Universiteit Leiden
The Netherlands

Master Computer Science

Working Towards Category Theoretic Semantics for Separation Logic

Berend van Starkenburg

Supervisors:
Dr. H. Basold
T.F.R. Ralaivaosaona
Prof. dr. M.M. Bonsangue

MASTER THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

20/09/2023

Abstract

Separation logic has emerged as a powerful formalism for reasoning about memory management and resource sharing in complex software systems. This thesis presents a category-theoretic framework for reasoning about locality and compositionality, as it occurs in separation logic, which forms the foundation of working towards an abstract framework that fully captures the underlying logic of separation logic. The framework is based on a sheaf-theoretic model, defined by the category of sheaves over a topological space $\mathcal{L}$. The core contribution of this work lies in uncovering the semantic foundation of process interaction via shared memory, which serves as the foundation for looking at separation logic through sheaves. Additionally, we provide a clear and comprehensive construction of Kleisli objects internally to a category with pullbacks and small products, establishing their relationship with their external counterparts. Our results demonstrate that the sheaf condition of our categorical construction enables local reasoning about effects of programs on specific memory regions and the subsequent composition of these local effects to derive the overall impact on the entire memory uniquely and consistently.

Contents

1 Introduction **1**

2 Separation Logic **2**

3 Related Work **5**
- 3.1 Categorical Logic . . . 5
- 3.2 Extensions of the Frame Rule . . . 6
- 3.3 Sheaves . . . 6
- 3.4 preliminaries . . . 6

4 Local State Monad **7**
- 4.1 Binary Product Functor . . . 9
- 4.2 Global State Monad . . . 10
- 4.3 Summary of Notation and Definitions . . . 16
- 4.4 Local State Monad . . . 18
- 4.5 Proofs of Monad Structure . . . 24
- 4.6 Modelling Stateful Computation . . . 27

5 (Co-)Sheaves of Kleisli Morphisms **30**

6 Internal Categories **33**
- 6.1 General internal structures . . . 33
- 6.2 Externalisations . . . 35
- 6.3 Internally Defining the Kleisli Category . . . 39
- 6.4 The Category of Sheaves over $\mathcal{L}$ . . . 43

7 Separation Logic and Process Composition **45**

8 Conclusion and Discussion **47**
- 8.1 Future Work . . . 48

References **50**

1 Introduction

In 1996, the maiden flight of the Ariane 5 rocket, developed by the European Space Agency (ESA), ended in a catastrophic failure just 37 seconds after liftoff. The rocket veered off course, self-destructed, and was lost, resulting in a significant financial loss and a considerable setback to the space program. The primary cause of the failure was traced back to a software issue in the rocket's Inertial Reference System (IRS), which used a 64-bit floating-point number to represent horizontal velocity. However, the actual horizontal velocity during the launch exceeded the range represented by a 64-bit floating-point number [LIO]. In hindsight, this software issue could have been identified and prevented using formal methods, such as formal verification and testing techniques. Formal verification could have mathematically proven the correctness of the IRS software for the Ariane 5 rocket's specific trajectory, ensuring that critical properties, such as range limits for floating-point numbers, were respected. After this incident, the European Space Agency recognised the importance of formal methods. It significantly enhanced the application of formal verification and testing in designing and verifying critical software for future space missions. This example demonstrates how we have to rely increasingly on formal verification techniques that can guarantee the correctness of complex systems in the rapidly evolving landscape of modern technology.

The formal verification system, which will be the focus of this thesis, is separation logic. Separation logic was introduced by Reynolds and O'Hearn [ORY01] in 2001. The system builds upon Hoare logic [Hoa69] in which program specifications are represented by the triple $\{P\}C\{Q\}$, where P is the set of assertions that describe the initial state of the system and Q is the set of assertions that describe the state of the program after executing the program C. The key idea behind separation logic is that it facilitates local reasoning over programs with shared mutable data structures. One can then scale these local observations to get a more global specification of the program's behaviour. This is achieved by the separating conjunction, $*$, which asserts that P and Q hold for disjoint parts of the memory, and the frame rule that states that for the Hoare triple $\{P\}C\{Q\}$ the effect of the program C on the memory outside that region is independent of the frame's content. The frame represents the portion of the heap unaffected by the program C.

A conventional research perspective for such formal verification models would be to see how concrete instances of these models should be implemented. Problems that often arise with such a perspective is that a concrete implementation of a formal verification model is tailored to a specific programming language, formalism, or verification tool. Also, detailed modelling of program behaviours and resource management is required. These requirements lead to a lack of generality, a high complexity that is hard to manage when either the programming language becomes more feature-rich to support advanced constructs or the separation logic is extended. The specific implementation leads to certain design choices that will limit the full expressive power of separation logic and obfuscate the underlying theoretical principles of separation logic. Therefore, this thesis will consider a different perspective and investigate how we can use categorical abstractions to capture the underlying logic of separation logic in a higher-level framework. The relationship between category theory and logic has been established by Lawvere [Law69], Jacobs [Jac99], Fourman and Scott [FS79], Makkai and Reyes [MR77], and Johnstone [JJ02], among others. Their research provides insight into how categorical frameworks are useful for quickly identifying models that fall under the framework and transposing knowledge to new instances. We expect that developing such

an abstract framework for separation logic provides similar benefits found in this earlier research.

The objective of this thesis is to provide the foundation of an abstract framework that captures the semantics of separation logic. We will define the local state monad and its associated Kleisli category to model processes operating on memory. Sheaves are a mathematical abstraction that we can then use to assemble processes interacting with each other via shared memory to a more extensive system that models this interaction. A complete abstract framework that provides semantics for separation logic would be beyond the scope of a master's thesis. Such a framework should include structure to verify and specify how assertions are transformed after program execution, include logical connectives such as the separating conjunction, and categorical constructions relate extensions of separation logic to the underlying logic in a way that the semantics of this underlying logic are preserved. We aim to give insight into how our foundational work forms the basis for a complete abstract framework through concrete examples derived from our categorical construction. Specifically, we show how key aspects of resource separation, local reasoning, and compositionality are captured within the categorical context. We hope the insights obtained from these examples demonstrate the generality and abstraction of the foundational framework that served as our motivation to investigate separation logic from this perspective. Furthermore, they suggest building upon this foundation to work towards the ultimate goal of developing a complete framework that provides unified semantics for separation logic.

In Section 2, we will give the necessary background on separation logic. Next, we will discuss literature related to the topic and provide a better understanding of categorical frameworks in Section 3. In Sections 4 and 6, we will define the categorical constructions that form the basis of our framework and use the concepts of internal categories and externalisations to confirm that these constructions are compatible with the framework. In Section 5, we will introduce the concept of sheaves. Finally, in Section 7, we will demonstrate how to extract the semantics of separation logic from the context of our framework by providing examples.

2 Separation Logic

Before we can give a category-theoretic framework for separation logic, we first need a deeper understanding of the semantics of separation logic. This section focuses on separation logic, a specialised logic for reasoning about programs with shared mutable states. We will emphasise the role of the separating conjunction and the frame rule, which provide features for breaking down complex proofs and ensuring composability when dealing with memory regions. The introduction given here is based on the storage separation logic explained in a paper by Reynolds [Rey02].

For clarity, we divide the introduction on separation logic into three parts:

1. The storage model: a *stack* describing contents of registers and a *heap* representing shared mutable data structures.
2. The programming language defines effects of programs on the shared data.
3. Assertions that describe the shared data structures.

Storage model:
The storage model in the storage separation logic is defined as follows: for storage access, we define a RAM model where values are natural numbers and addresses are integers:

$$\text{Values} \stackrel{\text{def}}{=} \mathbb{N} \quad \text{Addresses} \stackrel{\text{def}}{=} \mathbb{Z}$$

The heaps are the partial maps taking addresses to values

$$H \stackrel{\text{def}}{=} \bigcup_{\substack{A \subseteq \text{Addresses} \\ \text{fin}}} (A \rightharpoonup \text{Values})$$

where the set Addresses is infinite, but the sets A considered in the union are required to be finite subsets of the set of Addresses
The stack S is a mapping from a finite set of variables V to values:

$$S \stackrel{\text{def}}{=} V \to \text{Values}$$

Now the state is a pair of the stack and the heap:

$$\text{States} \stackrel{\text{def}}{=} S \times H$$

Programming Language:
The programming language considered in separation logic is given formalised by Hoare [Hoa69] and extended by the following heap-manipulating commands:

$\langle comm \rangle$ ::= $\langle comm \rangle ; \langle comm \rangle$
| $\langle var \rangle$:= **cons**($\{\langle exp \rangle, \ldots, \langle exp \rangle\}$)
| $\langle var \rangle := [\langle exp \rangle]$
| $[\langle exp \rangle] := \langle exp \rangle$
| **dispose** $\langle exp \rangle$

This syntax is an extension of the imperative language formalised by [Hoa69]. The first command $\langle comm \rangle ; \langle comm \rangle$ denotes sequential composition of commands, $\langle var \rangle := \mathtt{cons}(\{\langle exp \rangle, \ldots, \langle exp \rangle\})$ allocates n consecutive cells, initialises them with the values of the expressions in the list and stores the address of the first cell in $\langle var \rangle$, where n is the number of expressions in the list. The command $\langle var \rangle := [\langle exp \rangle]$ looks up the content in the address $[\langle exp \rangle]$ and stores it in $\langle var \rangle$. The command $[\langle exp \rangle] := \langle exp \rangle$ mutates the value stored in $\langle [exp] \rangle$ by the value of the expression. Finally, dispose$\langle exp \rangle$ deallocates the address.

Let us look at how this language can be used in an example.

Command	**Current Stack**	**Current Heap**	**Stack After**	**Heap After**
Allocation $x :=$ **cons**$(1, 2)$;	$x : 3, y : 4$	empty	$x : 7, y : 4$	$7 : 1, 8 : 2$
Lookup $y := [x]$;	$x : 7, y : 4$	$7 : 1, 8 : 2$	$x : 7, y : 1$	$7 : 1, 8 : 2$
Mutation $[x + 1] := 3$;	$x : 7, y : 1$	$7 : 1, 8 : 2$	$x : 7, y : 1$	$7 : 1, 8 : 3$
Deallocation **dispose**$(x + 1)$;	$x : 7, y : 1$	$7 : 1, 8 : 3$	$x : 7, y : 1$	$7 : 1$

Assertions: In order to describe the heap, the following assertions are introduced:

$\langle assert \rangle ::= \mathbf{emp}$
$\quad | \quad \langle exp \rangle \mapsto \langle exp \rangle$
$\quad | \quad \langle assert \rangle * \langle assert \rangle$
$\quad | \quad \langle assert \rangle \mathrel{-\!\!*} \langle assert \rangle$

The grammar is explained as follows:

- Empty heap: **emp**
  The heap is empty.
- Singleton heap: $e \mapsto e'$
  The heap contains one element at location e with value e'.
- Separating conjunction: $p_1 * p_2$
  The heap can be divided into two disjoint parts H, H' such that $H \cap H' = \emptyset$ and p_1 holds for H and p_2 holds for H'.
- Separating implication: $p_1 \mathrel{-\!\!*} p_2$
  If there exists the heap H, for which p_1 holds, then the separating implication states that the heap can be extended for a disjoint heap H', and then, p_2 holds for the extended heap, where H' is universally quantified.

In separation logic, programs are reasoned about using Hoare triples such that for a Hoare triple $\{p\}C\{q\}$, if starting from any state in which the set of assertions $\{p\}$ holds, if the program C is executed, we land in a state where the set of assertions $\{q\}$ holds. The following example shows how the effect of a program can be described in separation logic:

$\{x \mapsto 3 * y \mapsto 5\}$
$[x] := [x] + 1$
$\{(x \mapsto 4 * y \mapsto 5) \mathrel{-\!\!*} y \mapsto 5\}$

While the assertions describing the heap give a local specification of the memory, we can infer the global specification of the memory via the separating conjunction. Consider a program that modifies no variables occurring in a part of the heap described by assertions r. We can write this as the Hoare triple: $\{p * r\}C\{q * r\}$ for a program c. As this specification is tight, the union of the parts of the heap that p, q, and r specify should form the whole memory. Therefore, we can see that for the following specification, $\{p\}c\{q\}$, the initial Hoare triple holds for the entire memory. This can be used to form the following inference rule, called the frame rule:

$$\frac{\{p\}C\{q\}}{\{p * r\}C\{q * r\}}$$

With the frame rule, we can extend the heap mutating commands that only mutate a part of the heap:

$$\frac{}{\{e \mapsto -\}[e] := e'\{e \mapsto e'\}} \text{ (local)}$$

to commands that mutate the entire memory:

$$\frac{}{\{(e \mapsto -) * r\}[e] := e'\{(e \mapsto e') * r\}} \text{ (global)}$$

We can even reason backwards using the separating implication:

$$\frac{}{\{(e \mapsto -) * ((e \mapsto e') \mathrel{-\!\!*} p)\}[e] := e'\{p\}} \text{ (backwards reasoning)}$$

via the following inference rule:

$$\frac{p}{q * (q \mathrel{-\!\!*} p)}$$

If the memory satisfies q and the changes specified by $q \mathrel{-\!\!*} p$ can be made without affecting the rest of the memory described by q, then the resulting memory must satisfy p.

In the subsequent sections, we will build upon this understanding of separation logic and use it to investigate which mathematical abstractions could lead to an abstract framework that brings out the underlying logic.

3 Related Work

In this section, work related to the topic of separation logic will be presented. The introduction briefly mentioned our motivations for viewing separation logic from a category-theoretic perspective. As categorical logic is connected with this approach, literature dealing with this topic will be discussed to emphasise the value of categorical frameworks.

3.1 Categorical Logic

First, we must ask ourselves: What does giving a categorical framework or categorical semantics of a logic mean? To see what this means, we will compare by providing concrete implementations of a logic to see what the differences are and understand what a categorical framework is. Let us say we design an algorithm to sort a list of numbers. In a conventional model, we write a step-by-step algorithm explicitly defining how to compare elements, swap them, and repeat until the list is sorted. This algorithm serves as a direct instance of the sorting process. Now, consider a different approach. Instead of implementing a specific algorithm, we specify properties that a sorting algorithm must satisfy. This framework then defines essential properties of sorting algorithms without dictating the exact implementation. To pull this back to categorical logic, we emphasised specifying relationships and properties between mathematical structures. Lawvere [Law69] made significant contributions by pioneering the development of categorical semantics for logic. His work laid the groundwork for understanding the relationships between logic and category theory, opening up new perspectives for the study of formal systems and their interpretations. Later in his book Categorical Logic and Type Theory, Jacobs [Jac99] provides a categorical framework from the perspective of fibred categories. In this book, he aptly points out some advantages of categorical frameworks as opposed to concrete models. For instance, the ability to quickly identify that specific models are an instance of a

logical framework or the advantage of deepening one's understanding of logical systems. Providing categorical semantics has been explored before in the context of separation logic. A type calculus for separation logic, or more specifically, higher-order frame rules, has been constructed by Birkendal and co-authors [BBTS07a] [BTSY06]. In [MS18], Melliès and Stefanesco establish the frame rule and its concurrent variant using game semantics.

3.2 Extensions of the Frame Rule

Another advantage, not explicitly mentioned by Bart Jacobs, of categorical frameworks is that these frameworks transpose knowledge of earlier models to novel models via the language of category theory. As the frame rule is highly adaptive, it can be used to describe the separation of all kinds of other resources. Because of this, many variations on separation logic and the frame rule have been constructed. To bring attention to the primary: in [O'H04], O'Hearn extended the frame rule to establish the correctness of concurrent programs. A separation logic for probabilistic programming has been developed by [BHL19]. Biering and co-authors used bunched implication hyper-doctrines to extend separation logic to include higher-order types [BBTS07b].

3.3 Sheaves

We will use the sheaf theory to synchronise local observations on the shared mutable data to a global observation. We refer to Section 5 for a more detailed explanation of sheaf semantics. The idea that sheaves can be utilised to glue together local observations to provide a global observation consistently highlights the relevance of sheaves in addressing challenges related to the interaction of distributed systems, that is, which properties can be verified modularly when connecting the systems or the modelling of concurrent systems. In the paper of Tsukada and Luke Ong [TO15], the authors use game semantics to provide a model for stateless non-determinism. They point out that infinite non-determinism provides difficulties for reasoning about concurrent programs since it becomes challenging to prove that a particular event must always occur in all possible executions, as there are infinitely many interleavings of the program's actions. They use the notion of sheaves to prevent requiring additional information of infinite traces. Sofronie-Stokkermans [Sof09] develops a modular method for verifying the properties of distributed systems. A topology models the interacting systems, and sheaves capture states and transitions. The author then exploits the logic of sheaves to determine which properties are kept when connecting the systems. To our knowledge, sheaves have not been employed before in the context of separation logic.

3.4 preliminaries

In the coming sections we will define categorical constructions. This section will be used to clarify the notation we will use to define these structures.
For categories, we use calligraphic writing:

$$\mathcal{C}, \mathcal{D}, \mathcal{L}...$$

We write fundamental categories such as Set, the category of sets or Mnd(Set), the category of monads on Set in Roman font.

We use capital letters for objects and functors:

$$C \in \mathcal{C}, F : \mathcal{C} \to \mathcal{D}, ...$$

For natural transformations, we use lower-case Greek letters:

$$\alpha : F \Rightarrow G, \eta : \mathrm{Id} \Rightarrow T, ...$$

We use lower-case letters for morphisms:

$$f : C \to D, i : \ell' \to \ell$$

There are, of course, exceptions. For objects in the preorder category, we use ℓ, ℓ', or ℓ_1.
We use boldface in two instances: To denote the Grothendieck construction $\mathbb{G}$ and to define the base category $\mathbb{B}$ in which internal categories live.
For fibres, we use the lower-case letter p; for projections used in pullbacks, we use p_1 and p_2. However, we use π for projections of products and ends.

4 Local State Monad

In order to investigate the essence of the frame rule from a category-theoretic perspective, we first need to provide a categorical framework for reasoning about the effects of a program on a set of initial states. Take the Hoare triple $\{P\}C\{Q\}$ for example. We want a way of reasoning about the programmatic effects of the program C for all possible initial states $\{P\}$ and then be able to prove the Hoare triple holds. Utilising category theory to provide models describing computational effects has been explored thoroughly. Eugenio Moggi [Mog91] introduced categorical semantics for programmatic effects based on monads. Moggi identifies the following computational effects: `partiality`, `nondeterminism`, `side-effects`, `exceptions`, `continuations`, and `interactive input` and `output`. In order to give a notion to these computational effects, Moggi introduces a monad T as a unary type constructor and then obtains an object of computation by applying T to the object of values. Then, he forms the category of programs to give a common notion of these effects by constructing the Kleisli category for a given Kleisli triple.

Definition 4.1 (Kleisli triple and Kleisli Category)**.** *Given a monad T on a category $\mathcal{C}$, being the triple (T, η, μ), where T is the endofunctor $T : \mathcal{C} \to \mathcal{C}$ together with two natural transformations $\eta : \mathrm{Id} \Rightarrow T$ and $\mu : T^2 \Rightarrow T$, we define the following concepts:*

1. ***Kleisli Triple**:*
   - *A Kleisli triple over a category $\mathcal{C}$ is a triple $(T, \eta, \odot)$ where:*
     - *$T : \mathcal{C} \to \mathcal{C}$ is an endofunctor.*
     - *$\eta_A : A \to TA$ for all objects A in $\mathcal{C}$.*
     - *$f\odot : TA \to TB$ for all morphisms $f : A \to TB$ in $\mathcal{C}$*
   - *It satisfies the following equations:*
     - *$\eta_A\odot = \mathrm{id}_{TA}$*

- $f \odot \circ \eta_A = f$ *for all* $f : A \to TB$
- $g \odot \circ f \odot = (g \odot \circ f) \odot$ *for all* $f : A \to TB$ *and* $g : B \to TC$

The Kleisli triple corresponds with the monad, and while the notation $\odot$ *might seem suggestive now, its place will become clear in the definition of the Kleisli category.*

2. ***Kleisli Category***

- *The Kleisli category* $\mathcal{C}_T$ *of a monad* T *on the category* $\mathcal{C}$ *has the following properties:*
  (a) *Objects in* $\mathcal{C}_T$ *are the objects in* $\mathcal{C}$.
  (b) *Morphisms in* $\mathcal{C}_T$ *from* A *to* B *are the morphisms* $f : A \to TB$ *in* $\mathcal{C}$.
- *The Kleisli composition in* $\mathcal{C}_T$ *is defined as follows: for morphisms* $f : A \to TB$ *and* $g : B \to TC$, *the Kleisli composition is given by* $g \odot f = \mu_C \circ T(g) \circ f : A \to TC$.
- *Identity morphisms in* $\mathcal{C}_T$ *on objects* A *are the Kleisli morphisms given by the unit on* $A : \eta_A : A \to TA$.

The Kleisli category for a given Kleisli triplet then serves as the category of programs, where the morphisms are the programs themselves and the objects A are the computational notions of programs of type A. In the same paper, Moggi shows how to provide a model to understand the read and write operations of a memory register as a specific monadic effect called the state monad. Moggi defines this monad as follows: One defines a monad in the category of sets as follows:

$$T : \text{Set} \to \text{Set}$$

such that for a given set A and a fixed set of states S, we have that

$$TA = (A \times S)^S$$

where each element of S is a finite list representing the states of each allocated register.

