 holds, and there exists a type casting function $\mathcal{C}_{\mathbf{assoc}}$ such that

$$\mathcal{C}_{\mathbf{assoc}} \, P \left((\alpha \circ_i \beta) \circ_{\ell-1+j} \gamma \right) = (\alpha \circ_j \gamma) \circ_i \beta$$

The type-casting function $\mathcal{C}_{\mathbf{assoc}}$ is necessary, since we have defined in Coq for each $d : \mathbf{Type}$ and $\underline{c} : \mathbf{list} \ \mathbf{Type}$, that $\mathcal{O}\binom{d}{\underline{c}}$ be a type in $\mathbf{Type}$, and the casting function provides a proof that the equality of types in Equation (1) holds. However, the existence of $\mathcal{C}_{\mathbf{assoc}}$ relies entirely on the proof of the equality of lists in Equation (2). Now the equality in Equation (2) requires a significant effort, and the most difficult part of formalizing this axiom is in providing its proof.

Providing a formal specification of all other axioms in Definition 3 to be inserted into the fields of of our record **OperadLaws** follows the same path as above:

1. carefully curate the correct collection P of parameters needed for the axiom;
2. add in any proofs needed that can be deduced from everything currently in P;
3. show that any required equality of lists holds (this will be necessary for all axioms in Definition 3);
4. create the necessary casting function.

We have one last comment to make on the choices in our model.

▶ Remark 11. In [8] the definition for operads says that if $\underline{c} = \emptyset$, the empty list of symbols coming from the collection T, then the symbol $\mathcal{O}\binom{d}{\emptyset}$ still has meaning. Notice in Definition 1, we do not allow the existence of of such a symbol since we require that the list $\underline{c}$ is *not* empty. Our reason for doing so is that our main application relies on giving a version of Example 5 in Coq. Within $\mathbf{Sets}_T$, if $\underline{c} = \emptyset$, then the product of an empty list of sets is a singleton, $\{\bullet\}$, so that $\mathbf{Sets}_T\binom{d}{\emptyset} \cong \mathbf{Sets}_T\binom{d}{\{\bullet\}}$. We can model this situation in Coq by letting $\underline{c}$ be the list whose only entry is $\mathbb{U} : \mathbf{Type}$, the unit type.

3.2 Tarski Universes

A solution to using a subset T of $\mathbf{Type}$ is to define T in Coq as a *Tarski universe*. This defines $T : \mathbf{Type}$, as well as an *interpretation* that allows the terms of T be regarded as *codes* for actual types. In this way, the type T is a set together with an injective mapping to $\mathbf{Type}$, which is exactly the data of a subset of $\mathbf{Type}$. Our approach to implementing this definition in Coq involves the following:

1. a type $\mathcal{B}$ in Coq with nullary constructors, we call the *base types*, and whose terms we refer to as *type sigils*;
2. the constructors that define the type T, which include:
   - a constructor with signature $\mathbf{Ty} : \mathcal{B} \to T$ which encodes the base types into T;

- other constructors that may model products, such as $\mathbf{p} : T \to T \to T$, or $\mathbf{fn} : T \to T \to T$, which can model functions;

3. an assignment for $\mathcal{B}$ within **Type**, and a recursively-defined interpretation function $\mathbf{El} : T \to \mathbf{Type}$ that assigns a value within **Type** to each $t : T$.

We give an example of what this would look like explicitly.

▶ **Example 12.** We define our collection of base types $\mathcal{B}$ in Coq with the nullary constructors N, U, and B. Within Coq, we create a function $\mathbf{I}$ that interprets these type sigils: N is assigned to $\mathbb{N}$, the type of natural numbers; U to $\mathbb{U}$, the unit type; B to $\mathbb{B}$, which is bool.

If we want to model products and functions within in T, then we can define T with constructors:

1. $\mathbf{Ty} : \mathcal{B} \to T$;
2. $\mathbf{p} : T \to T \to T$;
3. $\mathbf{fn} : T \to T \to T$.

Now $\mathbf{El}$ will provide the embedding into Coq via the following recursion:

1. $\mathbf{El}\,(\mathbf{Ty}\,t) \Rightarrow \mathbf{I}\,t$
2. $\mathbf{El}\,(\mathbf{p}\,t\,t') \Rightarrow \mathbf{El}\,t \times \mathbf{El}\,t'$
3. $\mathbf{El}\,(\mathbf{fn}\,t\,t') \Rightarrow \mathbf{El}\,t \to \mathbf{El}\,t'$

For an explicit example of a code in T, we have $\mathbf{p}\,(\mathbf{Ty}\,N)(\mathbf{Ty}\,N) : T$, and via the embedding $\mathbf{El}$, this is a model for $\mathbb{N} \times \mathbb{N}$ in Coq.

