UNIVERSITY OF CALGARY

Higher-Order Message Passing in CaMPL

by

Melika Norouzbeygi

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

SEPTEMBER, 2025

Abstract

The **Ca**tegorical **M**essage **P**assing **L**anguage (**CaMPL**) is a statically-typed concurrent programming language. The sequential side of CaMPL is functional in style, and the concurrent side uses message passing as its core communication model in contrast to shared memory. The semantics of CaMPL lies in a linear actegory which is a linearly distributive category with a distributive monoidal category acting on it.

To implement higher-order functions such as `map` or `fold` on the concurrent side, or to define concurrent type classes in CaMPL, it is necessary to pass processes as arguments to other processes. Although this ability is already present in any closed linear type system, such as CaMPL's, supporting arbitrary recursive process definitions requires the ability to reuse passed processes. However, concurrent resources in CaMPL are linear and therefore cannot be duplicated. As a result, passing processes as linear closures does not provide the required flexibility.

To solve this issue, one must pass processes as sequential data. This enables a process to receive another process as a sequential input or to transmit it as a message over a channel. However, this means that one must be able to convert a process into sequential data, and also be able to convert it back into a callable concurrent process. Categorically, this means that the concurrent side is *enriched* in its sequential side, with hom-objects in the sequential category representing stored processes. The abstract categorical machinery that characterizes the storing of concurrent processes as sequential data, and the using of the stored processes is the theory of copowers. This characterizes what we understand to be *higher-order message passing*.

In this thesis, we prove that giving a right-enriched category with copowers is equivalent to giving a right actegory with hom-objects. While this result is known in the closed symmetric setting, our contribution extends the equivalence to non-closed and

non-symmetric monoidal settings. This generalization is motivated by the semantics of higher-order message passing in CaMPL, which need not to be closed.

We have also implemented the `store` and `use` language constructs in CaMPL's compiler and abstract machine and we describe the changes made to each stage of CaMPL's compiler in order to support this extension.

Acknowledgement

I am deeply grateful to my supervisor, Dr. Robin Cockett, for his unwavering support throughout these years. I have learned a lot from him, and this thesis would not have been possible without his mentorship and supervision. I would also like to thank Dr. Robin Cockett, Dr. Kristine Bauer and Dr. Philipp Woelfel for serving on my thesis committee.

I am thankful to my friends and lab mates Alexanna Little, Saina Daneshmandjahromi and Durgesh Kumar for their advice, and support throughout this process. I am grateful to Ali Madani for his constant encouragement and offering valuable guidance. Finally, I wish to express my heartfelt thanks to my family for their endless love and support, and for enduring the difficult days of separation throughout these years.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The **Ca**tegorical **M**essage **P**assing **L**anguage (**CaMPL**) is a statically-typed concurrent programming language with a categorical semantics. The sequential side of CaMPL is functional in style, supporting features such as data types, codata types and pattern matching. The concurrent side uses *message passing* as its core communication model, in contrast to shared memory. A CaMPL program consists of a list of definitions, including sequential (co)data type declarations, sequential function definitions, concurrent (co)data type definitions—referred to as (co)protocols—and process definitions. Each process operates with a sequential input context and communicates through input and output polarity channels, which are typed using protocols.

CaMPL is based on the work of Cockett and Pastro [19], which provides both a formal proof theory of message passing and its categorical semantics. CaMPL was implemented by Prashant Kumar [25] and more recently, adding races, by Jared Pon [30] at the University of Calgary.

The semantics of CaMPL is given by a *linear actegory* [19], which consists of a monoidal category acting on a linearly distributive category [11] using two action functors:

$$\circ : \mathbb{A} \times \mathbb{X} \to \mathbb{X} \quad \text{and} \quad \bullet : \mathbb{A}^{op} \times \mathbb{X} \to \mathbb{X},$$

where $\circ$ is the parameterized left adjoint of $\bullet$, that is, for every object $A \in \mathbb{A}$, $A \circ - \dashv A \bullet - : \mathbb{X} \to \mathbb{X}$.

In CaMPL, the category $\mathbb{A}$ corresponds to the sequential side of the language, while $\mathbb{X}$ models the concurrent side. The functors $\circ$ and $\bullet$ allow for sending and receiving sequential messages (objects of $\mathbb{A}$) along concurrent channels (objects of $\mathbb{X}$). In the programming syntax, the $\circ$ functor corresponds to the concurrent type `Put`, and the $\bullet$ functor corresponds to the type `Get`. The parameterized adjunction means that a process behaves equivalently when it sends or receives a message on either input or output channels.