While this traditional state monad may allow us to provide a framework for reasoning about the programmatic effects of C, separation logic extends the Hoare triple with the separation conjunction to reason about different parts of the memory independently. Therefore, the focus will lie more on reasoning about the effects of a program on local states and, when the affected regions are disjoint, compose the effects to model the interaction of the local effects so that we can reason about the entire state. Plotkin and Power [PP02] point out that the perspective of Moggi does not provide a model on which to compare the notions of programmatic effects. Most importantly, there is, for instance, a distinction between global and local phenomenons of the notions of computation that the perspective of Moggi does not capture. Therefore, the traditional state monad loses this local perspective, so we require a monadic construction that allows us to analyse the local state. In the paper of Maillard and Melliès[MM15] and in the paper of Plotlin and Power discussed earlier, the authors investigate how to construct such a monad. We will use some ideas in these papers and develop a local state monad that aligns well with the frame rule in separation logic.

We start by defining our memory model. Let Loc be a finite set regarded as memory locations. We define the following index category:

$$\mathcal{L} = (\mathcal{P}(\text{Loc}), \subseteq)$$

This category arises from the powerset of memory locations $\mathcal{P}(\text{Loc})$.
In this section, we will show there exists a functor S and define this functor that sends objects in $\mathcal{L}^{\text{op}}$ to the category of monads:

$$S : \mathcal{L}^{\text{op}} \to \text{Mnd}$$

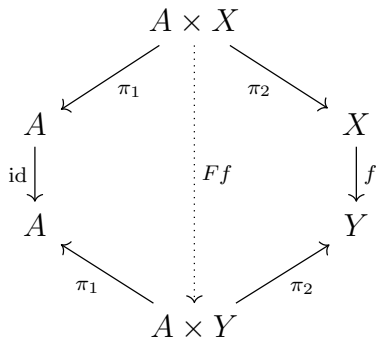
4.1 Binary Product Functor

Lemma 4.1. *For $A \in Set$, a binary product $A \times (-)$ induces a functor*

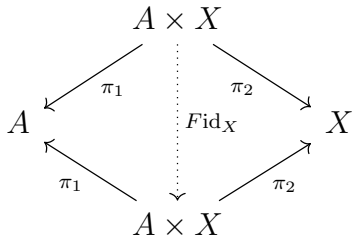
$$F : \text{Set} \to \text{Set}$$

such that for every set X, we have that $FX = A \times X$.

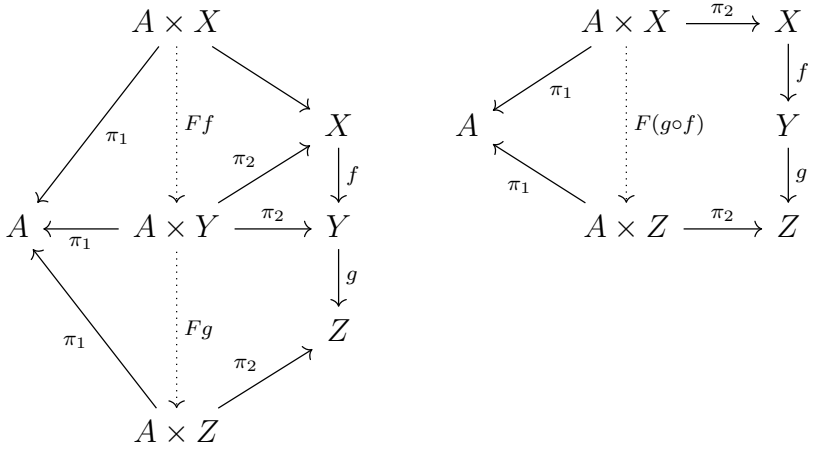
Proof. Given a map $f : X \to Y$, we have that $Ff : A \times X \to A \times Y$, induced by the universal mapping property(UMP) of the binary product:



By this result, we have that applying F to id_X induces the unique morphism $F\text{id}_X : FX \to FX$, such that the following diagram commutes,
and $\pi_1 \circ F\text{id}_X = \pi_1$, and $\pi_2 \circ F\text{id}_X = \pi_2$:



Now any other morphism that makes the above diagram commute must be the morphism $F\text{id}_X$, so to show that $\text{id}_{FX} = F\text{id}_X$, we show that $\pi_1 \circ \text{id}_{FX} = \pi_1$ and $\pi_2 \circ \text{id}_{FX} = \pi_2$, which hold by definition of the identity. F also preserves composition. Given morphisms $f : X \to Y$ and $g : Y \to Z$, applying F to the composition of g and f induces the unique map $F(g \circ f) : A \times X \to A \times Z$, such that $\pi_1 \circ F(g \circ f) = \pi_1$ and $\pi_2 \circ F(g \circ f) = g \circ f \circ \pi_2$. We will now show that $Fg \circ Ff = F(g \circ f)$ by showing $\pi_1 \circ Fg \circ Ff = \pi_1$ and $\pi_2 \circ Fg \circ Ff = g \circ \pi_2 \circ Ff$. This holds by the UMP of the product: Fg induces a unique map, such that $\pi_1 \circ Fg = \pi_1$, and $\pi_2 \circ Fg = g \circ \pi_2$. The diagrams below summarise the proof:



□

Now that we have proven that a binary product induces a functor, we proceed by constructing a functor that sends an object ℓ in $\mathcal{L}$ to the category of monads. Recall the traditional state monad defined by Moggi: $TA = (A \times S)^S$. We can use Lemma 4.1 to prove that there exists a functor that sends an object in Set to the binary product as seen in the traditional state monad.

Proposition 4.1. *Let $M : \mathcal{L}^{\text{op}} \to \text{Set}$ be a functor defined for every ℓ in $\mathcal{L}$ by $M(\ell) = [\ell, \mathbb{Z}]$, the set of all maps from ℓ to $\mathbb{Z}$. For a morphism $i : \ell \subseteq \ell'$ in $\mathcal{L}$ the functor is defined as $M(i) : M(\ell') \to M(\ell)$. For a fixed ℓ in $\mathcal{L}$, the mapping $F : \text{Set} \to \text{Set}$ defined for every X in Set by $FX = M(\ell) \times X$, is a functor.*

Proof. We have that $M(\ell) \times X$ is a set, and by Lemma 4.1, F preserves identity and composition. □

4.2 Global State Monad

Before we show the construction of the local state monad, some intuition behind how we choose to define the construction is required. Eventually, we want to construct a state monad that, at a given state, returns a set of operations that can be applied to the memory specified by the set of maps $M(\ell)$ for some set of memory locations ℓ in $\mathcal{L}$ after transitioning between states. Then, we want to utilise the properties of sheaves to infer information about this set of operations based on local observations for some $\ell'' \subseteq \ell'$, for objects ℓ', ℓ'' in $\mathcal{L}$.
First, define the slice category $\mathcal{L}/\ell$ of the category $\mathcal{L}$ over an object ℓ in $\mathcal{L}$, where objects are all the morphisms $i : \ell' \subseteq \ell$ in $\mathcal{L}$, and morphisms are from $j : \ell'' \subseteq \ell'$ to $i : \ell' \subseteq \ell$ in $\mathcal{L}$, such that for morphisms $k : \ell'' \subseteq \ell$, we have that $i \circ j = k$.
To get a sheaf of state monads, we have to construct a presheaf that, given an inclusion $\ell'' \subseteq \ell'$ can take a subset of possible memory operations $M(\ell')$ associated with a set of memory operations $M(\ell')$. Therefore, we want to define the following functor:

Definition 4.2. *For the memory specified by an object ℓ in $\mathcal{L}$, we want to define a functor*

$$T_\ell : (\mathcal{L}/\ell)^{\text{op}} \to \text{Endo(Set)}$$

that is defined on objects $i : \ell' \subseteq \ell$ in $\mathcal{L}/\ell$ as

$$T_\ell(i) = [M(\ell), M(\ell') \times \text{Id}]$$

Now, $T_\ell(i)$ is a set of functors that for a given map $m \in M(\ell)$, which is a memory operation on the memory ℓ, take m to the pair $(m' \times \mathrm{id})$, with $m' \in M(\ell')$. This pair represents a state after executing the operation m, where m' is a memory operation on the memory ℓ' and id *represents the set of possible states we can transition to.*

On morphisms between i and $j : \ell'' \subseteq \ell$ in $\mathcal{L}/\ell$ for some morphism $k : \ell'' \subseteq \ell'$ as

$$T_\ell(k) : T_\ell(i) \to T_\ell(j)$$

These maps can be seen as a natural transformation restricting the set of output maps of the state monad, with all objects X in Set *having components*

$$T_\ell(k)_X : T_\ell(i)(X) \to T_\ell(j)(X)$$

with $T_\ell(i)(X) = [M(\ell), M(\ell') \times X]$ and $T_\ell(j)(X) = [M(\ell), M(\ell'') \times X]$. Naturality of $T_\ell(k)$ will be proven later in this section.

Let us look at the construction T_ℓ in the context of modelling stateful computation. We can specify what happens for state transformations and state transitions for a part of the memory specified by ℓ. Eventually, we want to specify how these transitions occur for different subsets of the global memory, $\ell' \subseteq \ell$. Therefore, we also want a natural way of altering the domain of the functor $T_{\ell'}$, $\mathcal{L}/\ell'$ to the domain $\mathcal{L}/\ell$ of the functor T_ℓ for an inclusion $i : \ell \subseteq \ell'$. These maps essentially extend the input maps of the state monad. Now, as our functors require maps extending the input maps and restricting the output maps, a well-organised solution to manage these restrictions is needed. A good example of this problem is the following: given objects ℓ'', ℓ', ℓ in $\mathcal{L}$ with $\ell'' \subseteq \ell' \subseteq \ell$ and the functor $T_{\ell'}(\ell' \subseteq \ell')$. We restrict $\ell' \subseteq \ell'$ to $\ell'' \subseteq \ell'$ and map $T_{\ell'}$ to T_ℓ, giving the functor $T_\ell(\ell'' \subseteq \ell')$. The domain of T_ℓ is $\mathcal{L}/\ell$, so we need a way to lift the restriction $\ell'' \subseteq \ell'$ to $\ell'' \subseteq \ell$. We will introduce the Grothendieck construction to solve this problem.

Definition 4.3 (opfibration [Jac99])**.** *Given a base category $\mathcal{B}$ and the total category $\mathcal{E}$, a functor $p : \mathcal{E} \to \mathcal{B}$ is an opfibration if the functor $p : \mathcal{E}^{\mathrm{op}} \to \mathcal{B}^{\mathrm{op}}$ is a fibration. The cartesian arrows in $\mathcal{E}^{\mathrm{op}}$ are opcartesian arrows in $\mathcal{E}$. A cleavage for an opfibration consists of opcartesian arrows $\phi : e \to e'$ such that for a morphism $p(e') \to b$ in $\mathcal{B}$ there exists a cartesian lifting such that $p(\phi) = f$. A choice of a cleavage, therefore, gives rise to the functor $\Sigma_f : \mathcal{E}_a \to \mathcal{E}_b$ for each morphism $f : a \to b$ in $\mathcal{B}$.*

Definition 4.4 (Grothendieck Construction [GR71])**.** *Given a functor $F : \mathcal{C} \to$ Cat for a category $\mathcal{C}$, the Grothendieck construction of F, denoted as $\mathbb{G}(F)$ has as objects the pairs (C, A) for an object C in $\mathcal{C}$ and an object A in $F(C)$. Morphisms in the Grothendieck construction between objects (C, A) and (C', A) with C' are the pairs (f, ϕ) given the morphism $f : C' \to C$ in $\mathcal{C}$ and the morphism $\phi : F(f)(A) \to A'$.*

Now that we have established the fundamental definitions of opfibrations and the Grothendieck construction, we will introduce the construction of these concepts that fit with our goal of solving the mixed variance of the functor T_ℓ for an object ℓ in the index category $\mathcal{L}$.

Definition 4.5. *Given the index category $\mathcal{L} = (\mathcal{P}(\mathrm{Loc}), \subseteq)$ we define a functor*

$$\mathcal{L}/(-) : \mathcal{L} \to \mathrm{Cat}$$

taking objects ℓ in $\mathcal{L}$ to the opposite slice category $(\mathcal{L}/\ell)^{\mathrm{op}}$. The functor is defined on morphisms $i : \ell \subseteq \ell'$ as $\mathcal{L}/(i) : \mathcal{L}/\ell \to \mathcal{L}/\ell$.

Definition 4.6. *We define the functor*

$$P : \mathcal{L} \to \mathrm{Cat}$$

that is given by composing $[-, \mathrm{Endo(Set)}] : \mathrm{Cat}^{\mathrm{op}} \to \mathrm{Cat}$ and $\mathcal{L}/(-) : \mathcal{L} \to \mathrm{Cat}$.
The functor P takes an object ℓ in ℓ to the category of all functors from $\mathcal{L}/\ell$ to Endo(Set).
Now, we define the Grothendieck construction of P, denoted as $\mathbb{G}(P)$. Objects in $\mathbb{G}(P)$ are the pairs

$$(\ell, F)$$

with ℓ an object in $\mathcal{L}$ and $F \in [\mathcal{L}/\ell, \mathrm{Endo(Set)}] = P(\ell)$.
Morphisms between (ℓ', G) and (ℓ, F) with ℓ, ℓ' in $\mathcal{L}$, $F \in P(\ell)$ and $G \in P(\ell')$ are the pairs

$$(i, \phi)$$

with $i : \ell' \subseteq \ell$ being a morphism in $\mathcal{L}$ and the morphism $\phi : P(i)(G) \to F$.
The Grothendieck construction $\mathbb{G}(P)$ gives rise to the opfibration $p : \mathbb{G}(P) \to \mathcal{L}$, which in turn induces the functor

$$\Sigma_i : \mathcal{L}/\ell' \to \mathcal{L}/\ell$$

for all morphisms $i : \ell' \subseteq \ell$ in $\mathcal{L}$.
The functor is defined on objects $k \subseteq \ell'$ in $\mathcal{L}/\ell'$ as

$$\Sigma_i(k \subseteq \ell') = k \subseteq \ell$$

and on morphisms $f : (k \subseteq \ell') \to (u \subseteq \ell')$ as

$$\Sigma_i(f) : (k \subseteq \ell) \to (u \subseteq \ell)$$

If we now come back to see how P is defined on the morphisms $i : \ell' \subseteq \ell$ in $\mathcal{L}$, we let $P(i) = \Sigma_i$ and for a functor G in $P(\ell')$ we have that

$$P(i)(G) = G \circ \Sigma_i$$

Now given the natural transformation $\phi : P(i)(G) \to F$ in the Grothendieck construction $\mathbb{G}$ with F in $P(\ell)$ we have that

$$\phi(P(i)(G))(k \subseteq \ell') = \phi(G \circ \Sigma_i)(k \subseteq \ell) = F(k \subseteq \ell)$$

Given morphisms $f : (k \subseteq \ell') \to (u \subseteq \ell')$, ϕ is defined by composing f with the functor Σ_i and applying ϕ to $P(i)(G)$:

$$\phi(P(i)(G)(f)) = \phi(G \circ \Sigma_i)(\Sigma_i(f)) = F(\Sigma_i(f))$$

Proposition 4.2. *The mapping $P : \mathcal{L} \to \mathrm{Cat}$ is a functor.*

Proof. Identity is preserved:

$$P(\mathrm{id}_\ell)(F) = F \circ \Sigma_{\mathrm{id}:\ell\subseteq\ell} = F \circ \mathrm{id}_{\mathcal{L}/\ell} = \mathrm{id}_{P(\ell)}(F) \Rightarrow P(\mathrm{id}_\ell) = \mathrm{id}_{P(\ell)}$$

Composition is preserved:

$$P(\ell_1 \subseteq \ell_2 \circ \ell_2 \subseteq \ell_3)(F) = P(\ell_1 \subseteq \ell_3)(F) = F \circ \Sigma_{\ell_1\subseteq\ell_3}$$

and

$$\begin{aligned} P(\ell_1 \subseteq \ell_2) \circ P(\ell_2 \subseteq \ell_3)(F) &= P(\ell_1 \subseteq \ell_2) \circ F \circ \Sigma_{\ell_2\subseteq\ell_3} \\ &= F \circ \Sigma_{\ell_1\subseteq\ell_2} \circ \Sigma_{\ell_2\subseteq\ell_3} \\ &= F \circ \Sigma_{\ell_1\subseteq\ell_3} \end{aligned}$$

with $F \in P(\ell)$. □

This shows that we have defined a construction that, given a morphism $i : \ell' \subseteq \ell$ lifts a functor in $P(\ell')$ to a functor in $P(\ell)$, whilst respecting the objects and morphisms in the respective slice categories involved.

Now, we will still have to see how we can extract the state monad T_ℓ from this perspective for an object ℓ in $\mathcal{L}$. We define the functor $\triangle_{\mathbb{1}}$, that maps an $\ell \in \mathcal{L}$ to the category Δ_ℓ, with only one object, let us say $*$, and a unique morphism id_*. Then, we define a natural transformation between $\triangle_{\mathbb{1}}$ and P in such a way that the fibres of its Grothendieck construction give back the state monad. When we take the Grothendieck construction of $\triangle_{\mathbb{1}}$, we obtain an isomorphism between the Grothendieck construction and the category $\mathcal{L}$:

Theorem 1. *The Grothendieck construction of the functor $\Delta_{\mathbb{1}} : \mathcal{L} \to \mathrm{Cat}$, sending objects ℓ in $\mathcal{L}$ to the terminal category, is isomorphic to the category $\mathcal{L}$:*

$$\mathcal{L} \cong \mathbb{G}(\Delta_{\mathbb{1}})$$

Proof. We define two maps, $\zeta : \mathcal{L} \to \mathbb{G}(\Delta_{\mathbb{1}})$ and $\theta : \mathbb{G}(\Delta_{\mathbb{1}}) \to \mathcal{L}$ and show their compositions yield the identity functors. We first define ζ as follows:

1. For each object $\ell \in \mathcal{L}$, ζ maps ℓ to the object $(\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$, where $*$ is the unique object in the terminal category. For each morphism $i : \ell' \subseteq \ell$ in $\mathcal{L}$, ζ maps i to the morphism $(i, \mathrm{id}_*) : (\ell', *) \to (\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$.

2. For each object $(\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$, θ maps $(\ell, *)$ to the object ℓ in $\mathcal{L}$. For each morphism $(i, \mathrm{id}_*) : (\ell', *) \to (\ell, *)$, θ maps (i, id_*) to the morphism $i : \ell' \subseteq \ell$ in $\mathcal{L}$.

Now we verify that the compositions yield the identity functor:

1. $\zeta \circ \theta = \mathrm{id}_{\mathbb{G}(\Delta_{\mathbb{1}})}$: for each object $(\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$, θ maps it to the object ℓ in $\mathcal{L}$ and ζ maps each object ℓ in $\mathcal{L}$ back to the objects $(\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$. For each morphism (i, id_*) in $\mathbb{G}(\Delta_{\mathbb{1}})$, θ maps it to the morphism $i : \ell' \subseteq \ell$ in $\mathcal{L}$. Then ζ maps each morphism $i : \ell' \subseteq \ell$ in $\mathcal{L}$ to the morphism $(i, \mathrm{id}_*) : (\ell', x) \to (\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$. Since only one morphism exists in the terminal category, id_* is uniquely determined. Therefore, the morphisms in $\mathbb{G}(\Delta_{\mathbb{1}})$ between $(\ell', *)$ and $(\ell, *)$ are in one-to-one correspondence with the morphisms between objects ℓ' and ℓ in $\mathcal{L}$. Therefore $\zeta \circ \theta$ is the identity functor on $\mathbb{G}(\Delta_{\mathbb{1}})$.

2. $\theta \circ \zeta = \mathrm{id}_{\mathcal{L}}$: for each object ℓ in $\mathcal{L}$, θ maps it to the object $(\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$ and ζ maps that object back to the same object ℓ in $\mathcal{L}$. For each morphism $\ell' \subseteq \ell$ in $\mathcal{L}$, ζ takes it to the morphism $(i, \mathrm{id}_*) : (\ell', *) \to (\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$, which θ takes back to the same morphism i in $\mathcal{L}$, again, since there is a one-to-one correspondence between the morphisms, showing that $\theta \circ \zeta$ is the identity functor on $\mathcal{L}$.

□

Proposition 4.3. *There exists a natural transformation $\alpha : \Delta_{\mathbb{1}} \to P$, such that for objects ℓ in $\mathcal{L}$ the fibres of $\mathbb{G}(\alpha)$ give back the functor T_ℓ 4.2.*

Proof. We need to define the components of α at each ℓ in $\mathcal{L}$ and then show that these components satisfy the naturality condition. For an object ℓ_1 in $\mathcal{L}$, we define a functor $\alpha_{\ell_1} : \mathbb{1} \to P(\ell_1)$.