4 A Proof Using Our Model

Our goal in this section is to discuss the formal proof that the equivalent of Example 5 in Coq, which we denote by **Type** and define in Example 7, is an operad according to our model.

To formally demonstrate that **Type** is an operad, we first need a definition of the function in the only field of the record **Operad** (see Note 6). Next, in the record **OperadLaws**, we need to define, for $c :$ **Type**, the c-colored units (1 of Definition 1), provide proofs that $\mathbf{Type}\binom{d}{\underline{c}}$ is invariant (up to injection) under reordering of $\underline{c}$ (2 of Definition 1), define the multi-composition functions (3 of Definition 1), and show all axioms in Definition 3 hold according. Wrapping these assignments and proofs together provides a term of type **Operad** and **OperadLaws**, which gives our desired formal proof.

In Example 7, we give the definition of the required function in **Operad** for **Type**: given $\underline{c} = c_0, \ldots, c_{n-1} :$ **list Type** and $d :$ **Type**, we write:

$$\mathbf{Type}\binom{d}{\underline{c}} := c_0 \to \cdots \to c_{n-1} \to d,$$

which is the type of n-ary functions with codomain defined by $\underline{c}$ and return type d. Our instantiation of **OperadLaws** for **Type** will use this definition throughout.

The right-hand side of $\mathbf{Type}\binom{d}{\underline{c}}$ is defined via a recursive function, which we denote as **arr** (short for *arrow*), with type signature **list Type** $\to$ **Type** $\to$ **Type**. In particular, we define $\mathbf{arr}\,\emptyset\,d = d$ (where $\emptyset$ is the empty list).

4.1 Implementing the Data for Type in Coq

Next we discuss in a series of notes, the implementation of Definition 1 for **Type** in Coq, as well as the tools that were developed for use in this implementation.

▶ Note 13 (c-colored units in **Type**). If $\underline{c}$ has single entry $c :$ **Type**, then $\mathbf{Type}\binom{c}{c} = c \to c$, which is the type of all functions with domain and range given by c. Then $\mathbb{1}_c := \mathrm{id}_c$, the identity function on c.

▶ Note 14 ($\mathbf{Type}\binom{d}{c} \cong \mathbf{Type}\binom{d}{c\sigma}$). Our motivation is to provide the equivalent of Example 5 within Coq, and the analogous isomorphism for the operad $\mathbf{Sets}_T$ is,

$$\mathrm{Hom}(c_0 \times \cdots \times c_{n-1}, d) \cong \mathrm{Hom}(c_{\sigma(0)} \times \cdots \times c_{\sigma(n-1)}, d),$$

given $\underline{c} = c_0, \ldots, c_{n-1}$ and a permutation σ on n letters. The isomorphism in the context of Coq asks us to construct a bijection between the two sets above. Following Definition 8 and comments in Remark 9, we can translate this into Coq for **Type** as: for all $d :$ **Type**, $\underline{c}, \underline{c}' :$ **list Type** with the length $\underline{c}$ at least 1, and **Permutation** $\underline{c}\,\underline{c}'$, there is a bijection between $\mathbf{Type}\binom{d}{c}, \mathbf{Type}\binom{d}{c'} :$ **Type**.

We can prove this in Coq using induction on the length of $\underline{c}$, along with some preceding lemmas about the behavior of function composition and isomorphism in this setting.

▶ Note 15 ($\circ_i$ for **Type**). Lastly, we need to write (3) of Definition 8 for **Type** in Coq. Writing $\mathbf{Type}\binom{d}{c}$ in the curried form, as opposed to using a verbatim translation of $\mathbf{Sets}_T\binom{d}{c}$ from Example 5, provides the needed flexibility, via partial application, to implement the multi-composition function $\circ_i$ for **Type** in Coq with respect to (3) of Definition 1. The most important piece of our definition in Coq is that we define a recursive function **compose** with type signature

$$\mathbf{arr}\,\underline{c}'\,(t \to t') \to \mathbf{arr}\,\underline{b}\,t \to \mathbf{arr}\,\underline{c}'\,(\mathbf{arr}\,\underline{b}\,t');$$

where $t, t' :$ **Type**, and $\underline{c}', \underline{b} :$ **list Type**. If $\underline{c} = c_0, \ldots, c_{i-1}, c_i, c_{i+1}, \ldots, c_{n-1}$, we let $\underline{c}' = c_0, \ldots, c_{i-1}$, $t = c_i$, and $t' = c_i \to c_{i+1} \to \cdots \to c_{n-1} \to d$, and this gives us, provided the correct type inference is written in, the required multi-composition operator $\circ_i$ for **Type**.