A desirable feature to incorporate into CaMPL is support for **higher-order processes**, that is, allowing concurrent processes to be passed as arguments to other processes, in other words, to be treated as "first-class citizens". This form of process passing is already supported on the concurrent side of CaMPL, as it is a $*$-*autonomous* category [2], which is closed, and thus inherently supports such higher-order behavior by

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A^{\perp} \oplus B := A \multimap B}$$

This allows a process $A \multimap B$ to be passed to another process through an input polarity channel of type $A^{\perp} \oplus B$ and be "applied" to an input channel A to yield an output channel B. This can be written in CaMPL as shown in the following code snippet:

```
proc apply :: | A, Neg(A) (+) B => B =
    | a, neg_a_b => b ->
        fork neg_a_b as
                neg_a -> neg_a |=| neg(a)
                b' -> b |=| b'
```

However, since channels are concurrent resources, they are linear and cannot be duplicated. Thus, passing a process as a concurrent channel does not support arbitrary

recursive process definitions that may require channel duplication. For example, consider a process that receives two input channels: one of type A and another of type $A^{\perp} \oplus A$, representing an input process of type $A \multimap A$. Additionally consider that it has an output channel of type A, along with a sequential counter n : `Int`. The goal is to apply the input process to the input channel n times. This suggests the following CaMPL code to define this process:

```
proc q :: Int | A, Neg(A) (+) A => A =
    n | x, p => y -> case n of
            0 -> x |=| y                    -- base case
            _ -> plug                       -- recursive case
                    apply(x, p => z)        -- apply p
                    q( n - 1 | z, p => y)   -- recurse
```

However, this code snippet is **not** a valid CaMPL program, as the channel p is used more than once, violating the linearity constraint imposed on concurrent resources.

The solution to the problem is to store the concurrent process as sequential data which can then be duplicated and used multiple times. To achieve this, we introduce two new language constructs:

- `store`: Converts a concurrent process into sequential data.
- `use`: Allows the calling of a stored process.

We define a new sequential data type $Store(\Phi|\Gamma \Vdash \Delta)$ to represent stored processes. Using these new constructs, the previous program can be rewritten correctly as follows:

```
proc q :: Int, Store( | A => A) | A => A =
    n, p | x => y -> case n of
            0 -> x |=| y                    -- base case
            _ -> plug                       -- recursive case
```

```
use(p)( | x => z)          -- use the stored process
q( n - 1, p | z => y)      -- recurse
```

The categorical semantics that enables storing concurrent processes as sequential data is given by an enrichment of the concurrent world in the sequential world. This is abstractly given by the equivalence between a right $\mathbb{A}$-actegory [9] with hom-objects and a right $\mathbb{A}$-enriched category with copowers. Specifically, if an $\mathbb{A}$-actegory is equipped with hom-objects, meaning that the action functor $\lhd : \mathbb{X} \times \mathbb{A} \to \mathbb{X}$ admits a right adjoint written as $X \lhd - \dashv Hom(X, -) : \mathbb{A} \to \mathbb{X}$, then it forms an $\mathbb{A}$-enriched category with copowers. This theorem was proven by Kelly and Janelidze [20] under the assumption that the category $\mathbb{A}$ is right-closed, and it was investigated more generally in a bicategorical setting by Gordon and Power [17]. In this thesis, we extend these results to non-closed (non-symmetric) case by defining what "having copowers" means in the non-closed setting based on [28].

An $\mathbb{A}$-enriched category $\mathbb{X}$ has copowers if, for any $A \in \mathbb{A}$ and $X \in \mathbb{X}$, there exists a morphism $\eta : A \to \mathrm{Hom}(X, X \lhd A)$ in $\mathbb{A}$, referred to as the *unit* of the copower, such that for each morphism $f : B \to \mathrm{Hom}(X \lhd A, Y)$, there exists a bijective correspondence

$$\frac{B \xrightarrow{f} \mathrm{Hom}(X \lhd A, Y)}{A \otimes B \xrightarrow{(\eta \otimes f)m} \mathrm{Hom}(X, Y)}$$

where m is the composition morphism in $\mathbb{A}$. This definition is equivalent, in the case that $\mathbb{A}$ is closed, to having a $\mathbb{A}$-natural isomorphism $\gamma : \mathrm{Hom}(X \lhd A, Y) \xrightarrow{\sim} A \multimap \mathrm{Hom}(X, Y)$.