On objects: for the only object $*$ in $\Delta_{\mathbb{1}}$, $\alpha_{\ell_1}(*)$ with the alternative notation T_{ℓ_1} is defined as a functor as follows:

1. On objects: T_{ℓ_1} is the endofunctor that maps $\ell_2 \subseteq \ell_1$ in $\mathcal{L}/\ell_1$ to $[M(\ell_1), M(\ell_2) \times \mathrm{Id}]$.

2. On morphisms: Consider the morphism $j : \ell_3 \subseteq \ell_2$ such that $\ell_3 \subseteq \ell_2 \subseteq \ell_1$ in $\mathcal{L}/\ell_1$. Morphisms in the slice category are defined as the commutative triangle $i : \ell_2 \subseteq \ell_1$, $j : \ell_3 \subseteq \ell_2$ and $k : \ell_3 \subseteq \ell_1$, such that $i \circ j = k$. Therefore, we apply j to i, which sends $\ell_2 \subseteq \ell_1$ to $\ell_3 \subseteq \ell_1$. This means that T_{ℓ_1} is defined on every morphism $j : \ell_3 \subseteq \ell_2$ in $\mathcal{L}/\ell_1$ as

$$T_{\ell_1}(j) : T_{\ell_1}(k) \to T_{\ell_1}(i)$$

with

$$T_{\ell_1}(k) = [M(\ell_1), M(\ell_3) \times \mathrm{Id}], T_{\ell_1}(i) = [M(\ell_1), M(\ell_2) \times \mathrm{Id}]$$

Now, we need to show that these component functors satisfy the naturality condition for each morphism $\ell' \subseteq \ell$ by showing the diagram below commutes.

$$\begin{array}{ccc} \mathbb{1} & \xrightarrow{\alpha_{\ell'}} & P(\ell') \\ \Big\downarrow{\scriptstyle \Delta_{\mathbb{1}}(i)} & & \Big\downarrow{\scriptstyle P(i)} \\ \mathbb{1} & \xrightarrow{\alpha_{\ell}} & P(\ell) \end{array}$$

To show that the diagram commutes, we have to show that

$$\alpha_\ell(*) = P(i) \circ \alpha_{\ell'}(*)$$

These are defined as follows:

$$P(i) \circ \alpha_{\ell'}(*) = [\mathcal{L}/\ell, \text{Endo(Set)}]$$

$$\alpha_\ell(*) = [\mathcal{L}/\ell', \text{Endo(Set)}]$$

We know that $P(i)$ lifts $\mathcal{L}/\ell'$ to $\mathcal{L}/\ell$ so $P(i) \circ \alpha_{\ell'}(*)$ is a functor in $[\mathcal{L}/\ell, \text{Endo(Set)}]$, just as $\alpha_\ell(*)$ is. Therefore, we have to show that

$$P(i) \circ \alpha_{\ell'}(*)(i) = \alpha_\ell(*)(i)$$

for all objects $i : \ell' \subseteq \ell$ in $\mathcal{L}/\ell$.
We have that

$$P(i) \circ \alpha_{\ell'}(*)(i) = [M(\ell), M(\ell') \times \text{Id}]$$

and

$$\alpha_\ell(*)(i) = [M(\ell), M(\ell') \times \text{Id}]$$

This means we have to show that

$$P(i) \circ \alpha_{\ell'}(*)(i)(m) = \alpha_\ell(*)(i)(m)$$

for all m in $M(\ell)$.
Given an object X in Set we say that $\alpha_\ell(*)(i)_X$ maps m to the tuple (m', x), which is an object in the binary product $M(\ell') \times X$ with projections $\pi_1 : M(\ell') \times X \to M(\ell')$ and $\pi_2 : M(\ell') \times X \to X$. Then $P(i) \circ \alpha_{\ell'}(*)(i)_X$ maps m to the tuple (m'', x'), which is an object in the binary product $(M(\ell') \times X)'$ with projections $\pi_1' : (M(\ell') \times X)' \to M(\ell')$ and $\pi_2' : (M(\ell') \times X)' \to X$. This shows that

$$\pi_1 \circ \alpha_\ell(*)(i)(m) = \pi_1' \circ P(i) \circ \alpha_{\ell'}(i)(m)$$

and

$$\pi_2 \circ \alpha_\ell(*)(j)(m) = \pi_2' \circ P(i) \circ \alpha_{\ell'}(*)(i)(m)$$

□

Putting this together, we have a natural transformation that, given inclusions $i : \ell' \subseteq \ell$, $j : \ell'' \subseteq \ell'$ and $\ell'' \subseteq \ell$ in $\mathcal{L}$, the natural transformation $(P(i) \circ \alpha_\ell(*))(T_\ell)(j) = P(\ell')(T_{\ell'})(j)$ restricts the first argument in $\alpha_\ell(*)(j)$ from $[M(\ell), M(\ell'' \times \text{Id}]$ to $[M(\ell'), M(\ell'' \times \text{Id}] = \alpha_{\ell'}(*)(j)$, and the map $\alpha_\ell(*)(T_\ell)(j)$, where j is now seen as a morphism in $(\mathcal{L}/\ell)^{\text{op}}$ restricts $\alpha_\ell(*)(T_\ell)(i) = [M(\ell), M(\ell') \times \text{Id}]$ to $\alpha_\ell(*)(k) = [M(\ell), M(\ell'') \times \text{Id}]$, showing that the map restricts the second argument.

Recall that we want T_ℓ to be an endofunctors-on-Set-valued presheaf on $\mathcal{L}/\ell$. Currently, we define T_ℓ as a functor in $\alpha_\ell(*)$. Functors in $\alpha_\ell(*)$ are defined on morphisms $k : (j : \ell'' \subseteq \ell) \to (i : \ell' \subseteq \ell)$ in $\mathcal{L}/\ell$ as

$$T_\ell(k) : T_\ell(j : \ell'' \subseteq \ell) \to T_\ell(i : \ell' \subseteq \ell)$$

but for T_ℓ to be a presheaf we want T_ℓ to be defined on k as

$$T_\ell(k) : T_\ell(i : \ell' \subseteq \ell) \to T_\ell(j : \ell'' \subseteq \ell)$$

Earlier we have defined the functor $M : \mathcal{L}^{\text{op}} \to \text{Endo(Set)}$. As M is contravariant, we can instead define $T_\ell(k)$ as

$$T_\ell(k) = [M(\ell), M(k : \ell'' \subseteq \ell') \times \text{Id}] : [M(\ell), M(\ell') \times \text{Id}] \to [M(\ell), M(\ell'') \times \text{Id}] = T_\ell(i) \to T_\ell(j)$$

Proposition 4.4. *A functor T_ℓ, or $\alpha_\ell(*)$ for an object ℓ in $\mathcal{L}$ is an endofunctor-on-*Set*-valued presheaf on $\mathcal{L}/\ell$.*

Proof. For objects $i : \ell' \subseteq \ell$ in $\mathcal{L}/\ell$, $T_\ell(i)$ is defined on objects X in Set as

$$T_\ell(i)(X) = [M(\ell), M(\ell') \times X]$$

and on morphisms $f : X \to Y$ as

$$T_\ell(i)(f) : T_\ell(i)(X) \to T_\ell(i)(Y)$$

Functoriality of T_ℓ and $T_\ell(i)$ has been proven earlier in this section. We have that $T_\ell(i)(X)$ is an object in Set so all that is left to prove is that for morphisms $k : (j : \ell'' \subseteq \ell) \to (i : \ell' \subseteq \ell)$ the map $T_\ell(k) : T_\ell(i) \to T_\ell(j)$ is a natural transformation. To prove this, we have to show the diagram below commutes for all components

$$T_\ell(k)_X : T_\ell(i)(X) \to T_\ell(j)(X)$$

with X in Set and for all morphisms $f : X \to Y$ in Set.

$$\begin{array}{ccc} T_\ell(i)(X) & \xrightarrow{T_\ell(i)(f)} & T_\ell(i)(Y) \\ \downarrow{\scriptstyle T_\ell(k)_X} & & \downarrow{\scriptstyle T_\ell(k)_Y} \\ T_\ell(j)(X) & \xrightarrow{T_\ell(j)(f)} & T_\ell(j)(Y) \end{array}$$

For all functors $h : M(\ell) \to M(\ell'') \times X$ in $T_\ell(i)(X)$ we have that

$$(T_\ell(j)(f) \circ T_\ell(k)_X)(h) = (\mathrm{id} \times f) \circ (M(k) \times \mathrm{id}) \circ h$$

and

$$(T_\ell(k)_Y \circ T_\ell(i)(f))(h) = (M(k) \times \mathrm{id}) \circ (\mathrm{id} \times f) \circ h$$

Because of functoriality of T_ℓ, we have that

$$(T_\ell(k)_Y \circ T_\ell(i)(f))(h) = (\mathrm{id} \times f) \circ (M(k) \times \mathrm{id}) \circ h$$

showing the diagram commutes, proving naturality of $T_\ell(k)$. □

4.3 Summary of Notation and Definitions

In earlier sections, we defined many constructions. For clarity, in this section, we will summarise the key constructions that will form the foundation of the local state monad. The definitions of $\mathbb{1}$, $\Delta_{\mathbb{1}}$ and $\mathbb{G}(P)$ (the Grothendieck construction of P) will not be restated as they will not be regularly used in the coming sections.

- The category
$$\mathcal{L} = (\mathcal{P}(\mathrm{Loc}, \subseteq)$$
is an index category with objects ℓ in the powerset of memory locations $\mathcal{P}(\mathrm{Loc})$. For objects ℓ', ℓ in $\mathcal{L}$, a morphism $i : \ell' \to \ell$ exists if ℓ' is included in ℓ. Therefore, we write the morphism i as $i : \ell' \subseteq \ell$. As $\mathcal{L}$ is an index category, the inclusion morphisms are unique.

- The functor
$$M : \mathcal{L}^{\mathrm{op}} \to \mathrm{Set}$$
is defined on objects ℓ in $\mathcal{L}$ as $M(\ell) = [\ell, \mathbb{Z}]$, which is the set of maps from the memory location ℓ to the set of integers. M is defined on the morphisms $i : \ell' \subseteq \ell$ as $M(i) : M(\ell) \to M(\ell')$.

- The slice category
$$\mathcal{L}/\ell$$
is the slice category of $\mathcal{L}$ over ℓ, where ℓ is an object in $\mathcal{L}$. Objects in this category are all morphisms i in $\mathcal{L}$, such that $\mathrm{cod}(i) = \ell$, where $\mathrm{cod}(i)$ is the co-domain of the morphism i. Morphisms in this category from $j : \ell'' \subseteq \ell$ to $i : \ell' \subseteq \ell$ are the morphisms $k : \ell'' \subseteq \ell'$ in $\mathcal{L}$, such that $k \circ i = j$.

- The functor $\mathcal{L}/(-)$ 4.5,
$$\mathcal{L}/(-) : \mathcal{L} \to \mathrm{Cat}$$
is the "slice-functor" that maps all objects ℓ in $\mathcal{L}$ to the opposite slice category $\mathcal{L}/\ell^{\mathrm{op}}$. The slice-functor is defined on morphisms $i : \ell' \subseteq \ell$ as $\mathcal{L}/(i) : \mathcal{L}/(\ell') \to \mathcal{L}(\ell)$.

- The functor P 4.6,
$$P : \mathcal{L} \to \mathrm{Cat}$$
given by composition of $[-, \mathrm{Endo(Set)}] : \mathrm{Cat}^{\mathrm{op}} \to \mathrm{Cat}$ and $\mathcal{L}/(-)$ is defined on objects ℓ in $\mathcal{L}$ as $P(\ell) = [\mathcal{L}/\ell, \mathrm{Endo(Set)}]$, which is the category of functors from $(\mathcal{L}/\ell)$ to the category of set valued endo-functors. P is defined on morphisms $i : \ell' \subseteq \ell$ as the natural transformation $P(i) : P(\ell') \to P(\ell)$.

- The natural transformation
$$\alpha : \Delta_{\mathbb{1}} \to P$$
has as components the functors $\alpha_\ell : \mathbb{1} \to P(\ell)$ for all objects ℓ in $\mathcal{L}$. For the terminal object $*$ in $\mathbb{1}$, this functor is also written as T_ℓ.

- The functor
$$T_\ell : \mathcal{L}/\ell \to \mathrm{Endo(Set)}$$
that maps objects $i : \ell' \subseteq \ell$ in $\mathcal{L}/\ell$ to $[M(\ell), M(\ell') \times \mathrm{Id}]$.

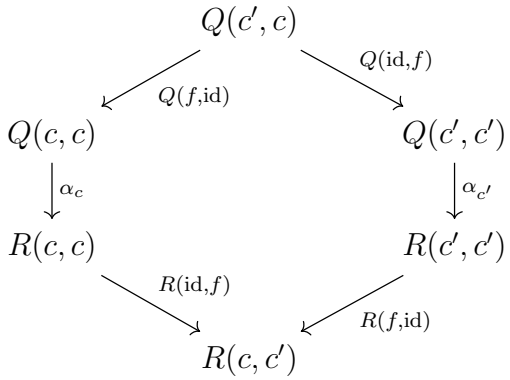
  It is defined on morphisms $k : (\ell'' \subseteq \ell) \to (\ell' \subseteq \ell)$ in the slice category by composition with the functor M, which gives the definition
$$T_\ell(k) : T_\ell(\ell' \subseteq \ell) \to T_\ell(\ell'' \subseteq \ell)$$
  with $T_\ell(\ell'') = [M(\ell), M(\ell'') \times \mathrm{Id}]$

- For an object $i : \ell' \subseteq \ell$ in $\mathcal{L}/\ell$ the functor
$$T_\ell(i) : \mathrm{Set} \to \mathrm{Set}$$
defined on objects X in Set as $T_\ell(i)(X) = [M(\ell), M(\ell') \times X]$ and on morphisms $f : X \to Y$ in Set as $T_\ell(i)(f) : T_\ell(i)(X) \to T_\ell(i)(Y)$.

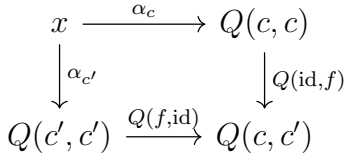
4.4 Local State Monad

There are certain limitations to using a point-wise approach for understanding these constructions. Studying the natural transformation α at individual components ℓ does not capture the overall coherence of the objects and morphisms in $\mathcal{L}$. Also, with this local approach, it becomes cumbersome to analyze the complex interactions that the functor $\alpha_\ell(*)$ might have with various objects and morphisms in $\mathcal{L}$, as seen in the proof of Proposition 4.3. Therefore, we will introduce a complex limit, called `end`, that can provide a global view of the functor's behaviour. This is especially advantageous when dealing with functors of mixed variance, such as the state monad, where interactions between objects in $\mathcal{L}$ and its opposite category present intricate relationships that are best understood through a complete perspective.

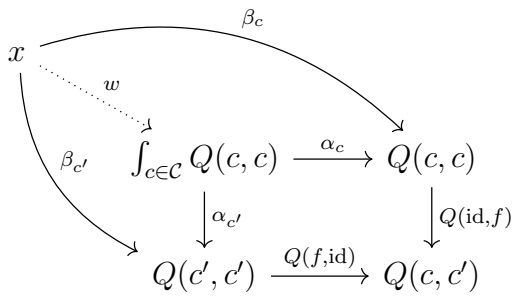
Definition 4.7 (Dinatural transformation)**.** *Given two categories $\mathcal{C}$ and $\mathcal{D}$, and two functors $Q, R : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$, a dinatural transformation $\alpha : Q \dot{\to} R$ consists of a family of components $\alpha_c : Q(c,c) \to R(c,c)$ for all objects c in $\mathcal{C}$, such that the following diagram commutes for all morphisms $f : c \to c'$ in $\mathcal{C}$:*



Definition 4.8 (Wedge)**.** *Given a functor $Q : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$, a wedge from an object $x \in \mathcal{D}$ to the functor Q, in the category of wedges from x to Q, $\mathrm{Wd}(x, Q)$ consists of a family of components $e_c : x \to Q(c,c)$ in $\mathcal{D}$ for all objects c in $\mathcal{C}$ such that the following diagram commutes for all morphisms $f : c \to c'$ in $\mathcal{C}$:*



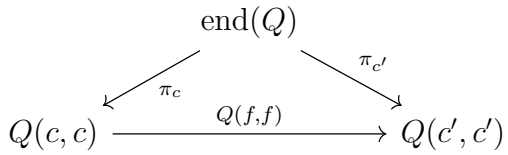
Definition 4.9 (End)**.** *Given a functor $Q : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$, the end of Q, written as $\int_{c \in \mathcal{C}} Q(c,c)$ or $\mathrm{end}(Q)$ is an object in $\mathcal{D}$, such that a wedge in $\mathrm{Wd}(\mathrm{end}(Q), Q)$ is a terminal wedge, with as universal property that for any other wedge $\mathrm{Wd}(x, Q)$ the following diagram commutes for a unique morphism $w : x \to \int_{c \in \mathcal{C}} Q(c,c)$ for all morphisms $f : c \to c'$ in $\mathcal{C}$:*



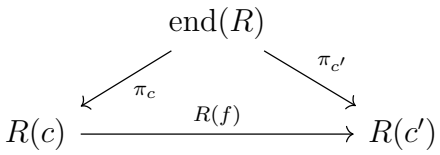
In Definition 4.9, we define an end of the functor $Q : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$, which is a two-variable functor. It is also possible to take an end of a one-variable functor. Consider the functor $R : \mathcal{C} \to \mathcal{D}$. The end of R, $\int_{c\in\mathcal{C}} R(c)$ is then an object in $\mathcal{D}$ such that the wedge $w : \mathrm{end}(R) \to R$ is universal. In Definition 4.8, we see the wedge defined for a two-variable functor, but how is the wedge for a one-variable functor then defined? If we see the functor $Q : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathcal{D}$ as a functor in which we keep the first argument constant, we define a wedge in $\mathrm{Wd}(\mathrm{end}(Q), Q)$ as the commutative diagram below, with extranatural components π_c for all objects c in $\mathcal{C}$ and morphisms $f : c \to c'$ in $\mathcal{C}$.

$$\begin{array}{ccc} \mathrm{end}(Q) & \xrightarrow{\pi_c} & Q(c,c) \\ \downarrow{\scriptstyle \pi_{c'}} & & \downarrow{\scriptstyle Q(\mathrm{id},f)} \\ Q(c',c') & \xrightarrow{Q(f,\mathrm{id})} & Q(c,c') \end{array}$$

As we kept the first argument constant, we have that $Q(f, \mathrm{id}) = \mathrm{id}$; therefore, we can consider the wedge as the following limiting cone: $v : \mathrm{end}(Q) \to Q$, such that the diagram below commutes for all components π_c and morphisms $f : c \to c'$ in $\mathcal{C}$.



This informs us that the end of R is the universal cone



We can see the end of R as a generalization of the concept of a limit, so we also write the end of R as $\prod_{c\in\mathcal{C}} Q(c)$.

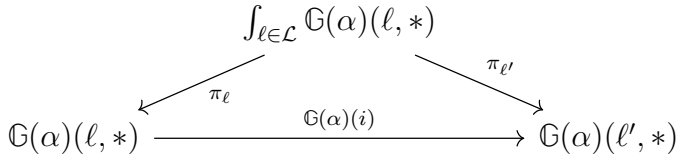
We will now see how the end can be used to get a more global understanding of the constructions we defined in Section 4.2. Recall the natural transformation $\alpha : \Delta_{\mathbb{1}} \Rightarrow P$, the Grothendieck construction of α defines a functor from $\mathbb{G}(\Delta_{\mathbb{1}}) \cong \mathcal{L}$ to $\mathbb{G}(P)$:

$$\mathbb{G}(\alpha) : \mathbb{G}(\Delta_{\mathbb{1}}) \to \mathbb{G}(P)$$

This functor is defined on the objects $(\ell, *)$ in $\mathbb{G}(\Delta_{\mathbb{1}})$ as $\mathbb{G}(\alpha)(\ell, *) = (\ell, F)$ with $F \in P(\ell)$ and on morphisms $(i : \ell \subseteq \ell', \mathrm{id}_*)$ as $\mathbb{G}(\alpha)(i, \mathrm{id}_*) = (i, \phi)$, or just $\mathbb{G}(\alpha)(i)$ for short, with $\phi : P(i)(F) \to G$ and $G \in P(\ell')$.
We can take the end of the Grothendieck construction of α, giving us the following definition:

$$\int_{\ell \in \mathcal{L}} \mathbb{G}\alpha(\ell, *) \cong \prod_{\ell \in \mathcal{L}} [\mathcal{L}/\ell, \mathrm{Endo(Set)}]$$

The universal wedge $w : \int_{\ell \in \mathcal{L}} \mathbb{G}(\alpha(\ell, *) \to \mathbb{G}(\alpha)$ is defined by the commuting diagram below, for all ℓ in $\mathcal{L}$, all inclusion maps $i : \ell \subseteq \ell'$ and components π_ℓ.



Now we will prove a few results using this construction to make life easier, after which we will define the functor that maps objects $i : \ell \subseteq \ell'$ in $\mathcal{L}/\ell$ to the category of monads on Set.