4.2 Implementing the Axioms for Type in Coq

Definition 1 provides the base to build the axioms of an operad, which are given in Definition 3, and we discuss our implementation in Coq of this here. There are several axioms listed in Definition 3, and as in Note 10, we keep our discussion focused on the horizontal associativity axiom (1 of Definition 3), as the proof that **Type** satisfies all other axioms in Definition 3 follows from a similar procedure in Coq.

Our first hurdle comes from noticing that our definition for $\circ_i$ in Note 15 is what we want mathematically, but that Coq does not automatically recognize the equality of types:

$$\underline{c}'\,(\mathbf{arr}\,\underline{b}\,t') = \mathbf{arr}(\underline{c} \bullet_i \underline{b})\,d,$$

where $\underline{c} = c_0, \ldots, c_{i-1}, c_i, c_{i+1}, \ldots, c_{n-1}$, $\underline{c}' = c_0, \ldots, c_{i-1}$, $t = c_i$, $t' = c_i \to c_{i+1} \to \cdots \to c_{n-1} \to d$, so that (see 3 of Definition 1) $\underline{c} \bullet_i b = c_0, \ldots, c_{i-1}, \underline{b}, c_{i+1}, \ldots, c_{n-1}$. Since we

---

require for $f : \mathbf{Type}\binom{d}{\underline{c}}, g : \mathbf{Type}\binom{c_i}{\underline{b}}$ that $f \circ_i g : \mathbf{Type}\binom{d}{\underline{c}\bullet_i\underline{b}}$, this presents an issue whose solution is, as in Note 10, a type casting function.

The remainder of our formalization of **Type** within Coq is a tour de force of type casting, and we discuss the tools we use in this proof below. First, we give the formal definition we use for a type cast within Coq.

▶ **Definition 16.** *Given A, B : Type, and an equation, $A = B$, a **type cast** $\mathcal{C}_{A=B}$ is a function such that for $a : A$, $\mathcal{C}_{A=B}\, a : B$.*

In order to manipulate the type casts that occur throughout our proof that **Type** satisfies the axioms in Definition 3, we prove a handful of general facts about type casts in Coq, which we discuss below.

▶ Note 17 (Type Casts for Definition 3 in Coq). **1.** The composition two type casts is a type cast: given equations of types $A = B$ and $B = C$, we have $\mathcal{C}_{B=C} \circ \mathcal{C}_{A=B} = \mathcal{C}_{A=C}$.

2. A type cast using an equation of types $A = A$ (i.e., a type cast between two types Coq recognizes as identical) is equal to the identity: $\mathcal{C}_{A=A}\, a = a$ for $a : A$.

3. Two type casts between the same two types (i.e., both using equations of type $A = B$) are equal: for all $a : A$, $\mathcal{C}_{A=B}\, a = \mathcal{C}'_{A=B}\, a$.

4. Type casting a function and then applying it to an argument is the same as applying the original function to an argument that had been type cast: if $f : B \to C$ and $a : A$, then $(\mathcal{C}_{A\to B=B\to C}\, f)\, a = f\, (\mathcal{C}_{A=B}\, a)$

These facts smooth the process of showing **Type** satisfies the operad axioms of Definition 3, as this involves manipulations of several type casts. For example, 2 and 3 of Note 17 ensure that it is not necessary to keep track of how these manipulations impact the *equations* on which the type casts rely, since we need only that the *types* involved match in order to show equality.