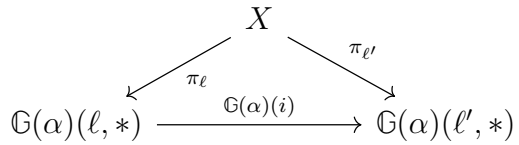
Lemma 4.2. *Given the functor $\mathbb{G}(\alpha) : \mathbb{G}(\Delta_{\mathbb{1}}) \to \mathbb{G}(P)$, a wedge $w : X \to \mathbb{G}(\alpha)$ with X in $\mathbb{G}(P)$ in the category of wedges from X to $\mathbb{G}(\alpha)$, $\mathrm{Wd}(X, \mathbb{G}(\alpha))$ is the same as giving a map from X to $\mathbb{G}(\alpha)(\ell, *)$ in $\mathrm{Set}(X, \mathbb{G}(\alpha)(\ell, *))$:*

$$\mathrm{Wd}(X, \mathbb{G}(\alpha)) \cong \mathrm{Set}(X, \mathbb{G}(\alpha)(\ell, *))$$

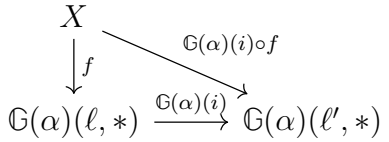
Proof. We define maps

$$\psi : \mathrm{Set}(X, \alpha_\ell(*)) \to \mathrm{Wd}(X, \mathbb{G}(\alpha))$$
$$\chi : \mathrm{Wd}(X, \mathbb{G}(\alpha)) \to \mathrm{Set}(X, \alpha_\ell(*))$$

and show that $\psi \circ \chi = \mathrm{id}_{\mathrm{Wd}(X, \mathbb{G}(\alpha))}$ and $\chi \circ \psi = \mathrm{id}_{\mathrm{Set}(X, \mathbb{G}(\alpha)(\ell, *))}$. First, we show how the map χ is defined. We have that $w : X \to \mathbb{G}(\alpha)$ is defined for all objects ℓ in $\mathcal{L}$, all components π_ℓ and all inclusion maps $i : \ell \subseteq \ell'$ as the commuting diagram below.



We define χ for all wedges w in $\mathrm{Wd}(X, \mathbb{G}(\alpha))$ as $\chi(w) = \pi_\ell$. From $\chi(w)$, we can retrieve w, because the component π_ℓ projects X to $\mathbb{G}(\alpha)(\ell, *)$, where ℓ is the lowest index of all objects in $\mathcal{L}$, and therefore can always extend $\mathbb{G}(\alpha)(\ell, *)$ to $\mathbb{G}(\alpha)(\ell', *)$ for an inclusion $i : \ell \subseteq \ell'$ with $\mathbb{G}(\alpha)(i)$, such that the diagram below commutes.



Therefore, we define ψ for every map f in $\mathrm{Set}(X, \mathbb{G}(\alpha)(\ell, *))$ as $\psi(f) = \mathbb{G}(\alpha)(i) \circ f = w$. Now it remains for us to check if ψ and χ are inverses.

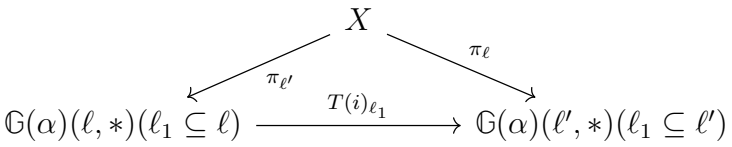
$$\chi(\psi(f)) = \chi(w) = \pi_\ell$$

Since $\mathbb{G}(\alpha)(i) \circ f = \mathbb{G}(\alpha)(i) \circ \pi_\ell$, we know that $f = \pi_\ell$ and therefore $\chi \circ \psi = \mathrm{id}_f$.
We have that $\psi(\chi(w)) = \mathbb{G}(\alpha)(i) \circ f = w$, so $\psi \circ \chi = \mathrm{id}_w$ showing that the isomorphism exists. □

Corollary 4.2.1. *Lemma 4.2 can be extended for any object $\ell_1 \subseteq \ell$ in $\mathcal{L}/\ell$, such that a wedge from X to $\mathbb{G}(\alpha)(\ell, *)$ is the same as giving a map from X to $\mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$. Recall that $\mathbb{G}(\alpha)(\ell, *) \in [\mathcal{L}/\ell, \mathrm{Endo(Set)}]$ and $\mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell) = [M(\ell), M(\ell_1) \times \mathrm{Id}]$. For this, we will show the following:*

$$\mathrm{Wd}(X, \mathbb{G}(\alpha_\ell(*))) \cong \mathrm{Set}(X, \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$$

Proof. A wedge w in $\mathrm{Wd}(X, \mathbb{G}(\alpha_\ell(*)))$ is defined by the commuting diagram below for all objects ℓ in $\mathcal{L}$, all projections π_ℓ in the wedge, and all inclusion maps $i : \ell \subseteq \ell'$ in $\mathcal{L}$.

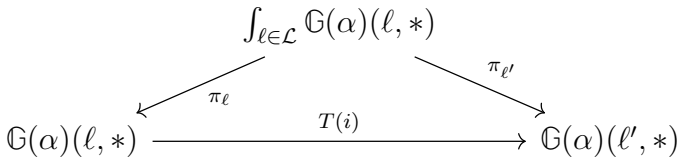


with $T(i)_{\ell_1} = \mathbb{G}(\alpha)(i)(\ell_1 \subseteq \ell)$.
If we analyze how the projections of the wedge are defined, the projection

$$\pi_\ell : \int_{\ell \in \mathcal{L}} \mathbb{G}(\alpha)(\ell_1 \subseteq \ell) \to \mathbb{G}(\alpha)(\ell', *)(\ell_1 \subseteq \ell')$$

for instance maps the functor from $\mathbb{G}(\alpha)(\ell, *)$ to $\mathbb{G}(\alpha)(\ell', *)$, which means the slice category $\mathcal{L}/\ell$ is mapped to the slice category $\mathcal{L}/\ell'$. Therefore, the inclusion object $\ell_1 \subseteq \ell$ gets extended to $\ell_1 \subseteq \ell'$. Again, a map f in $\mathrm{Set}(X, \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$ maps X to $\mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$ where ℓ has the lowest index. As the map $\mathbb{G}(\alpha)(i)(\ell_1 \subseteq \ell)$ is defined by $\phi : P(i)(F) \to G$, with $P(i) = \Sigma_i$, $F \in P(\ell)$ and $G \in P(\ell')$ we see how the Grothendieck construction provides a way to lift the functor $\mathbb{G}(\alpha)(\ell, *)$ to the functor $\mathbb{G}(\alpha)(\ell', *)$ for all ℓ' that have a higher index than ℓ, which means that we can reconstruct the wedge from the map f and that the projection to the lowest index of all ℓ in $\mathcal{L}$ is used to obtain the map f from the wedge. □

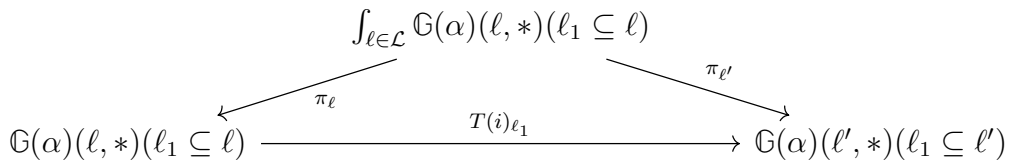
Proposition 4.5. *For an object ℓ in $\mathcal{L}$, taking the end of $\mathbb{G}(\alpha)(\ell, *)$, defined by the commuting diagram below for all objects ℓ in $\mathcal{L}$, inclusions $\ell \subseteq \ell'$ and components $\pi_\ell : \int_{\ell \in \mathcal{L}} \mathbb{G}(\alpha)(\ell, *)$*



induces the functor

$$S : \mathcal{L}^{\mathrm{op}} \to \mathrm{Endo(Set)}$$

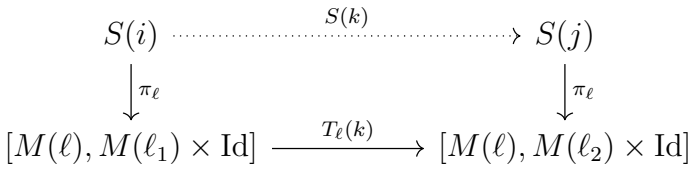
that is defined on all objects ℓ_1 in $\mathcal{L}$ as $S(\ell_1) = \int_{\ell \in \mathcal{L}} \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$, which is the end defined by the commuting diagram below for all components π_ℓ and inclusions $i : \ell \subseteq \ell'$



with $T(i)_{\ell_1} = \mathbb{G}(\alpha)(i)(\ell_1 \subseteq \ell)$.
On all morphisms $k : \ell_2 \subseteq \ell_1$ *in* $\mathcal{L}$, S *is defined as*

$$S(k) : S(i) \to S(j)$$

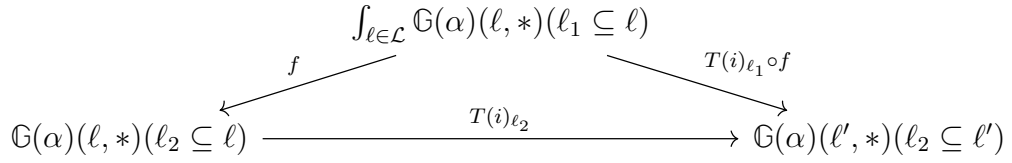
with $j : \ell_2 \subseteq \ell$ *in* $\mathcal{L}$ *and* $S(j) = \int_{\ell\in\mathcal{L}}[M(\ell), M(\ell_2) \times \mathrm{Id}]$ *summarized in the diagram below.*



Proof. First, we have to show that for morphisms $k : \ell_2 \subseteq \ell_1$ the map $S(k)$ exists for objects ℓ_1, ℓ_2 and the morphism k in $\mathcal{L}$. By Lemma 4.2 the map

$$f = \mathbb{G}(\alpha)(\ell,*)(k) \circ \pi_\ell : \int_{\ell\in\mathcal{L}} \mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell) \to \mathbb{G}(\alpha)(\ell,*)(\ell_2 \subseteq \ell)$$

induces the following wedge



with $T(i)_{\ell_2} = \mathbb{G}(\alpha)(\ell,*)(\ell_2 \subseteq \ell)$.
By the universal property of the end, the unique morphism $S(k) : S(\ell_1) \to S(\ell_2)$ is induced, such that $S(\ell_1)$ factors through the universal wedge of $S(\ell_2)$ via this map.
Now, we still have to show that S preserves identity:

$$\mathrm{id}_{S(\ell_1)} = S(\mathrm{id}_{\ell_1})$$

Nevertheless, as $\mathrm{id}_{S(\ell_1)}$ respects the extranatural components of $S(\ell_1)$ per definition and $S(\mathrm{id}_{\ell_1})$ is uniquely induced by the universal property of the end such that it respects the extranatural components of $S(\ell_1)$, they are the same. We also have to show that S preserves composition for morphisms $k : \ell_2 \subseteq \ell_1$ and $m : \ell_3 \subseteq \ell_2$

$$S(m) \circ S(k) = S(m \circ k)$$

with $S(m \circ k) : S(\ell_1) \to S(\ell_3)$ and $S(m) \circ S(k) : S(\ell_1) \to S(\ell_3)$.
Similarly, this follows from uniqueness of the maps $S(m) \circ S(k)$ and $S(m \circ k)$. □

The intuition behind S is that instead of having a state monad such as $T_\ell(i : \ell' \subseteq \ell) = [M(\ell), M(\ell') \times \mathrm{Id}]$ that only evaluates memory operations on the memory of ℓ we now can project to all ℓ in $\mathcal{L}$ via the projections of S to evaluate the memory operations at all these subsets of the whole memory. Furthermore, the UMP of the end provides a unique way to restrict the memory of the output in $M(\ell')$ to all subsets of memory that are included in ℓ', with ℓ' in $\mathcal{L}$.

Proposition 4.6. *The functor S is an endofunctors-in-*Set*-valued presheaf on $\mathcal{L}$*

Proof. First we have to see how $S(\ell_1) : \mathrm{Set} \to \mathrm{Set}$ is defined, where ℓ_1 is an object in $\mathcal{L}$. We define $S(\ell_1)$ for all objects X in Set as $S(\ell_1)(X) = \int_{\ell \in \mathcal{L}} \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)(X)$, which is defined by the commuting diagram below for inclusions $i : \ell \subseteq \ell'$.

$$\begin{array}{ccc} & S(\ell_1)(X) = \int_{\ell \in \mathcal{L}} [M(\ell), M(\ell_1) \times X] & \\ \swarrow^{\pi_\ell} & & \searrow^{\pi_{\ell'}} \\ \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)(X) & \xrightarrow{T(i)_{\ell_1, X}} & \mathbb{G}(\alpha)(\ell', *)(\ell_1 \subseteq \ell')(X) \end{array}$$

with $T(i)_{\ell_1, X} = \mathbb{G}(\alpha)(i)(\ell_1 \subseteq \ell)(X) : [M(\ell), M(\ell_1) \times X] \to [M(\ell'), M(\ell') \times X]$
We define $S(\ell_1)$ on all morphisms $f : X \to Y$ in Set as

$$S(\ell_1)(f) : S(\ell_1)(X) \to S(\ell_1)(Y)$$

Again, Lemma 4.2 together with the UMP of the end induces the unique morphism $S(\ell_1)(f)$ via the map

$$\mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)(f) \circ \pi_\ell : \int_{\ell \in \mathcal{L}} \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)(X) \to \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)(Y)$$

That $S(\ell_1)$ preserves identity and compositions follows from the uniqueness of this map (we have seen this before), which proves functoriality of $S(\ell_1)$.

Finally, we have to show that for a morphism $k : \ell_2 \subseteq \ell_1$ in $\mathcal{L}$ the map

$$S(k) : S(\ell_1) \to S(\ell_2)$$

is a natural transformation, which is the case if the diagram below commutes for all morphisms $f : X \to Y$ in Set.

$$\begin{array}{ccc} S(\ell_1)(X) & \xrightarrow{S(\ell_1)(f)} & S(\ell_1)(Y) \\ \downarrow{\scriptstyle S(k)_X} & & \downarrow{\scriptstyle S(k)_Y} \\ S(\ell_2)(X) & \xrightarrow{S(\ell_2)(f)} & S(\ell_2)(Y) \end{array}$$

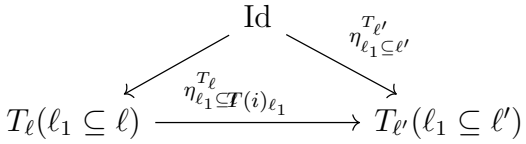
For this, we need to show that the compositions $S(\ell_2)(f) \circ S(k)_X$ and $S(k)_Y \circ S(\ell_1)(f)$ are equal. One only has to show these compositions are uniquely induced with respect to the components of the wedge from $S(\ell_1)(X)$ to $\mathbb{G}(\alpha)(\ell, *)_X$ using Lemma 4.2 together with the UMP of the end, which proves the compositions are equal. Note that the functor $\mathbb{G}(\alpha)(\ell, *)_X$ sends objects $\ell_1 \subseteq \ell$ in $\mathcal{L}/\ell$ to $\mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)(X)$ for an object X in Set. □

4.5 Proofs of Monad Structure

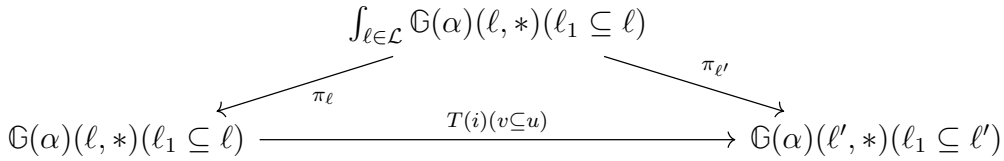
We now have defined constructions that we suspect can serve as a traditional state monad, the endofunctor $T_\ell(\ell_1 \subseteq \ell) = \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$, and a local state monad, the endofunctor $S(\ell_1)$ with ℓ_1 in $\mathcal{L}$ and $\ell_1 \subseteq \ell$ in $\mathcal{L}/\ell$. We still have to define units η^{T_ℓ} and η, and multiplications μ^{T_ℓ} and μ for T_ℓ and S respectively and prove that the monad axioms hold.

Proposition 4.7. *For an object ℓ_1 in $\mathcal{L}$, $S(\ell_1)$ is a monad and the maps $\eta : \mathrm{Id} \to S(\ell_1)$ and $\mu : S^2(\ell_1) \to S(\ell_1)$ exist and serve as the unit and multiplication of $S(\ell_1)$ respectively.*

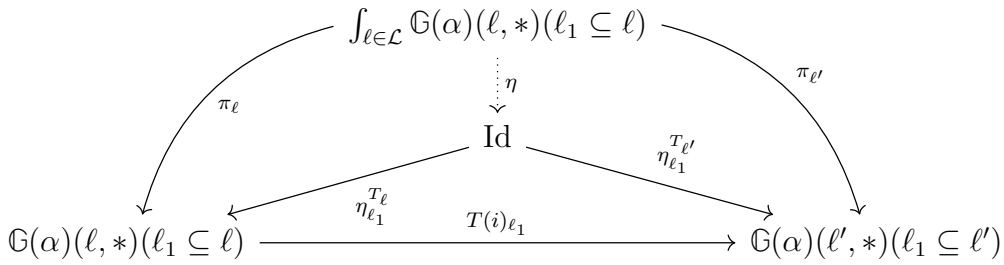
Proof. First, we will show that the unit $\eta : \mathrm{Id} \to S(\ell_1)$ exists. Given the functor T_ℓ, we define the global state monad as $T_\ell(\ell_1 \subseteq \ell)$, with a unit $\eta^{T_\ell} : \mathrm{Id} \to T_\ell(\ell_1 \subseteq \ell)$. By Lemma 4.2 this map induces a wedge from Id to T_ℓ such that the diagram below commutes



for all objects ℓ in $\mathcal{L}$, inclusions $i : \ell \subseteq \ell'$ and components $\eta^T_{\ell_1 \subseteq \ell} : \mathrm{Id} \to T_\ell(\ell_1 \subseteq \ell)$ and with $T(i)_{\ell_1} = \mathbb{G}(\alpha)(i)(\ell_1 \subseteq \ell)$.
The end $S(\ell_1)$ is defined as the following universal wedge:



By the UMP of the end, there exists a unique map from Id to $S(\ell_1)$ such that the diagram below commutes:



We define the unit

$$\eta : \mathrm{Id} \to S(\ell_1)$$

to be this unique map.

Given the map $\pi_\ell * \pi_\ell : S^2(\ell_1) \to (\mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell))^2$, where $*$ denotes the Godement product of the projection π_ℓ of the end $S(\ell_1)$, and given the map of the global state monad

$$\mu^{T_\ell}_{\ell_1} : (\mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell))^2 \to \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$$

the map

$$\mu_{\ell_1}^{T_\ell} \circ (\pi_\ell * \pi_\ell) : S(\ell_1)^2 \to \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$$

induces a wedge from $S(v \subseteq u)^2$ to $\mathbb{G}(\alpha)(\ell, *)$ by Lemma 4.2. As we have seen with the unit, this wedge entails the existence of the unique map from $S^2(\ell_1)$ to $S(\ell_1)$, which respects the extra natural components of $S(\ell_1)$. We define the multiplication

$$\mu : S(\ell_1)^2 \to S(\ell_1)$$

to be this map.

Now, given this unit and multiplication, we have to prove the unit axioms, for which we need to show the four diagrams below commute:

$$\begin{array}{ccc} S(\ell_1) & \xrightarrow{\eta S(\ell_1)} & S^2(\ell_1) \\ & \searrow^{\mathrm{id}} & \downarrow \mu \\ & & S(\ell_1) \end{array} \qquad \begin{array}{ccc} S(\ell_1) & \xrightarrow{S(\ell_1)\eta} & S^2(\ell_1) \\ & \searrow^{\mathrm{id}} & \downarrow \mu \\ & & S(\ell_1) \end{array}$$

$$\begin{array}{ccc} S^3(\ell_1) & \xrightarrow{\mu S(\ell_1)} & S^2(\ell_1) \\ \downarrow S(\ell_1)\mu & & \downarrow \mu \\ S^2(\ell_1) & \xrightarrow{\mu} & S(\ell_1) \end{array} \qquad \begin{array}{ccc} S^2(\ell_1)(X) & \xrightarrow{S^2(\ell_1)(f)} & S^2(\ell_1)(Y) \\ \downarrow \mu_X & & \downarrow \mu_Y \\ S(\ell_1)(X) & \xrightarrow{S(\ell_1)(f)} & S(\ell_1)(Y) \end{array}$$

For the first diagram, as both η and μ respect the extra-natural components of the end $S(\ell_1)$, we have that the composition of these maps respects this extra-natural transformation as well. Per definition $\mathrm{id}_{S(\ell_1)}$ also respects these components, so we have that $\mathrm{id} = \mu \circ \eta S(\ell_1)$ proving the left unit law.

By Lemma 4.2 the map

$$(\pi_\ell * \pi_\ell) \circ S(\ell_1)\eta : S(\ell_1) \to (\mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)_\ell(*))^2$$

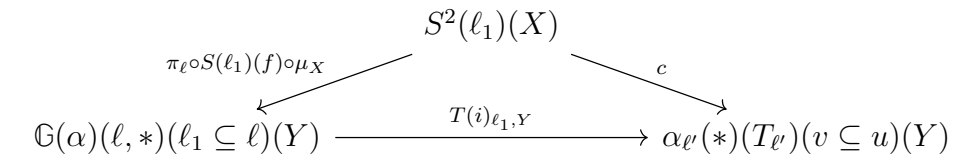
induces a wedge from $S(\ell_1)$ to $(\mathbb{G}(\alpha)(\ell, *))^2$. By the UMP of the end, the map $S(\ell_1)\eta$ is then unique with respect to the extra natural components, showing that $S(\ell_1)\eta = \eta S(\ell_1)$ and thereby proving the right unit law.

The same proof pattern can be used to prove the equivalence of $\mu S(\ell_1)$ and $S(\ell_1)\mu$ to show that $\mu \circ \mu S(\ell_1) = \mu \circ S(\ell_1)\mu$.

Now for the last diagram, by Lemma 4.2 the map

$$\pi_\ell \circ S(\ell_1)(f) \circ \mu_X : S(\ell_1)^2 \to \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)(Y)$$

induces a wedge from $S^2(\ell_1)(X)$ to $T_{\ell,Y}$ defined by the following commuting diagram



where $c = T(i)_{\ell_1,Y} \circ \pi_\ell \circ S(\ell_1)(f) \circ \mu_X$. The map

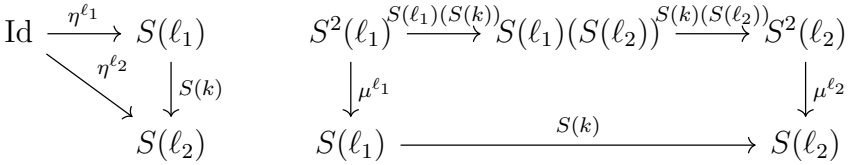
$$\pi_\ell \circ \mu_Y \circ S^2(\ell_1)(f) : S^2(\ell_1)(X) \to \mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(Y)$$

also induces a wedge from $S^2(\ell_1)(X)$ to $\mathbb{G}(\alpha)(\ell,*)(Y)$. This entails that the maps $S(\ell_1)(f) \circ \mu_X$ and $\mu_Y \circ S^2(\ell_1)(f)$ are both unique with respect to the extra natural components of these wedges and therefore they must be equal, proving naturality of μ.

□

Proposition 4.8. *Given objects ℓ_1 and ℓ_2 and a morphism $k : \ell_2 \subseteq \ell_1$ in $\mathcal{L}$ the map $S(k) : S(\ell_1) \to S(\ell_2)$ is a monad morphism.*

Proof. For a monad $S(\ell_1)$ with unit $\eta^{\ell_1} : \mathrm{Id} \to S(\ell_1)$ and multiplication $\mu^{\ell_1} : S^2(\ell_1) \to S(\ell_1)$, and a monad $S(\ell_2)$ with unit $\eta^{\ell_2} : \mathrm{Id} \to S(\ell_2)$ and multiplication $\mu^{\ell_2} : S^2(\ell_2) \to S(\ell_2)$, we have to show the diagrams below commute.



For the first diagram, we have by Lemma 4.2 and the UMP of the end that the map

$$\pi_\ell \circ S(k) \circ \eta^{\ell_1} : \mathrm{Id} \to \mathbb{G}(\alpha)(\ell,*)(\ell_2 \subseteq \ell)$$

shows that $S(k) \circ \eta^{\ell_1}$ is unique with respect the the extra natural components of the end $S(\ell_2)$. Similarly, $\pi_\ell \circ \eta^{\ell_2} : \mathrm{Id} \to S(\ell_2)$ shows the map η^{ℓ_2} is unique with respect to the same components, proving the maps are equal.

For the second diagram, we first have to see how the map $S(\ell_1)(S(k))$ is defined. We have that $S^2(\ell_1) = \int_{\ell \in \mathcal{L}}[M(\ell), M(\ell_1) \times \int_{\ell \in \mathcal{L}}[M(\ell), M(\ell_1) \times \mathrm{Id}]]$.
The projection π_ℓ maps $S^2(\ell_1)$ to $[M(\ell), M(\ell_1) \times \int_{\ell \in \mathcal{L}}[M(\ell), M(\ell_1) \times \mathrm{Id}]]$, which can be denoted as $\mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(S(\ell_1))$. As we have seen before, the map $f : X \to Y$ in Set induces the map $\mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(f) : \mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(X) \to \mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(Y)$. Now, we have the map

$$S(k) : S(\ell_1) \to S(\ell_2)$$

which induces the map

$$\mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(S(k)) : \mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(S(\ell_1)) \to \mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(S(\ell_2))$$

Now we see how we can use Lemma 4.2 together with the UMP of the end to show that the map $\mathbb{G}(\alpha)(\ell,*)(\ell_1 \subseteq \ell)(S(k)) \circ \pi_\ell$ induces the unique map $S(\ell_1)(S(k))$ with respect to the extranatural

components of $S(\ell_1)(S(\ell_2))$. By Lemma 4.2 and the UMP of the end, the map $T_\ell(k)(S(\ell_2)) \circ \pi_\ell : S(\ell_1)(S(\ell_2)) \to \mathbb{G}(\alpha)(\ell, *)(\ell_2 \subseteq \ell)(S(\ell_2))$ induces the unique map $S(k)(S(\ell_2))$ with respect to the components of $S^2(\ell_2)$. The maps μ^{ℓ_1} and μ^{ℓ_2} are unique with respect to the components of $S(\ell_1)$ and $S(\ell_2)$ respectively. That $S(k)$ is unique with respect to the components of $S(\ell_2)$ we have seen in the proof of the first diagram. Now we use the fact that the composition $S(k) \circ \mu^{\ell_1}$ is unique with respect to the components of $S(\ell_2)$ and the composition $\mu^{\ell_2} \circ S(k)(S(\ell_2)) \circ S(\ell_1)(S(k))$ is unique with respect to the components of $S(\ell_2)$, proving that the diagram commutes. □

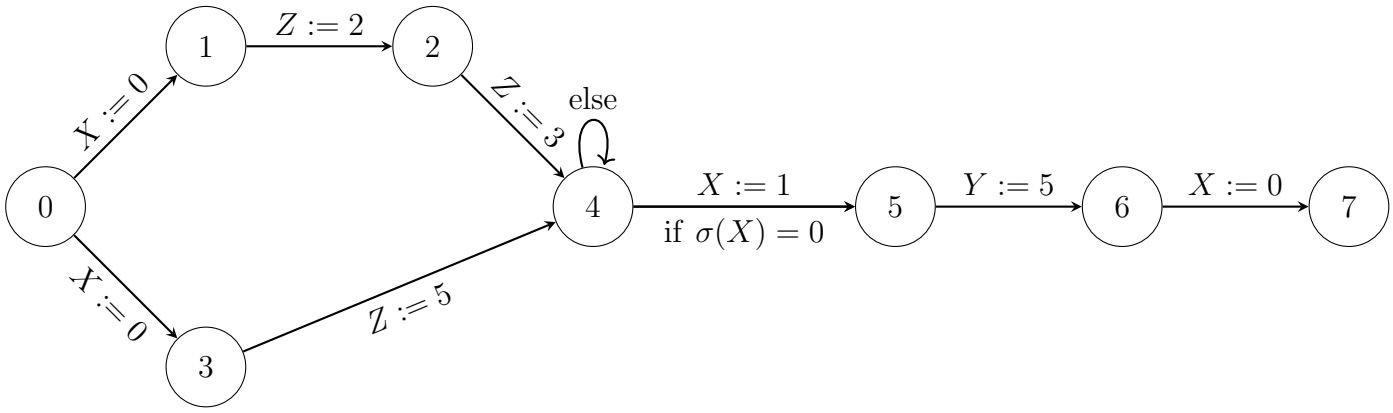
4.6 Modelling Stateful Computation

Now that we have defined a construction for the local state monad, we will analyse which insights the construction can give us on reasoning about programs interacting with memory from a local and global perspective. First, we need to see how we can reason about programs in a category theoretical way. For this, we need a construction that can compose computation steps of a program that interact with memory, which will be represented as morphisms in a certain category. With these program steps, computational side effects arise, so our construction must be able to handle these. We still need to keep abstraction to hide details of specific programs and memory configurations. For this, we will use the Kleisli category introduced in Section 4.1.
In the papers of Plotking and Power [KK13] and Beohar and co-authors [BKKS16], examples are given of how the Kleisli category can be used to model transition systems and incorporate side effects produced by a monad. In our following example, we will model a concurrent stateful computation, where states represent values stored in memory locations at a particular moment in the computation, and transitions represent the execution of instructions that modify memory locations. Each memory location can be associated with an integer value. We use Kleisli morphisms to capture the transitions between memory configurations and represent computations.

Example 4.1. *Consider the labelled transition system L, where $L = (Q, \Lambda, T)$.*
The set of States Q is in $\mathcal{P}[7]$. Labels represent assignments to memory using maps in $M(\mathrm{Loc}) = [\mathrm{Loc}, \mathbb{Z}]$, associated with transitions between states in the system, for $\mathrm{Loc} = \{X, Y, Z\}$ with $\mathcal{P}(\mathrm{Loc})$ being the set of objects in $\mathcal{L}$. Labelled transition relations T with $(p, a, q) \in T$, $a \in \Lambda$, and $p, q \in Q$ are the set of computations

$T = \{(0, X := 0, 1), (0, X := 0, 3), (1, Z := 2, 2), (2, Z := 3, 4), (3, Z := 5, 4), (4, X := 1, 5),$
$(5, Y := 5, 6), (6, X := 0, 7)\}$

Pictorially, this labelled transition system is represented by the diagram below.



Transition relations are modeled by the Kleisli morphisms $c \in Kl(S(\mathrm{Loc}))$, defined by the functor $S : \mathcal{L}^{\mathrm{op}} \to \mathrm{Mnd}(\mathrm{Set})$, defined on objects ℓ_1 in $\mathcal{L}$ as the end $\int_{\ell \in \mathcal{L}} \mathbb{G}(\alpha)(\ell, *)(\ell_1 \subseteq \ell)$, summarized by the universal wedge below for inclusions $i : \ell \subseteq \ell'$ with the map $T(i)_{\ell_1} = \mathbb{G}(\alpha)(i)(\ell_1 \subseteq \ell)$

$$\begin{array}{ccc} & S(\ell_1) & \\ {\scriptstyle \pi_\ell}\swarrow & & \searrow{\scriptstyle \pi_{\ell'}} \\ [M(\ell), M(\ell_1) \times \mathrm{Id}]] & \xrightarrow{T(i)_{\ell_1}} & [M(\ell'), M(\ell_1) \times \mathrm{Id}] \end{array}$$

The type of the Kleisli map c is

$$c : [7] \to S(\mathrm{Loc})(\mathcal{P}[7])$$

The state transitions that assign values to memory are modelled by a map $\sigma \in M(\mathrm{Loc}) = [\mathrm{Loc}, \mathbb{Z}]$ such that for a map σ in $M(\mathrm{Loc})$ and the state $1 \in Q$, we have that

$$c(1)(\sigma) = (\sigma[Z \mapsto 2], \{2\})$$

where $\sigma[Z \mapsto 2]$ models an update of the memory in Z. Now, we will see how the computations proceed.

Starting from state 0:

$$c(0)(\sigma) = (\sigma[X \mapsto 0], \{1, 3\})$$

Now, we have two possible transitions from state 0:

1. *Apply label $X := 0$ and transition to state 1 with the updated memory $\sigma[X \mapsto 0]$*
2. *Apply label $X := 0$ and transition to state 3 with the updated memory $\sigma[X \mapsto 0]$*

Next:

$$c(1)(\sigma) = (\sigma[Z \mapsto 2], \{2\})$$

From state 1, we have a single transition:
Apply label $Z := 2$ and transition to state 2 with the updated memory $\sigma[Z \mapsto 2]$.

Next:

$$c(2)(\sigma) = (\sigma[Z \mapsto 3], \{4\})$$

From state 2*, we have a single transition:*
Apply label $Z := 3$ *and transition to state* 4 *with the updated memory* $\sigma[Z \mapsto 3]$*.*

Next:

$$c(3)(\sigma) = (\sigma[Z \mapsto 5], \{4\})$$

From state 3*, we have a single transition:*
Apply label $Z := 5$ *and transition to state* 4 *with the updated memory* $\sigma[Z \mapsto 5]$*.*

Next:

$$c(4)(\sigma) = (\sigma[X \mapsto 1], \{5\}) \text{ if } \sigma(X) = 0$$
$$c(4)(\sigma) = (\sigma, \{4\}) \text{ if } \sigma(X) \neq 0$$

From state 4*, we have two possible transitions:*

1. *If* $\sigma(X) = 0$*, then apply label* $X := 1$ *and transition to state* 5 *with the updated memory* $\sigma[X := 1]$*.*
2. *If* $\sigma(X) \neq 0$*, then no memory update occurs, and we stay in state* 4 *with the same memory configuration.*

Next:

$$c(5)(\sigma) = (\sigma[Y \mapsto 5], \{6\})$$

From state 5*, we have a single transition:*
Apply label $Y := 5$ *and transition to state* 6 *with the updated memory* $\sigma[Y \mapsto 5]$*.*

Finally:

$$c(6)(\sigma) = (\sigma[X \mapsto 0], \{7\})$$

From state 6*, we have a single transition:*
Apply label $X := 0$ *and transition to state* 7 *with the updated memory* $\sigma[X \mapsto 0]$*.*

There are some important things to take note of in this example. The Kleisli composition lets us combine two Kleisli morphisms to obtain a new one. In the context of transition relations, this enables us to compose different transitions to get a more complex computation. Suppose we have two Kleisli morphisms:

$$c(0) : [7] \to S(\text{Loc})(\{1, 4\})$$
$$c(1) : [7] \to S(\text{Loc})(\{3\})$$

To compute the transition from state 0 to state 3, we can use the Kleisli composition. It involves applying the map $c(1)$ to all the local memory locations generated by $c(0)$. This combines the transition specified by $c(0)$ and $c(1)$ to obtain a new transition from state 0 to state 3. Secondly, the end in the monad $S(\ell_1) = [\int_{\ell \in \mathcal{L}} M(\ell), M(\ell_1) \times \mathrm{Id}]$ with ℓ_1 in $\mathcal{L}$ allows us to switch between local and global computation. The first argument of $S(\ell_1)$ represents a collection of local memory configurations, i.e., configurations of all subsets of memory locations $\ell \in \mathcal{L}$. The second index of $S(\ell_1)$ represents the part of the memory that can be operated on by the maps specified by $M(\ell_1)$ after transitioning between states. By using the end to vary the memory in the first index, applying a map $\sigma \in M(\ell)$ to $S(\ell_1)$, we are performing a local computation. It means we consider the possible transitions for each local memory configuration and, after the state transition, specify possible memory operations specified by the fixed part of the memory location ℓ_1. If we moved the end to the second index, we would only update the state based on the fixed part of the memory global memory when applying σ to $S(\ell_1)$, potentially leading to a more restricting computation, especially when the part of the memory specified by ℓ_1 does not include the memory that is needed to update the values between state transitions. If we consider Example 4.1 with the monad in which we take the end over the output maps and specify the fixed part of the memory for the input maps to be $\ell_1 = \{Y, Z\}$, we do not know how the computation $c(0)(\sigma)$, with

$$c : [7] \to \left[M(\ell_1 = \{Y, Z\}), \int_{\ell \in \mathcal{L}} M(\ell) \times \mathcal{P}([7]) \right]$$

because this computation step requires access to the memory described by X.

5 (Co-)Sheaves of Kleisli Morphisms

Now that we have seen how the Kleisli category can be used to model stateful computations, we can also find a way to intuitively comprehend the frame rule in the context of Kleisli morphisms: we can consider objects ϕ in the Kleisli category $Kl(S(\ell))$ as a set of assertions that hold for a particular memory location ℓ. Morphisms in this Kleisli category can be seen as program effects modifying memory stored at ℓ. Given an assertion ϕ for some part of the heap ℓ, and twp disjoint memory locations ℓ_1 and ℓ_2, if we execute a program that only touches the memory in ℓ_1, we can frame the formula ϕ, with the programs effect and obtain a new formula ϕ' over the combined memory locations $\ell_1 \cup \ell_2$, such that ϕ' holds over the entire combined memory. The essence of this perspective is that we "glue" local states back together to obtain a global state over which we can reason with the observations from the local states. Sheaves are a mathematical abstraction that can provide this property.

Definition 5.1 (Presheaf)**.** *Let X be a topological space. A presheaf F on X functor from the category of open sets to* Set*:*

$$F : X^{\mathrm{op}} \to \mathrm{Set}$$

For every open set U in X, there is an associated set $F(U)$, and from every inclusion map $V \subseteq U$ of open sets in X, there is a restriction map $\rho_V^U : F(U) \to F(V)$, such that ρ respects identity and compositions.

Definition 5.2 (Sheaf)**.** *Let X be a topological space. A sheaf F is a presheaf that satisfies the following two additional properties:*

1. ***Locality (Identity Axiom):*** *for every open set U in X and every open cover $\{U_i\}_{i\in I}$ of U, if two sections $s, t \in F(U)$ agree on each U_i, meaning that for all $i \in I$ $\rho^U_{U_i}(s) = \rho^U_{U_i}(t)$, then $s = t$.*

2. ***Gluing (Gluability Axiom):*** *for every open set U in X and every open cover $\{U_i\}_{i\in I}$ of U, if there exists a collection of elements $\{s_i \in F(U_i)\}$ such that they agree on overlaps, meaning that for any U_i and U_j with $i, j \in I$ and $U_i \cap U_j \neq \emptyset$, $\rho^{U_i}_{U_i\cap U_j}(s_i) = \rho^{U_j}_{U_i\cap U_j}(s_j)$, then there exists a unique element $s \in F(U)$ such that for all $i \in I$, the restriction of s to U_i coincides with s_i, meaning that $\rho^U_{U_i}(s) = s_i$.*

The gluing axiom is captured by the isomorphism between $F(U)$ and the limit of the presheaf F over the directed set of open covers of U [Cur14]:

$$F(U) \xrightarrow{\simeq} \lim(\prod_{k\in I} F(U_k) \rightrightarrows \prod_{i,j\in I} F(U_i \cap U_j))$$

Proposition 5.1. *Given the Kleisli category associated with a monad $S(\ell)$, $Kl(S(\ell))$, with $\ell \in \mathcal{L}$, the underlying presheaf of Kleisli morphisms induced by composition with $S : \mathcal{L}^{\text{op}} \to \text{Mnd(Set)}$, $Kl(X, Y) : \text{Mnd(Set)} \to \text{Set}$, which are of the form $c : X \to S(\ell)(X) \in Kl(X, Y)$, with X, Y in* Set *is a sheaf.*

Proof. First we check if $Kl(X, Y)$ is well-defined: Given $N_1, N_2 \in \text{Mnd(Set)}$, with monad morphism $\epsilon : N_1 \to N_2$, we have that $Kl(X, Y)(N_1) = Kl(N_1)(X, Y)$, and $Kl(X, Y)(\epsilon)$ is given by the composition $Kl(X, Y)(\epsilon)(c : X \to N_1(\ell)) = \epsilon_Y \circ c$. Now, we need to formulate a sheaf condition. For each $\ell \in \mathcal{L}$ and for each open covering of ℓ denoted as $\{\ell_i\}_{i\in I}$ the sheaf condition requires the following:

For any compatible family of Kleisli morphisms $\{c_i \in Kl(S(\ell_i))(X, Y)\}_{i\in I}$, where compatible means that for any pair of indices $i, j \in I$, the restriction of c_i and c_j to $(\ell_i \cap \ell_j)$ are the same:

$$c_i|_{(\ell_i\cap\ell_j)} = c_j|_{(\ell_i\cap\ell_j)}$$

then there exists a unique Kleisli morphism $c \in Kl(S(\ell))(X, Y)$ such that for each $i \in I$ the restriction of c to ℓ_i gives back the respective c_i.

We proceed as follows. We have that $\ell_i \cap \ell_j \subseteq \ell_i \subseteq \ell$ and $\ell_i \cap \ell_j \subseteq \ell_j \subseteq \ell$. The end $S((\ell_i \cap \ell_j))$ is the following universal wedge. For clarity, we write ℓ_o for $(\ell_i \cap \ell_j)$.