Now we discuss how to utilize the facts we demonstrated in Note 17, by discussing their use in our proof the horizontal associativity axiom (1 of Definition 3) is satisfied in **Type**. Our first step is to prove the following key lemma of equality involving the **compose** function:

$$\mathbf{compose}\,(\,\mathcal{C}\,(\,\mathbf{compose}\,(\,\mathcal{C}\,\alpha\,)\,\beta\,)\,)\,\gamma = \ \mathcal{C}\,(\,\mathbf{compose}\,(\,\mathcal{C}\,(\,\mathbf{compose}\,(\,\mathcal{C}\,\alpha\,)\,\gamma\,)\,)\,\beta\,). \quad (4)$$

Here, α, β, γ are terms of the appropriate types in the operad **Type**, and the type casting equations have been suppressed from the notation. We can show the equality in Equation 4 by induction on the appropriate lists and several uses of 4 of Note 17. Notice that Equation 4 is essentially the horizontal associativity axiom in **Type**, and this is the case: to prove the horizontal associativity axiom, we manipulate type casts using our work in Note 17 until they match the equality in Equation 4.

Demonstrating the remainder of the axioms in Definition 3 for **Type** within Coq follows a similar pattern: show the pattern for the given axiom holds for **compose**, and then manipulate the type casts appropriately using Note 17 to arrive at the desired axiom.

---

5 Related Work

In [5], the authors present a formalization of a simpler type of an operad using Cubical Agda, which is an extension of Agda with Cubical Type Theory. Cubical Type Theory is an alternative to Homotopy type theory that is more directly amenable to constructive interpretations, so fully understanding the implementation of operads in [5] requires a working knowledge of a variant of Homotopy type theory, as well as how to use its implementation in Agda.

Moreover, Agda does not have significant automation, so showing, for example, our proof in Section 4 would require *significantly* more work. However, we do not think that our work would be impossible to translate into Agda, just require much more boilerplate code (e.g. handwriting structural induction tactics).

We also want to compare what was formalized in our work to that of [5]. What they use in [5] to refer to an operad is an operad with a collection of types T for which $|T| = 1$. In particular, $\mathcal{O}\binom{d}{\underline{c}}$ can be parametrized by the natural numbers, so we can write $\mathcal{P}(n) := \mathcal{O}\binom{d}{\underline{c}}$, if $|\underline{c}| = n$, where $\underline{c} = c_0, \ldots, c_{n-1}$, with $c_i = d$. Moreover, there is a unique identity $(1 \in \mathcal{P}(1))$, the functions $\circ_i$ have type $\mathcal{P}(n) \to \mathcal{P}(m) \to \mathcal{P}(n + m - 1)$, and there is a significant simplification of the associativity axioms in Definition 3. We also note [5] they define their singly-colored operads to have the *equivariance axiom* given in [8].

───── **References** ─────

1 John C. Baez and John Foley. Operads for designing systems of systems. *CoRR*, abs/2009.12647, 2020. URL: `https://arxiv.org/abs/2009.12647`, `arXiv:2009.12647`.

2 John C. Baez and Nina Otter. Operads and phylogenetic trees. *Theory Appl. Categ.*, 32:Paper No. 40, 1397–1453, 2017.

3 John D. Foley, Spencer Breiner, Eswaran Subrahmanian, and John M. Dusel. Operads for complex system design specification, analysis and synthesis. *CoRR*, abs/2101.11115, 2021. URL: `https://arxiv.org/abs/2101.11115`, `arXiv:2101.11115`.

4 Samuele Giraudo, Jean-Gabriel Luque, Ludovic Mignot, and Florent Nicart. Operads, quasi-orders, and regular languages. *Adv. Appl. Math.*, 75:56–93, 2016. `doi:10.1016/j.aam.2016.01.002`.

5 Brandon Hewer and Graham Hutton. Hott operads. *Symposium on Principles of Programming Languages*, 2023. URL: `http://www.cs.nott.ac.uk/~pszgmh/operads.pdf`.

6 Sophie Libkind, Andrew Baas, Evan Patterson, and James Fairbanks. Operadic modeling of dynamical systems: Mathematics and computation. *Electronic Proceedings in Theoretical Computer Science*, 372:192–206, nov 2022. `doi:10.4204/eptcs.372.14`.

7 David Spivak. The operad of wiring diagrams: Formalizing a graphical language for databases, recursion, and plug-and-play circuits. 2013. URL: `https://arxiv.org/pdf/1305.0297.pdf`, `arXiv:1305.0297v1`.

8 Donald Yau. *Operads of wiring diagrams*, volume 2192 of *Lecture Notes in Mathematics*. Springer, Cham, 2018. `doi:10.1007/978-3-319-95001-3`.