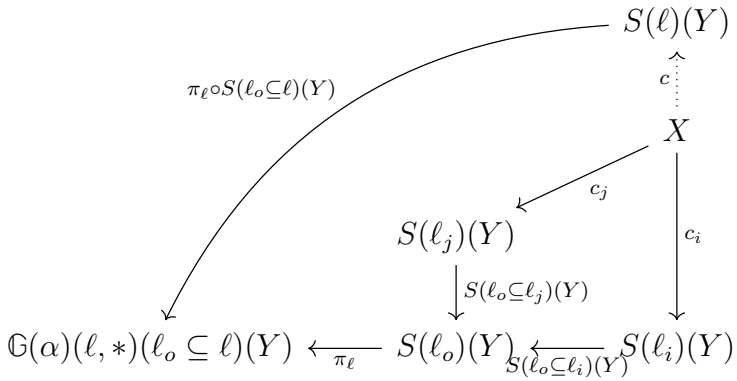
$$\begin{array}{ccc} & \int_{\ell\in\mathcal{L}}[M(\ell), M(\ell_o) \times Y] & \\ {\scriptstyle \pi_\ell}\swarrow & & \searrow{\scriptstyle \pi_{\ell'}} \\ [M(\ell), M(\ell_o) \times Y] & \xrightarrow{T(i)_{\ell_o,Y}} & [M(\ell'), M(\ell_o) \times Y] \end{array}$$

First, we prove existence and uniqueness of the map $c : X \to S(\ell)(Y)$. The map $\pi_\ell \circ S(\ell_o \subseteq \ell)_Y : S(\ell)(Y) \to \mathbb{G}(\alpha)(\ell, *)(\ell_o \subseteq \ell)$ induces a wedge from $S(\ell)(Y)$ to $\mathbb{G}(\alpha)(\ell, *)_Y$. By definition of the end, this wedge is universal. Now we show the wedge $w : X \to \mathbb{G}(\alpha)(\ell, *)$ exists.

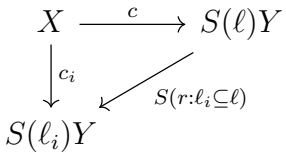
By assumption, we have that for any pair of indices $i, j \in I$, we have that

$$c_i|_{\ell_o} = c_j|_{\ell_o}$$

with $c_i : X \to S(\ell_i)(Y)$ and $c_j : X \to S(\ell_j)(Y)$. We also have maps $\pi_\ell \circ c_i : X \to \mathbb{G}(\alpha)(\ell, *)(\ell_i \subseteq \ell)(Y)$ for all $i \in I$ and by Lemma 4.2, this induces the wedge $w : X \to \mathbb{G}(\alpha)(\ell, *)_Y$. Now, by the UMP of the end, we have that this wedge factors through the universal wedge from $S(\ell)(Y)$ to $\mathbb{G}(\alpha)(\ell, *)_Y$ via the unique map $c : X \to S(\ell)(Y)$. This proof is summarised by the commuting diagram below:



We still have to prove the sheaf condition holds, meaning that $c_i = c|_{\ell_i}$ for all $i \in I$, for which the following diagram needs to commute:



for morphisms $r : \ell_i \subseteq \ell$ in $\mathcal{L}$ for all $i \in I$. The maps $S(r) \circ c$ and c_i are both unique with respect to the extra-natural components of $S(\ell_i)_Y$, showing that $c_i = S(r) \circ c$ thereby proving the sheaf condition holds.

□

Remark. *Another perspective that might be worth investigating is, instead of restricting memory in M, extending memory by letting M be covariant and constructing a (pre-)cosheaf construction. One can then extract a compatibility axiom by looking at the isomorphic colimit of this cosheaf, which we denote as $\hat{S}$*

$$\operatorname{colim}(\coprod_{i,j \in I} \hat{S}(\ell_i \cup \ell_j) \rightrightarrows \to \coprod_{k \in I} \hat{S}(\ell_k)) \xrightarrow{\simeq} \hat{S}(\ell)$$

which is the coproduct of the extension maps for the inclusion of each ℓ_i into ℓ and where the arrows are the coproducts of extension maps for the inclusion of $\ell_i \cup \ell_j$ into ℓ_i and ℓ_j, respectively.

Another way to shift the perspective is by looking at how the local state monad is defined by Plotkin and Power [PP02]. Instead of using an end to define the local state monad, they use a coend, which is dual to the notion of the end.

6 Internal Categories

Given the functor $S_\ell : (\mathcal{L}/\ell)^{\mathrm{op}} \to \mathrm{Mnd}(\mathrm{Set})$ we obtain a presheaf of monads (or a pre-cosheaf for the functor $\hat{S}$). The category of monads does not form a concrete category, so how do we formulate the sheaf condition? We can solve this problem by considering the category $\mathrm{Sh}(\mathcal{L})$ of set-valued sheaves over $\mathcal{L}$ and then define functors and monads internally to $\mathrm{Sh}(\mathcal{L})$. First, we will show how these structures can be defined internally for general categories, then define the structures for the specific category $\mathrm{Sh}(\mathcal{L})$.

6.1 General internal structures

Definition 6.1 (Internal Category)**.** *A category $\mathcal{C}$ internal to an ambient category $\mathbb{B}$ consists of an object of objects $\mathcal{C}_0$ and an object of morphisms $\mathcal{C}_1$, together with target and source maps $s : \mathcal{C}_1 \to \mathcal{C}_0$, $t : \mathcal{C}_1 \to \mathcal{C}_0$, and the identity map $e : \mathcal{C}_0 \to \mathcal{C}_1$, such that the following diagram commutes:*

$$\begin{array}{ccc} \mathcal{C}_0 & \xrightarrow{e} & \mathcal{C}_1 \\ \downarrow{\scriptstyle e} & \searrow & \downarrow{\scriptstyle s} \\ \mathcal{C}_1 & \xrightarrow{t} & \mathcal{C}_0 \end{array}$$

One can think of $\mathcal{C}_0$ as the objects of $\mathcal{C}$ and of $\mathcal{C}_1$ as the morphisms as $\mathcal{C}$.
The composition of morphisms is defined on the pullback $M_{\mathcal{C}} = s \times_{\mathcal{C}_0} t$:

$$\begin{array}{ccc} M_{\mathcal{C}} & \xrightarrow{p_1} & \mathcal{C}_1 \\ \downarrow{\scriptstyle p_2} & & \downarrow{\scriptstyle t} \\ \mathcal{C}_1 & \xrightarrow{s} & \mathcal{C}_0 \end{array}$$

together with the morphism $c : M_{\mathcal{C}} \to \mathcal{C}_1$ such that the following diagrams commute:

$$\begin{array}{ccc} M_{\mathcal{C}} & \xrightarrow{p_1} & \mathcal{C}_1 \\ \downarrow{\scriptstyle c} & & \downarrow{\scriptstyle s} \\ \mathcal{C}_1 & \xrightarrow{s} & \mathcal{C}_0 \end{array} \qquad \begin{array}{ccc} M_{\mathcal{C}} & \xrightarrow{p_2} & \mathcal{C}_1 \\ \downarrow{\scriptstyle c} & & \downarrow{\scriptstyle t} \\ \mathcal{C}_1 & \xrightarrow{t} & \mathcal{C}_0 \end{array}$$

The composition also expresses the unit law, such that the following diagram commutes:

$$\begin{array}{ccccc} \mathrm{id} \times_{\mathcal{C}_0} t & \xrightarrow{\mathcal{C}\times\mathrm{id}} & s \times_{\mathcal{C}_0} t & \xleftarrow{\mathrm{id}\times\mathcal{C}} & s \times_{\mathcal{C}_0} \mathrm{id} \\ & {\scriptstyle p_2}\searrow & \downarrow{\scriptstyle c} & \swarrow{\scriptstyle p_1} & \\ & & \mathcal{C}_1 & & \end{array}$$

Now, given categories $\mathcal{C}, \mathcal{D}$ internal to $\mathbb{B}$, one can internally define a functor $F : \mathcal{C} \to \mathcal{D}$.

Definition 6.2 (Internal Functor)**.** *Given an ambient category $\mathbb{B}$ and categories $\mathcal{C}, \mathcal{D}$ internal to $\mathbb{B}$, an internal functor $F : \mathcal{C} \to \mathcal{D}$ consists of a morphism of objects $F_0 : \mathcal{C}_0 \to \mathcal{D}_0$, and a morphism of morphisms $F_1 : \mathcal{C}_1 \to \mathcal{D}_1$ such that it respects the maps s, t, e, c:*

$$\begin{array}{ccccc}
\mathcal{C}_0 & \underset{t}{\overset{s}{\underset{\longleftarrow}{\overset{\longleftarrow}{e\rightarrow}}}} & \mathcal{C}_1 & \xleftarrow{c} & M_{\mathcal{C}} \\
\downarrow{\scriptstyle F_0} & & \downarrow{\scriptstyle F_1} & & \downarrow{\scriptstyle s\times_{F_1} t} \\
\mathcal{D}_0 & \underset{t}{\overset{s}{\underset{\longleftarrow}{\overset{\longleftarrow}{e\rightarrow}}}} & \mathcal{D}_1 & \xleftarrow{c} & M_{\mathcal{D}}
\end{array}$$

Meaning that $s \circ F_1 = F_0 \circ s$, $t \circ F_1 = F_0 \circ t$, $e \circ F_1 = F_0 \circ e$, and $c \circ s \times_{F_1} t = F_1 \circ c$, where $s \times_{F_1} t$ is the composition $M_{\mathcal{C}}$ in $\mathcal{D}$.

To define monads internally, we first require an internal definition of natural transformations.

Definition 6.3 (Internal Natural Transformation)**.** *Given internal functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{C} \to \mathcal{D}$:*

$$\begin{array}{ccc}
\mathcal{C}_1 & \underset{s}{\overset{t}{\underset{\longleftarrow}{\overset{\longleftarrow}{i\rightarrow}}}} & \mathcal{C}_0 \\
{\scriptstyle F_1}\big(\quad\big){\scriptstyle G_1} & & {\scriptstyle F_0}\big(\quad\big){\scriptstyle G_0} \\
D_1 & \underset{s}{\overset{t}{\underset{\longleftarrow}{\overset{\longleftarrow}{e\rightarrow}}}} & \mathcal{D}_0
\end{array}$$

A natural transformation between functors F, G is a morphism α, such that it respects the source and target maps:

$$\begin{array}{ccccc}
D_1 & \xleftarrow{\alpha} & \mathcal{C}_0 & \xrightarrow{\alpha} & \mathcal{D}_1 \\
\downarrow{\scriptstyle t} & \swarrow{\scriptstyle G_0} & & \searrow{\scriptstyle F_0} & \downarrow{\scriptstyle s} \\
\mathcal{D}_0 & & & & \mathcal{D}_0
\end{array}$$

and satisfies the naturality condition:

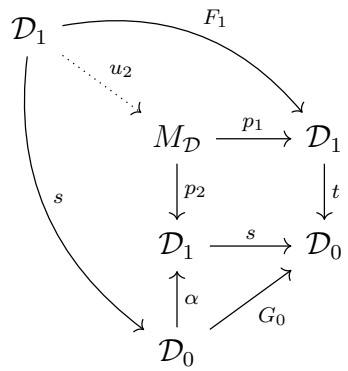
$$\begin{array}{ccc}
\mathcal{C}_1 & \xrightarrow{F_1} & \mathcal{D}_1 \\
\downarrow{\scriptstyle \alpha\circ s} & & \downarrow{\scriptstyle \alpha\circ t} \\
\mathcal{D}_1 & \xrightarrow{G_1} & \mathcal{D}_1
\end{array}$$

The compositions $\alpha \circ t \circ F_1$ and $G_1 \circ \alpha \circ s$ are uniquely defined on the following pullbacks respectively:

$$\begin{array}{ccccc}
\mathcal{D}_1 & & \xrightarrow{\quad t \quad} & & \mathcal{D}_0 \\
& \overset{u_1}{\searrow} & & & \downarrow{\scriptstyle \alpha} \\
{\scriptstyle F_1}\big(& M_{\mathcal{D}} & \xrightarrow{p_1} & \mathcal{D}_1 & \big){\scriptstyle F_0} \\
& \downarrow{\scriptstyle p_2} & & \downarrow{\scriptstyle s} & \\
& \mathcal{D}_1 & \xrightarrow{t} & \mathcal{D}_0 &
\end{array}$$

We have that $t \circ F_1 = s \circ \alpha \circ t$, because we have that $F_0 \circ t = t \circ F_1$ and $s \circ \alpha = F_0$ by our earlier diagrams.

We do the reverse for the composition $G_1 f \circ \alpha_X$:



Again, we have that $s \circ G_1 = t \circ \alpha s$ because $s \circ G_1 = G_0 \circ s$, since the functor respects the source map, and $t \circ \alpha = G_0$ since the natural transformation respects the target map. By the UMP of the pullback, we have that $u_1 = u_2$, proving the axiom of a natural transformation is satisfied.

6.2 Externalisations

Now we can try to define a monad on $\mathcal{C}$ internal to $\mathbb{B}$ as the endofunctor $F : \mathcal{C} \to \mathcal{C}$ a unit $\eta : \mathcal{C}_0 \to \mathcal{C}_1$ and a multiplication $\mu : \mathcal{C}_0 \to \mathcal{C}_1$. A problem arises when we have to prove the monad axioms. For instance, how do we define the component of the unit at X, $\eta_X : X \to F_0 X$, with $X \in \mathcal{C}_0$? As seen in the complexity of Definition 6.3, working with internal natural transformation will become cumbersome.
Therefore, to take away a load of internal reasoning, we introduce the concept of externalisations to take away a load of internal reasoning.

Definition 6.4 (Externalisation)**.** *Given an ambient category $\mathbb{B}$ and a category $\mathcal{C}$ internal to $\mathbb{B}$ the externalisation of $\mathcal{C}$ is the fibration $P_C : \underline{\mathcal{C}} \to \mathbb{B}$, where objects of $\underline{\mathcal{C}}$ are pairs $(I, X : I \to \mathcal{C}_0)$ and morphisms the maps $(u : I \to J, f : I \to \mathcal{C}_1)$ with commuting diagrams:*

$$\begin{array}{ccc} I & \xrightarrow{f} & \mathcal{C}_1 \\ & \searrow^{X} & \downarrow s \\ & & \mathcal{C}_0 \end{array} \qquad\qquad \begin{array}{ccc} I & \xrightarrow{f} & \mathcal{C}_1 \\ \downarrow u & & \downarrow t \\ J & \xrightarrow{Y} & \mathcal{C}_0 \end{array}$$

Composition of morphisms $(u : I \to J, f : I \to \mathcal{C}_1)$ and $(v : J \to K, g : J \to \mathcal{C}_1)$ is defined as

$$(v, g) \circ (u, f) = (v \circ u, c \circ \langle f, g \circ u \rangle)$$

with $\langle f, g \circ u \rangle : I \to s \times_{\mathcal{C}_0} t$ and $c : s \times_{\mathcal{C}_0} t \to \mathcal{C}_1$.

Now if we show an equivalence between the category of split fibrations over $\mathbb{B}$, SpFib($\mathbb{B}$) and the ambient category Cat($\mathbb{B}$, we can define a fibred monad on P_C which then should reflect a monad internal to $\mathbb{B}$, which reduces the load of internal reasoning.

Theorem 2. *Given an ambient category $\mathbb{B}$, there exists an equivalence of categories between the category of split fibrations over $\mathbb{B}$,* SpFib($\mathbb{B}$) *and* Cat($\mathbb{B}$.

Proof. Given the internalisation

$$i : \mathrm{SpFib}(\mathbb{B}) \to \mathrm{Cat}(\mathbb{B})$$

and externalisation

$$\underline{(-)} : \mathrm{Cat}(\mathbb{B}) \to \mathrm{SpFib}(\mathbb{B})$$

and categories $\mathcal{C}, \mathcal{D}$ internal to $\mathbb{B}$, to show the equivalence of categories, we have to prove the natural isomorphism

$$i \circ (-) = \mathrm{Id}_{\mathrm{Cat}(\mathbb{B})}$$
$$(-) \circ i = \mathrm{Id}_{\mathrm{SpFib}(\mathbb{B})}$$

exists.
For this, we have to show that:

$$F \mapsto \underline{F}$$
$$\alpha \mapsto \underline{\alpha}$$
$$iG \leftarrow\!\shortmid \underline{G}$$
$$i\gamma \leftarrow\!\shortmid \gamma$$

With functors $F, G : \mathcal{C} \to \mathcal{D}$, internal natural transformation $\alpha : F \Rightarrow G$ and external vertical transformation $\gamma : \underline{F} \Rightarrow \underline{G}$.

First, we need to define a functor $\underline{F} : \underline{\mathcal{C}} \to \underline{\mathcal{D}}$ for externalisations $P_C : \underline{\mathcal{C}} \to \mathbb{B}$, $P_D : \underline{\mathcal{D}} \to \mathbb{B}$ for internal categories $\mathcal{D}, \mathcal{C} \in \mathbb{B}$ together with $F : \mathcal{C} \to \mathcal{D}$.
We define the action of $\underline{F}$ on the objects of $\underline{\mathcal{C}}$ as

$$\underline{F}(I, X : I \to \mathcal{C}_0) = (I, F_0 X : I \to \mathcal{D}_0)$$

and on the morphism as

$$\underline{F}(u, f) = (u, F_1 f : I \to \mathcal{D}_1)$$

First, we prove the $\underline{F}$ preserves identity. The identity of $X \in \underline{\mathcal{C}}$ is defined as the composite map $e \circ X$.

$$\underline{F}_{\mathrm{id}_{(I,X)}} = (\mathrm{id}_I, F_1(e \circ X)) = (\mathrm{id}_I, eF_0X)$$

since $F_1 e = eF_0$ holds by functoriality of F. Also, $(\mathrm{id}_I, eF_0X) = \mathrm{id}_{\underline{F}(I,X)}$.

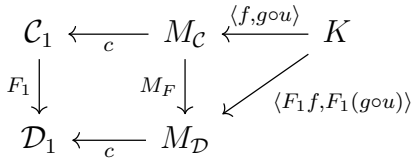
$\underline{F}$ also preserves composition: We have to show that

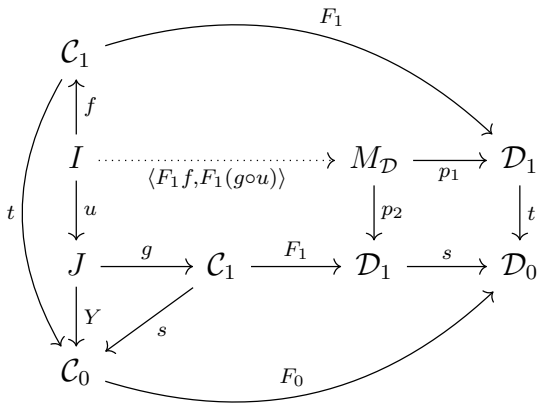
$$\underline{F}(v, g) \circ \underline{F}(u, f) = \underline{F}(v \circ u, c \circ \langle f, g \circ u \rangle)$$

and we have that

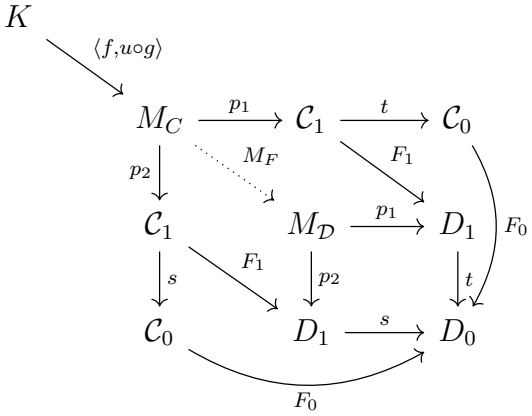
$$\underline{F}(v, g) = (v, F_1 g)$$
$$\underline{F}(u, f) = (u, F_1 f)$$
$$(v, F_1 g) \circ (u, F_1 f) = (v \circ u, c \circ \langle F_1 f, F_1(g \circ u) \rangle)$$
$$\underline{F}(v \circ u, c \circ \langle f, g \circ u \rangle) = (v \circ u, F_1(c \circ \langle f, g \circ u \rangle))$$

By functoriality of F, we have that $c \circ M_F = F_1 \circ c$, with $M_F : M_{\mathcal{C}} \to M_{\mathcal{D}}$. If we can show that $\langle F_1 fv, F_1 g \rangle$ defines the map $\mathcal{C}_1 \times_{\mathcal{C}_0} \mathcal{C}_1 \to F_1\mathcal{C}_1 \times_{\mathcal{C}_0} F_1\mathcal{C}_1$, by the UMP of the pullback $M_{\mathcal{D}}$, we have that $M_F \circ \langle f, g \circ u \rangle = \langle F_1 f, F_1(g \circ u) \rangle$ and $c \circ \langle F_1 f, F_1(g \circ u) \rangle = F_1(c \circ \langle f, g \circ u \rangle)$ and by the following diagram the functor $\underline{F}$ preserves composition:





We have that $t \circ F_1 \circ f = s \circ F_1 \circ g \circ u$, because $F_0 \circ s = s \circ F_1$ and $F_0 \circ t = t \circ F_1$ because of functoriality of F, and $Y \circ u = t \circ f$ and $s \circ s = Y$, because of $P_{\mathcal{C}}$ being a fibration. Consequently, the unique arrow concerning the UMP of the pullback $\langle F_1 f, F_1(u \circ g) \rangle$ is induced.



We have that $F_0 \circ s \circ p_2 \circ \langle fv, g \rangle = F_0 \circ t \circ p_1 \circ \langle fv, g \rangle$, so $M_F \circ \langle fv, g \rangle$ is unique by the UMP of the pullback, proving

$$F_1 \circ c \circ \langle f, u \circ g \rangle = c \circ M_F \circ \langle f, u \circ g \rangle$$

and thereby proving that

$$F_1 \circ c \circ \langle f, u \circ g \rangle = c \circ \langle F_1 f, F_1(u \circ g) \rangle$$

This proves an internal functor $F = \langle F_0, F_1 \rangle : \mathcal{C} \to \mathcal{D}$ yields a split fibred functor $\underline{F} : \underline{C} \to \underline{D}$.

Now, for the split fibred functor $\underline{F}$, the functor $i\underline{F}$ is defined on objects as

$$i\underline{F}(I, X) = i(I, FX) = FX$$

and on morphisms as

$$i\underline{F}(u : I \to J, f : I \to \mathcal{C}_0) = i(u, F_1 f) = F_1 f$$

showing that the split fibred functor $\underline{F}$ induces a functor $i\underline{F}$ between the corresponding internal categories, such that objects and morphisms in the internal categories are preserved, proving that i sends split fibred functors to internal functors.
Now, we go on to prove the correspondence of an internal natural transformation $\alpha : \mathcal{C}_0 \to \mathcal{D}_1$ and a vertical natural transformation $\gamma : \underline{F} \Rightarrow \underline{G}$. For each morphism $X : I \to \mathcal{C}_0$ in $\underline{\mathcal{C}}$, we get the components $\gamma_X : I \to \mathcal{D}_1$, which are morphisms in $\underline{\mathcal{D}}$. We want to show that we get a unique internal natural transformation α, such that $\underline{\alpha} = \gamma$. Now we let $I = \mathcal{C}_0$, $X = \mathrm{id}_{\mathcal{C}_0}$, which results in a unique morphism $\alpha = \gamma_{\mathrm{id}_{\mathcal{C}_0}} : \mathcal{C}_0 \to \mathcal{D}_1$. As we want to show that $i\gamma \leftarrow\!\shortmid \gamma$, we let $\alpha = \gamma_{\mathrm{id}_{\mathcal{C}_0}} = i\gamma$. Now we have to show that $\underline{i\gamma} = \gamma$, which is the case if $\underline{i\gamma}_{(I,X)} = \gamma_{(I,X)}$ but this follows from naturality of γ, if we consider the morphism $(X, e \circ X) : (I, X) \to (\mathcal{C}_0, \mathrm{id}_{\mathcal{C}_0})$:

$$\begin{array}{ccc} \underline{F}(I, X) & \xrightarrow{\gamma_X} & \underline{G}(I, X) \\ {\scriptstyle \underline{F}(X, e\circ X)}\downarrow & & \downarrow{\scriptstyle \underline{G}(X, e\circ X)} \\ \underline{F}(\mathcal{C}_0, \mathrm{id}_{\mathcal{C}_0}) & \xrightarrow{\gamma_{\mathrm{id}_{\mathcal{C}_0}}} & \underline{G}(\mathcal{C}_0, \mathrm{id}_{\mathcal{C}_0}) \end{array}$$

By definition of composition, we have that

$$\gamma_{(\mathcal{C}_0, \mathrm{id}_{\mathcal{C}_0})} \circ \underline{F}(X, e \circ X) = (X, c \circ \langle F_1 eX, \gamma_{(\mathcal{C}_0, \mathrm{id}_{\mathcal{C}_0})} \rangle$$
$$\underline{G}(X, e \circ X) = \gamma_{(I,X)} = (X, c \circ \langle \gamma_{(I,X)}, G_1 eX \rangle$$

meaning that

$$c \circ \langle F_1 eX, \gamma_{(\mathcal{C}_0, \mathrm{id}_{\mathcal{C}_0})} \rangle = c \circ \langle \gamma_{(I,X)}, G_1 eX \rangle$$

By the unit laws of the composition, we have that

$$\begin{aligned} c \circ \langle eF_0 x, \gamma_{\mathrm{id}_{\mathcal{C}_0}} \circ X \langle &= c \circ (e \times \mathrm{id}) \circ \langle F_0 X, \gamma_{id_{\mathcal{C}_0}} X \rangle \\ &= p_2 \circ \langle F_0 X, \gamma_{id_{\mathcal{C}_0}} X \rangle \\ &= \gamma_{\mathrm{id}_{\mathcal{C}_0}} \end{aligned}$$

Now, the other way around

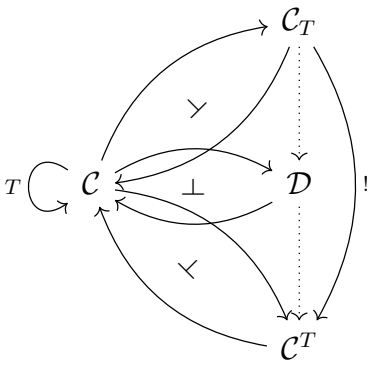
$$\begin{aligned} c \circ \langle \gamma_{(I,X)}, G_1 ex \rangle &= c \circ \langle \gamma_X, eG_0 X \rangle \\ &= p_1 \circ \langle \gamma_X, G_0 X \rangle \\ &= \gamma_X \end{aligned}$$

Proving that $i\gamma \circ X = \gamma_X$. Uniqueness of α is easily proven by taking another internal natural transformation α' and showing they share the same components at X and the same naturality condition.
Now, to show that the internal natural transformation α yields an external vertical transformation, we have to show that $\underline{\alpha}$ satisfies the naturality condition for every object (I, X) in $\underline{\mathcal{C}}$. We have that $\underline{\alpha}_X = \alpha \circ X = i\gamma \circ X = \gamma_{\mathcal{C}_0, \mathrm{id}_{\mathcal{C}_0}} = \gamma_{I,X}$, which means $\underline{\alpha}$ shares the components of γ, which entails that $\underline{\alpha}$ satisfies the naturality condition for every object (I, X) in $\underline{C}$. □

6.3 Internally Defining the Kleisli Category

Now that we know how to define structures internally and have shown the correspondence between Cat($\mathbb{B}$) and SpFib($\mathbb{B}$), we will see how a Kleisli object can be defined internally to the category of categories.

Definition 6.5 (Internal Kleisli Object)**.** *First, we describe an internal Kleisli object.* Cat($\mathbb{B}$) *and* SpFib($\mathbb{B}$) *are both 2-categories. A monad in* SpFib *is a fibred functor, together with a fibred natural transformation. Now, for the 2-category* Cat($\mathbb{B}$)*, we can define a monad internally. We define a 1-cell in* Cat($\mathbb{B}$)*, $T : \mathcal{C} \to \mathcal{C}$, which is an internal functor. We define the 2-cell $\eta : \mathrm{Id} \to T$, with the 1-cell* Id*, and we define the 2-cell $\mu : T \circ T \to T$. Both η and μ are internal natural transformations. Now, we describe a Kleisli object $\mathcal{C}_T$, with 1-cell $f_T : \mathcal{C} \to \mathcal{C}_T$ and 2-cell $\lambda : f_T \circ T \to T$, meaning $f_T(X) = TX$, $f_T(f) = Tf$, and $\lambda_Y : T^2Y \to TY = \mu_Y$. Splitting T into an adjunction gives us the following diagram:*



The map $\mathcal{C}_T \to \mathcal{C}^T$ is unique such that for any other $\mathcal{D}$, $g : \mathcal{C} \to \mathcal{D}$, $\gamma : g \circ T \to g$ there exists a unique arrow $k : \mathcal{C}_T \to \mathcal{D}$, such that $k \circ f_T = g$, and the 2-cell $k\lambda : kf_TT = gT \to g = \gamma$ is induced by whiskering.

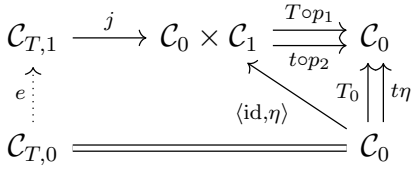
Now that we have a description of an internal Kleisli object, we still need a construction. The way we achieve this is by seeing how the Kleisli category is defined in Cat.

Theorem 3. *For a category $\mathcal{C}$, internal to* Cat*, together with the internal functor $T : \mathcal{C} \to \mathcal{C}$ and internal natural transformations $\mu : \mathcal{C}_0 \to \mathcal{C}_1$ and $\eta : \mathcal{C}_0 \to \mathcal{C}_1$, we can define the internal Kleisli category $\mathcal{C}_T$.*

Proof. Objects $\mathcal{C}_{T,0}$ are the same as the objects in $\mathcal{C}$, $\mathcal{C}_0$. Morphisms $\mathcal{C}_{T,1}$ are captured by the equalizer:

$$\mathcal{C}_{T,1} \xrightarrow{\ j\ } \mathcal{C}_0 \times \mathcal{C}_1 \underset{t \circ p_2}{\overset{T \circ p_1}{\rightrightarrows}} \mathcal{C}_0$$

We define the identity assigning morphism $e : \mathcal{C}_{T,0} \to \mathcal{C}_{T,1}$ by inducing it with the universal property of the equalisers as illustrated in the diagram below:



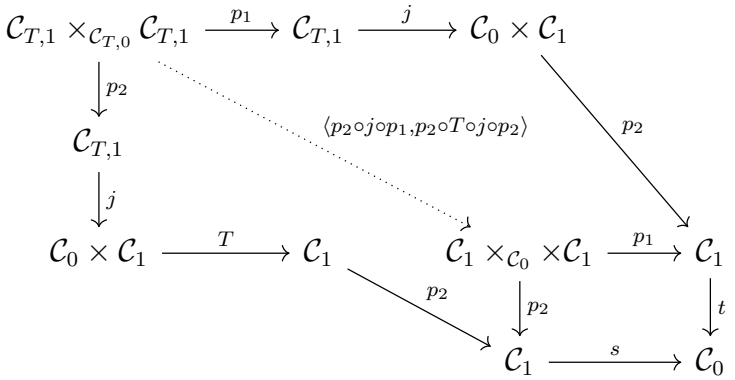
As we have that $T_0 X = t\eta$, the unique map $e : \mathcal{C}_{T,0} \to \mathcal{C}_{T,1}$ is induced, and it is defined by the unit η of the associated monad T.

We let the source and target maps be the following maps respectively:

$$s = s \circ p_2 \circ j : \mathcal{C}_{T,1} \to \mathcal{C}_{T,0}$$

$$t = p_1 \circ j : \mathcal{C}_{T,1} \to \mathcal{C}_{T,0}$$

Now, it remains to define the composition morphism. For an external Kleisli category, given Kleisli morphisms $f : X \to TY$, $g : Y \to TZ$, notice that $g \circ_{\text{Kleisli}} f = \mu_Z \circ Tg \circ f$. Therefore, we will try to show that the composition map can be induced by the universal property of the equaliser, using the multiplication of the associated monad and the composition of the morphisms $\mathcal{C}_1 \times \mathcal{C}_1$, corresponding to the morphisms $\mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1}$, and applying T to the second projection of this product. To get this composition, which we can see as $Tg \circ f$ per abuse, we will see how the map $\mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} \to \mathcal{C}_1 \times \mathcal{C}_1$ is defined on the following pullback:



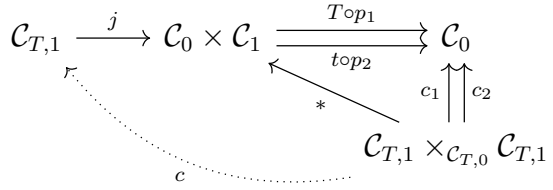
Now, we want to compose this composition with the multiplication applied to the object of objects obtained by the first projection after applying j to the second morphism in $\mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1}$, which, if we follow the example of the external Kleisli composition, can, per abuse, see as Z, which gives us the map $c \circ \langle c \circ \langle p_2 \circ j \circ p_1, p_2 \circ T \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle : \mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} \to \mathcal{C}_1$. As we need the map $\mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} \to \mathcal{C}_0 \times \mathcal{C}_1$, we obtain $\mathcal{C}_0$ from $\mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1}$ with the map $p_2 \circ j \circ p_1$, finally giving us the following map:

$$\langle p_2 \circ j \circ p_1, c \circ \langle c \circ \langle p_2 \circ j \circ p_1, p_2 \circ T \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle \rangle : \mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} \to \mathcal{C}_0 \times \mathcal{C}_1$$

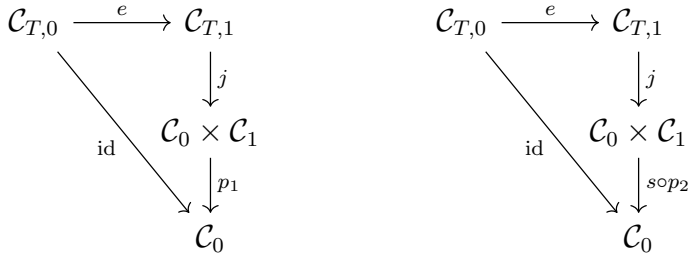
Now, to induce the map $c : \mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} \to \mathcal{C}_{T,1}$ with the universal property of the equaliser, we need to find two maps from $\mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1}$ to $\mathcal{C}_0$ and show they are equal. The first map is given by$t \circ c \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle$. The second map is given by $t \circ p_2 \circ j \circ p_2$. By the pullback

$$\begin{array}{ccc} \mathcal{C}_1 \times_{\mathcal{C}_0} \mathcal{C}_1 & \xrightarrow{c} & \mathcal{C}_1 \\ \downarrow{\scriptstyle p_2} & & \downarrow{\scriptstyle t} \\ \mathcal{C}_1 & \xrightarrow{t} & \mathcal{C}_0 \end{array}$$

we have that $t \circ c = t \circ p_2$, showing that $t \circ c \circ \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle = t \circ \mu \circ p_1 \circ j \circ p_2$. By diagram 6.3, we have that $t \circ \mu = T$, showing that $T \circ p_1 \circ j \circ p_2 = T \circ p_1 \circ j \circ p_2$ and by the universal property of the equaliser we have that $T \circ p_1 \circ j = t \circ p_2 \circ j$, showing that $T \circ p_1 \circ j \circ p_2 = t \circ p_2 \circ j \circ p_2$, proving the maps are equal, and inducing the map $c : \mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} \to klM$. The relevant maps are shown in the diagram below, with $* = \langle p_1 \circ j \circ p_2\, c \circ \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle \rangle$, $e_1 = t \circ p_2 \circ j \circ p_2$, and $e_2 = t \circ c \circ \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle$



Now, it remains for us to check the laws of the internal category $\mathcal{C}_T$. First, we check the axioms for the source and target of the identity assigning morphism by showing the following diagrams commute:



By the universal property of the equaliser, we have that $j \circ e = \langle \mathrm{id}, \eta \rangle$, meaning that we have to show that $p_1 \circ \langle \mathrm{id}, \eta \rangle = \mathrm{id}$, which holds by definition, and that $s \circ p_2 \circ \langle \mathrm{id}, \eta \rangle = \mathrm{id}$. We have that $p_2 \circ \langle \mathrm{id}, \eta \rangle = \eta$, and that $s \circ \eta = \mathrm{id}$, since the unit of the internal monad has to respect the source and target maps, proving that $s \circ p_2 \circ j \circ e = \mathrm{id}$.

Now, for the composition axioms, we first have to show the diagram below commutes:

$$\begin{array}{ccccc} \mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} & \xrightarrow{c} & \mathcal{C}_{T,1} & \xrightarrow{j} & \mathcal{C}_0 \times \mathcal{C}_1 \\ \downarrow{\scriptstyle p_1} & & & & \downarrow{\scriptstyle s \circ p_2} \\ \mathcal{C}_{T,1} & \xrightarrow{j} & \mathcal{C}_0 \times \mathcal{C}_1 & \xrightarrow{s \circ p_2} & \mathcal{C}_0 \end{array}$$

By the universal property of the equaliser, we can substitute $j \circ c$ for

$$\langle p_1 \circ j \circ p_2, c \circ \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle \rangle$$

Now we have to show that

$$s \circ p_2 \circ \langle p_1 \circ j \circ p_2, c \circ \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle \rangle = s \circ p_2 \circ j \circ p_1$$

$s \circ p_2 \circ \langle p_1 \circ j \circ p_2, c \circ \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle \rangle = s \circ c \circ \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle$

By the composition axioms, we have that $s \circ c = s \circ p_1$, giving

$$s \circ c \circ \langle c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle, \mu \circ p_1 \circ j \circ p_2 \rangle = s \circ c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle$$

$$s \circ c \circ \langle p_2 \circ j \circ p_1, T \circ p_2 \circ j \circ p_2 \rangle = s \circ p_2 \circ j \circ p_1$$

proving this holds.

Secondly, we have to show the following diagram commutes:

$$\begin{array}{ccccc} \mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} & \xrightarrow{c} & \mathcal{C}_{T,1} & \xrightarrow{j} & \mathcal{C}_0 \times \mathcal{C}_1 \\ \downarrow{\scriptstyle p_2} & & & & \downarrow{\scriptstyle p_1} \\ \mathcal{C}_{T,1} & \xrightarrow{j} & \mathcal{C}_0 \times \mathcal{C}_1 & \xrightarrow{p_1} & \mathcal{C}_0 \end{array}$$

Again, we substitute $j \circ c$. As we have to prove that $p_1 \circ j \circ c = p_1 \circ j \circ p_2$, applying the first projection to the substitution of $j \circ c$, gives us $p_1 \circ j \circ p_2$, which proves the diagram commutes.

Finally, we have to prove the unit laws for composition holds, for which we need to show that the following diagram commutes:

$$\begin{array}{ccccc} \mathcal{C}_{T,0} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} & \xrightarrow{e \times_{\mathcal{C}_{T,0}} \mathrm{id}} & \mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,1} & \xleftarrow{\mathrm{id} \times_{\mathcal{C}_{T,0}} e} & \mathcal{C}_{T,1} \times_{\mathcal{C}_{T,0}} \mathcal{C}_{T,0} \\ & {\scriptstyle p_2} \searrow & \downarrow{\scriptstyle e} & \swarrow {\scriptstyle p_1} & \\ & & \mathcal{C}_{T,1} & & \end{array}$$

First, we prove the left unit law. If we consider the equaliser

$$\mathcal{C}_{T,1} \xrightarrow{j} \mathcal{C}_0 \times \mathcal{C}_1 \underset{t \circ p_2}{\overset{T \circ p_1}{\rightrightarrows}} \mathcal{C}_0$$

We have that the map p_2 is unique concerning the universal property of this equaliser, such that $T \circ p_1 \circ j \circ p_2 = t \circ p_2 \circ j \circ p_2$. Now, for any other map $f : \mathcal{C}_{T,0} \times_{\mathcal{C}_{T,}}$ id for which we have that $T \circ p_1 \circ j \circ f = t \circ p_2 \circ j \circ f$, we have that $f = p_2$. Therefore, we will show that $T \circ p_1 \circ j \circ c \circ e \times_{\mathcal{C}_{T,0}} \mathrm{id} = t \circ p_2 \circ j \circ c \circ e \times_{\mathcal{C}_{T,0}} \mathrm{id}$ but as c is already unique such that

$$T \circ p_1 \circ j \circ c = t \circ p_2 \circ j \circ c$$

this is the case. In a similar vein, the right unit law can be proven. □

6.4 The Category of Sheaves over $\mathcal{L}$

Now that we have a description of an internal Kleisli object and have a construction of the Kleisli category internal to Cat, we will now see how the Kleisli category is defined internally to the category of sheaves over $\mathcal{L}$ by defining the functor $S : \mathcal{L}^{\text{op}} \to \text{Mnd}(\text{Set})$ internally to this category. Before we can define our constructions internally to $\text{Sh}(\mathcal{L})$, we must consider the size of our objects and constructions. If we, for instance, consider the constant sheaf in $\text{Sh}(\mathcal{L})$ that takes objects ℓ in $\mathcal{L}$ and maps them to the category on which $S(\ell)$ operates, we observe the following. The domain of this sheaf is the collection of objects in the powerset of memory locations $\mathcal{P}(\text{Loc})$. The target of the sheaf is the category on which the functor $S(\ell)$ operates, which is the category of large sets. It is hard to get an explicit concept of the category of large sets, which would mean we have to define the set of all sets we want to avoid. Therefore, we introduce a concept that allows us to define the set of all sets of a specific size, namely the Grothendieck universes.

Definition 6.6 (Grothendieck Universe [ML69])**.** *A Grothendieck Universe U is a pure set satisfying the following properties:*

1. ***Closure Under Subsets:*** *For any set $x \in U$, if $y \subseteq x$ then $y \in U$.*
2. ***Closure Under Cartesian Products:*** *If $x, y \in U$, then $x \times y \in U$.*
3. ***Closure Under Function Spaces:*** *If $X \in U$ and y is a set of functions from sets in U to sets in U, then the set of all functions from x to y is also in U.*
4. ***Closure Under Union:*** *If $I \in U$ and x_i is an I-index family of sets with $x_i \in U$ for all $i \in I$, then the union $\bigcup_i x_i$ is in U.*
5. ***Closure Under Powersets:*** *If $x \in U$, then $\mathcal{P}(x) \in U$.*
6. ***Limit-Preservation:*** *If λ is a limit ordinal and $\{x_\alpha | \alpha < \lambda\}$ is a collection of sets in U then the union $\bigcup_{\alpha<\lambda} x_\alpha$ is also in U.*

Now we define two universes U_0 and U_1 such that for the Grothendieck universe U, we have that $U = U_0 \subseteq U_1$ and $U_0 \in U_1$. By Definition 6.6, U_0 and U_1 are closed under set operations and respect limits.
We also defined the category of $U-$sized categories $U - Cat$, in which the collection of objects is bound by U_0.
We can now define the size of categories relative to other universes via the following scheme:

	$\lvert\mathbb{C}\rvert \subseteq U$	$\lvert\mathbb{C}\rvert \in U$
$\mathbb{C}(A,B) \subseteq U$	$(U-)$Cat	Locally large
$\mathbb{C}(A,B) \in U$	Locally $(U-)$ small	Small $(U-)$Cat

where $|\mathbb{C}|$ is the collection of objects in $\mathbb{C}$ and $\mathbb{C}(A, B)$ is the collection of morphisms in $\mathbb{C}$.
The idea now is to show that $\text{Sh}(\mathcal{L})$ is locally U_1-small. As a consequence of this, a small $U-$category is a category internal in $\text{Sh}(\mathcal{L})$.

Proposition 6.1. *The category of sheaves over $\mathcal{L}$,* $\mathrm{Sh}(\mathcal{L})$ *is locally U_1-small.*

Proof. We have that the target category of objects in $\mathrm{Sh}(\mathcal{L})$ are elements in U_1. The category of sheaves over $\mathcal{L}$ is a subset of the category of presheaves over $\mathcal{L}$ meaning that $|\mathrm{Sh}(\mathcal{L})| \subseteq [\mathcal{L}^{\mathrm{op}}, U_1]$, showing that the collection of objects in $|\mathrm{Sh}(\mathcal{L})|$ is bounded by U_1. Now we have to show the collection of morphisms in $\mathrm{Sh}(\mathcal{L})$ is an element in U_1, for which we have to determine the size of the morphisms in $[\mathcal{L}^{\mathrm{op}}, U_1]$. We have that the set of memory locations Loc is an element in U_0. As U_0 is closed under set operations, $\mathcal{P}(\mathrm{Loc}) \in U_0$, meaning that $\mathcal{L} \in U_0$. For every $\ell \in \mathcal{L}$, we have a collection of morphisms. This means the collection of morphisms in $[\mathcal{L}^{\mathrm{op}}, U_1]$ can be written as

$$\prod_{\ell \in \mathcal{L}} [F_\ell, G_\ell]$$

with $F_\ell, G_\ell \in [\mathcal{L}^{\mathrm{op}}, U_1]$. F_ℓ and G_ℓ are U_1-sets. By Definition 6.6 a set of functions between two U_1-sets is also an object in U_1, proving that $\mathrm{Sh}(\mathcal{L})$ is locally small. □

Objects in $\mathrm{Sh}(\mathcal{L})$ are set-valued sheaves over $\mathcal{L}$ with a sheaf condition of the following form. A presheaf over $\mathcal{L}$ is a functor $F : \mathcal{L}^{\mathrm{op}} \to \mathrm{Set}$. The presheaf F is a sheaf over $\mathcal{L}$ if and only if for each object ℓ in $\mathcal{L}$ and for each open covering of $\mathcal{L}$, $\{\ell_i\}_{i \in I}$ there exists a unique element s in $F(\ell)$ such that $s|\ell_i = s_i$ for any compatible family of elements $\{s_i \in F(\ell_i)\}_{i \in I}$. Morphisms in $\mathrm{Sh}(\mathcal{L})$ are morphisms of sheaves, which are morphisms of the underlying presheaves.

Proposition 6.2. *The functor $S : \mathcal{L}^{\mathrm{op}} \to \mathrm{Mnd}(\mathrm{Set})$ leads to a monad internal in* $\mathrm{Sh}(\mathcal{L})$ *on the following category $\mathcal{C}$ internal in* $\mathrm{Sh}(\mathcal{L})$.

Proof. The category $\mathcal{C}$ is defined as the constant Set sheaf, meaning that the object of objects $\mathcal{C}_0$ is the constant sheaf that maps all objects ℓ to the category on which the functor S operates, which is the category of sets, Set. The object of morphisms $\mathcal{C}_1$ is the constant sheaf that maps all ℓ in $\mathcal{C}$ to the morphisms in the category on which S operates, which are morphisms in Set. Now to show that $S(\ell)$ is a monad internal in $\mathrm{Sh}(\mathcal{L})$, we use Theorem 2 and show that $S(\ell)$ forms a monad in $\mathrm{SpFib}(\mathrm{Sh}(\mathcal{L}))$. A monad in $\mathrm{SpFib}(\mathrm{Sh}(\mathcal{L}))$ is a fibred monad on $P_{\mathcal{C}} : \underline{\mathcal{C}} \to \mathrm{Sh}(\mathcal{L})$, where $\underline{\mathcal{C}}$ is the externalisation of $\mathcal{C}$. As $\mathcal{C}$ is the constant set sheaf, by Theorem 2 its externalisation is isomorphic to the category Set, meaning we can use the original definition of $S(\ell)$ for all ℓ in $\mathcal{L}$, to obtain a fibred monad on $P_{\mathcal{C}}$. $\mathcal{C}_0$ and $\mathcal{C}_1$ actually are sheaves, as for all inclusions $i : \ell_1 \subseteq \ell_2$ the restriction maps $\rho^0_{12} : \mathcal{C}_0\ell_2 \to \mathcal{C}_0\ell_1$ and $\rho^1_{12} : \mathcal{C}_1\ell_2 \to \mathcal{C}_1\ell_1$ actually have to be the identity maps on the category on which S operates, as $\mathcal{C}_0$ and $\mathcal{C}_1$ assign the same set to every open subset in $\mathcal{L}$. The glueing axiom trivially holds because of this. □

As a consequence of Proposition 6.2, we can construct the internal Kleisli category $\mathcal{C}_S$ corresponding with this internal monad as follows.

Definition 6.7. *We define $\mathcal{C}_{S,0}$ as the constant sheaf that maps all objects ℓ in $\mathcal{L}$ to the objects in* Set. *We define $\mathcal{C}_{S,1}$ as the sheaf that maps objects ℓ in $\mathcal{L}$ to the collection of all Kleisli morphisms in $Kl(S(\ell))$. Again, we consider the following equaliser to capture these morphisms:*

$$\mathcal{C}_{S,1} \xrightarrow{\ j\ } \mathcal{C}_0 \times \mathcal{C}_1 \underset{t \circ p_2}{\overset{T \circ p_1}{\rightrightarrows}} \mathcal{C}_0$$

which exists as $\mathrm{Sh}(\mathcal{L})$ *is a Grothendieck topos and therefore is (co-)complete and presentable [Bor94]. Now, the maps* c_S, e_S, s_S, t_S *are defined as follows:*

$$c_S = \langle p_2 \circ j \circ p_1, c \circ \langle c \circ \langle p_2 \circ j \circ p_1, p_2 \circ T \circ j \circ p_2 \rangle \mu \circ p_1 \circ j \circ p_2 \rangle \rangle : \mathcal{C}_{T,1} \times_{\mathcal{C}_{S,0}} \mathcal{C}_{T,1} \to \mathcal{C}_{S,1}$$

$$e_S = \langle \mathrm{id}, \eta \rangle : \mathcal{C}_{S,0} \to \mathcal{C}_{S,1}$$

$$s_S = s \circ p_2 \circ j : \mathcal{C}_{S,1} \to \mathcal{C}_{S,0}$$

$$t_S = p_1 \circ j : \mathcal{C}_{S,1} \to \mathcal{C}_{S,0}$$

That $\mathcal{C}_{S,1}$ *is a sheaf, is the consequence of Proposition 5.1.*

With these constructions, we can finally see how to form a sheaf condition for a "sheaf of Kleisli morphisms" by analysing the sheaf condition of $\mathcal{C}_{S,1}$: For any compatible family of Kleisli morphisms $\{f_i \in \mathcal{C}_{S,1}(\ell_i)\}_{i \in I}$, meaning that for any pair of indices $i, j \in I$, $f_i|_{\ell_i \cap \ell_j} = f_j|_{\ell_i \cap \ell_j}$, then there exists a unique Kleisli morphism $f \in \mathcal{C}_{S,1}(\ell)$ such that $f|_{\ell_1} = f_1$.

7 Separation Logic and Process Composition

To analyse how we can extract process composition and separation logic from this perspective, we will give a few examples using the definitions and construction of the Kleisli category internal to $\mathrm{Sh}(\mathcal{L})$

Example 7.1. *First, we give an example to illustrate the restriction of Kleisli morphisms associated with different memory locations.*

Suppose we have three memory locations in $\mathcal{L}$*:* $\ell_1 = \{1, 2, 3\}$*,* $\ell_2 = \{2, 3, 4\}$*,* $\ell_3 = \{3, 4, 5\}$ *and an open covering* $\ell = \{\ell_1, \ell_2, \ell_3\}$ *of their union.*

Suppose the Kleisli morphisms are as follows:

1. f_1 *is a Kleisli morphism in* $\mathcal{C}_{S,1}(\ell_1)$ *from* $X \to S(\ell_1)Y$
2. f_2 *is a Kleisli morphism in* $\mathcal{C}_{S,1}(\ell_2)$ *from* $X \to S(\ell_2)Y$
3. f_3 *is a Kleisli morphism in* $\mathcal{C}_{S,1}(\ell_3)$ *from* $X \to S(\ell_3)Y$

We consider the intersections

1. $\ell_1 \cap \ell_2 = \{2, 3\}$ *and the restriction of* f_1 *to* $\ell_1 \cap \ell_2$ *denoted as*

$$f_{12} = f_1|_{\ell_1 \cap \ell_2}$$

2. $\ell_1 \cap \ell_3 = \{3\}$ *and the restriction of* f_1 *to* $\ell_1 \cap \ell_3$ *denoted as*

$$f_{13} = f_1|_{\ell_1 \cap \ell_3}$$

3. $\ell_2 \cap \ell_3 = \{3, 4\}$ *and the restriction of* f_2 *to* $\ell_2 \cap \ell_3$ *denoted as*

$$f_{23} = f_2|_{\ell_2 \cap \ell_3}$$

The gluing axiom states that if $f_{12} = f_{13} = f_{23}$*, then there exists a unique Kleisli morphism* f *over* ℓ *such that* $f|_{\ell_1} = f_1$*,* $f|_{\ell_2} = f_2$*, and* $f|_{\ell_3} = f_3$*.*

In this example, we see that the sheaf condition ensures that the Kleisli morphisms on the intersections are compatible, and we can glue them together consistently to obtain a unique Kleisli morphism together consistently to obtain a unique Kleisli morphism over the entire memory state ℓ*.*

Example 7.2. *We consider a scenario where we have two processes,* P_1 *and* P_2*, each represented as a Kleisli morphism in* $\mathcal{C}_{S,1}$*. We assume each process operates on a distinct memory location* ℓ_1 *and* ℓ_2 *in* $\mathcal{L}$ *respectively:*

1. $P_1 : X \to S(\ell_1)Y$
2. $P_2 : Y \to S(\ell_2)Z$

In this context, X, Y, Z *are objects in* Set *and represent the types of data that the processes operate on, and* $S(\ell_1)$ *and* $S(\ell_2)$ *are the Kleisli extensions associated with the memory locations* ℓ_1 *and* ℓ_2*.*

We want to compose these processes sequentially, meaning that the output of P_1 *will serve as the input for* P_2*. To achieve this composition, we will use the glueing axiom of our sheaf in the following way:*

1. *Restrict* P_1 *to* $\ell_1 \cap \ell_2$*:*
$$P_1|_{\ell_1 \cap \ell_2} : X \to S(\ell_1 \cap \ell_2)Y$$
2. *Restrict* P_2 *to* $\ell_1 \cap \ell_2$*:*
$$P_2|_{\ell_1 \cap \ell_2} : Y \to S(\ell_1 \cap \ell_2)Z$$

Now, we check if these restricted processes satisfy the sheaf condition, which is the case if the restrictions of P_1 *and* P_2 *to* $\ell_1 \cap \ell_2$ *are compatible.*

If the restricted processes satisfy the sheaf condition, we can form the composed process

$$P_2|_{\ell_1 \cap \ell_2} \circ P_1|_{\ell_1 \cap \ell_2} : X \to S(\ell_1 \cap \ell_2)Z$$

The sheaf condition guarantees that the composed process is well-defined and consistent with the individual processes.

Example 7.3. *Suppose we have the following Kleisli morphisms in* $\mathcal{C}_{S,1}$*:*

1. $c_1 \in \mathcal{C}_{S,1}(\ell_1) : X \to S(\ell_1)Y$*, a program effect modifying* ℓ_1
2. $c_2 \in \mathcal{C}_{S,1}(\ell_2) : X \to S(\ell_2)Y$*, a program effect modifying* ℓ_2
3. $c_{12} \in \mathcal{C}_{S,1}(\ell_1 \cup \ell_2) : X \to S(\ell_1 \cup \ell_2)Y$*, a program effect modifying both the contents in* ℓ_1 *and* ℓ_2

We assume the memory locations ℓ_1 and ℓ_2 are disjoint. By this assumption, the following equations hold:

1. $c_{12}|_{\ell_1} = c_1$, *meaning the program effect on* $\ell_1 \cup \ell_2$ *is the same as the effect on* ℓ_1 *alone*
2. $c_{12}|_{\ell_2} = c_2$, *meaning the program effect on* $\ell_1 \cup \ell_2$ *is the same as the effect on* ℓ_2 *alone*

Because of these qualities, we use the sheaf condition to compose the local effects to infer the global effect on the entire memory state: the sheaf condition ensures the existence of a unique Kleisli morphism $c : X \to S(\mathrm{Loc})Y$ *such that:*

- $c|_{\ell_1} = c_1$, *meaning the effect of* c *on* ℓ_1 *is the same as the local effect*
- $c|_{\ell_2} = c_2$, *meaning the effect of* c *on* ℓ_2 *is the same as the local effect*

Therefore, we have composed the local effect c_1 and c_2 into a global effect c representing the overall effect on the entire memory state.

8 Conclusion and Discussion

This thesis makes two contributions. First, the category-theoretic foundation for reasoning about locality and compositionality allows working towards a complete abstract framework for separation logic. Secondly, a clear description and construction of Kleisli objects internally to a category with pullbacks and small products and the relationship with their external representation. A sheaf-theoretic framework was constructed by defining the category of sheaves over a topological space $\mathcal{L}$, where $\mathcal{L}$ represents the preorder category of memory locations. This framework allows for precise definitions, systematic reasoning, and by using category theory to infer global properties of computational states based on local observations. By defining the Kleisli category internally to this category, we enriched the semantic expressiveness of the framework so that we can represent and reason about side effects or other computational aspects within the context of the sheaf-theoretic model. An isomorphism between how the structures occur internally and their external representation has been established to reduce the load of reasoning in internal categories. Example 7.3 adequately gives insight into how our construction forms the foundation for an abstract framework for separation logic, while showing that additional structure is required for a complete framework. If we analyse this example more closely and introduce predicates $P(\ell_1)$ and $P(\ell_2)$ that represent properties of the data stored in the memory described by $\ell_1, \ell_2 \in \mathcal{L}$ we can use the sheaf condition to reason about the effect of c on the predicates $P(\ell_1)$ and $P(\ell_2)$ based on the effects of c_1 on $P(\ell_1)$ and c_2 on $P(\ell_2)$ which would lead to the semantics of the frame rule. However, in this scenario, we require the ability to reason about how the predicates $P(\ell_1)$ and $P(\ell_2)$ evolve as a result of applying c_1 and c_2, respectively. This involves predicate transformers, which our current construction does not support.

Additional limitations are that our model does not include quantification over predicates. In separation logic, multiple predicates can describe the heap's state. To express multiple predicates with our framework, we must define specific sheaf objects for multiple predicates and reason about their glueing and compatibility to infer their property. This approach does not scale well and is

impractical. Secondly, while the direct mapping of memory provided by the functor M provides a pleasing simplicity, it does not capture all the nuances and complex relationships that arise in more advanced memory models. It restricts our model to be limited to theoretical exploration.

8.1 Future Work

The first natural step to proceed to a complete framework is adding structure for reasoning about predicates and how they change when transitioning between states in a program. Notions of fibred categories are known to provide predicates over categories [Jac99]. In the study of [AK20], the authors characterise Hoare Triples as morphisms in the lifting of a monad on a category modelling computational effects along a fibration mapping a category of predicates to this underlying category. This concept helps provide predicate transformers in our categorical construction. Once this framework has been established, the next step of further research would be to investigate how the underlying logic of the framework relates to extensions of separation logic. Secondly, while the direct mapping of memory provided by the functor M provides a pleasing simplicity, it does not capture all the nuances and complex relationships that arise in more advanced memory models. It restricts our model to be limited to theoretical exploration. In the paper of Ferreira and co-authors [FFS10], a parameterised memory in the context of concurrent separation logic is discussed that could serve as inspiration.

References

[AK20] Alejandro Aguirre and Shin-ya Katsumata. Weakest preconditions in fibrations. *Electronic Notes in Theoretical Computer Science*, 352:5–27, 2020. The 36th Mathematical Foundations of Programming Semantics Conference, 2020.

[BBTS07a] Bodil Biering, Lars Birkedal, and Noah Torp-Smith. Bi-hyperdoctrines, higher-order separation logic, and abstraction. *ACM Transactions on Programming Languages and Systems*, 29(5):24, 2007.

[BBTS07b] Bodil Biering, Lars Birkedal, and Noah Torp-Smith. Bi-hyperdoctrines, higher-order separation logic, and abstraction. *ACM Trans. Program. Lang. Syst.*, 29(5):24–es, aug 2007.

[BHL19] Gilles Barthe, Justin Hsu, and Kevin Liao. A probabilistic separation logic. *Proc. ACM Program. Lang.*, 4(POPL), dec 2019.

[BKKS16] Harsh Beohar, Barbara König, Sebastian Küpper, and Alexandra Silva. A coalgebraic treatment of conditional transition systems with upgrades. *Logical Methods in Computer Science*, 14, 12 2016.

[Bor94] Francis Borceux. *Handbook of Categorical Algebra*, volume 3 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994.

[BTSY06] Lars Birkedal, Noah Torp-Smith, and Hongseok Yang. Semantics of separation-logic typing and higher-order frame rules for algol-like languages. *Logical Methods in Computer Science*, Volume 2, Issue 5, 2006.

[Cur14] Justin Curry. Sheaves, cosheaves and applications, 2014.

[FFS10] Rodrigo Ferreira, Xinyu Feng, and Zhong Shao. Parameterized memory models and concurrent separation logic. *Programming Languages and Systems*, page 267–286, 2010.

[FS79] M. P. Fourman and D. S. Scott. *Sheaves and logic*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1979.

[GR71] Alexander Grothendieck and Michèle Raynaud. *Revêtements Etales et Groupe Fondamental*, volume 224 of *Lecture Notes in Mathematics*. Springer, Berlin, Heidelberg, 1971.

[Hoa69] C. A. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

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

[JJ02] Peter T. Johnstone and Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium: Volume 1*. Oxford Logic Guides. Oxford University Press, Oxford, New York, September 2002.

[KK13] Henning Kerstan and Barbara König. Coalgebraic trace semantics for continuous probabilistic transition systems. *Log. Methods Comput. Sci.*, 9(4), 2013.

[Law69] F. William Lawvere. Adjointness in foundations. *dialectica*, 23(3–4):281–296, 1969.

[LIO] J. L. LIONS. Ariane 5 flight 501 failure report by the inquiry board.

[ML69] Saunders Mac Lane. One universe as a foundation for category theory. In *Reports of the Midwest Category Seminar III*, pages 192–200, Berlin, Heidelberg, 1969. Springer Berlin Heidelberg.

[MM15] Kenji Maillard and Paul-André Melliès. A fibrational account of local states. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 402–413. IEEE Computer Society, 2015.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991. Selections from 1989 IEEE Symposium on Logic in Computer Science.

[MR77] Michael Makkai and Gonzalo E. Reyes. *First Order Categorical Logic: Model-Theoretical Methods in the Theory of Topoi and Related Categories*, volume 611 of *Lecture Notes in Mathematics*. Springer, Berlin, Heidelberg, 1977.

[MS18] Paul-André Melliès and Léo Stefanesco. A game semantics of concurrent separation logic. *Electronic Notes in Theoretical Computer Science*, 336:241–256, 2018.

[O'H04] Peter W. O'Hearn. Resources, concurrency and local reasoning. *CONCUR 2004 Concurrency Theory*, page 49–67, 2004.

[ORY01] Peter O'Hearn, John Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. *Computer Science Logic*, page 1–19, 2001.

[PP02] Gordon D. Plotkin and John Power. Notions of computation determine monads. In *Foundations of Software Science and Computation Structures*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.

[Rey02] J.C. Reynolds. Separation logic: A logic for shared mutable data structures. *Proceedings 17th Annual IEEE Symposium on Logic in Computer Science*, 2002.

[Sof09] Sofronie-Stokkermans, V. Sheaves and geometric logic and applications to modular verification of complex systems. *Electronic Notes in Theoretical Computer Science*, 230:161–187, 2009.

[TO15] Takeshi Tsukada and C.H. Luke Ong. Nondeterminism in game semantics via sheaves. *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2015.