We also prove that a category is both powered and copowered if and only if, for any $A \in \mathbb{A}$, the action functor has a parameterized left adjoint $- \lhd A \dashv A \blacktriangleright -$. This result is particularly important for us because such an adjunction exists in a linear actegory, which is the underlying structure of CaMPL's semantics.

1.1 Contribution of the Thesis

This thesis presents both the theoretical foundations and practical implementation of higher-order processes in CaMPL. A central contribution of this work is the proof that there is an equivalence between right $\mathbb{A}$-enriched categories with copowers and right $\mathbb{A}$-actegories equipped with hom-objects in cases where $\mathbb{A}$ is neither closed nor symmetric. This equivalence had not been previously proved in this setting. Furthermore, we prove that with some extra conditions in $\mathbb{X}$, a right $\mathbb{A}$-actegory with hom-objects is a linear/non-linear model [4].

Beyond the theoretical contributions, this project also involves the full integration of the `store` and `use` language constructs in CaMPL, which enable the storage of a process as sequential data and its arbitrary reuse. The design and implementation of these mechanisms are discussed in detail in Chapter 4 of this thesis.

1.2 Structure of the Thesis

This thesis consists of two main parts: the categorical semantics and theory behind higher-order processes, and the implementation details of higher-order processes in CaMPL. Specifically:

- **Chapter 2** presents the fundamental definitions from category theory used in the thesis.
- **Chapter 3** provides a theoretical perspective on higher-order processes, which is essential before proceeding to implementation. It defines their semantics and proves a theorem establishing an equivalence between a right actegory with hom-objects and enriched categories with copowers. It then provides a proof that shows that with some extra conditions on $\mathbb{X}$ (the category being act on), the $\mathbb{A}$-actegory with

hom-objects is a non-closed linear/non-linear model.

- **Chapter 4** details the implementation of higher-order processes in CaMPL. It examines different stages of the compiler – including parsing, type checking and inference, λ-lifting, pattern matching compilation, and the abstract machine – explaining the modifications made at each stage to integrate `store` and `use` commands.

- **Chapter 5** presents the conclusion and possible directions for future work.

The material discussed in Chapter 3 and parts of Chapter 2 of this thesis was also presented as a conference paper [12] at the Applied Category Theory Conference (ACT) 2025. The content in these chapters builds upon and extends the ideas introduced in that presentation, providing additional details, proofs, and discussions not included in the conference version.

1.3 Related Work

In this section, we examine related work on higher-order concurrency from both the category theory and programming languages perspectives. On the category theory side, we review the work of Kelly and Janelidze [20] on powers and copowers in monoidal closed categories, as well as the contributions of Gordon and Power [17], who investigate these structures within the framework of bicategories. From the programming languages perspective, we explore support for higher-order concurrency in the Haskell programming language and in Concurrent ML (CML) and the programming language created by Toninho *et al.* [32], which is based on the π-calculus and session types.

1.3.1 Related Work in Category Theory

1. **G. Janelidze and G. M. Kelly (2001)** [**20**]: In their paper, Janelidze and Kelly describe how an action of a monoidal category $\mathbb{V} = (\mathbb{V}, \otimes, I)$ on a category $\mathbb{A}$ corresponds to a strong monoidal functor:

$$F : \mathbb{V} \to [\mathbb{A}, \mathbb{A}].$$

Here $[\mathbb{A}, \mathbb{A}]$ is the category of endofunctors on $\mathbb{A}$, equipped with the strict monoidal structure $([\mathbb{A}, \mathbb{A}], \circ, 1_{\mathbb{A}})$, where $\circ$ denotes composition and $1_{\mathbb{A}}$ is the identity endofunctor. Such a functor F induces an action functor

$$* : \mathbb{V} \times \mathbb{A} \to \mathbb{A},$$

given by, for all $A \in \mathbb{A}$ and $X \in \mathbb{V}$, $X * A = F(X)(A)$.

They observe that in many practical cases, the functor F admits a right adjoint G, and this motivates a study of the conditions under which such a right adjoint exists. A necessary condition for the existence of G is that each functor $- * A : \mathbb{V} \to \mathbb{A}$ admits a right adjoint. This is characterized by a natural isomorphism:

$$\mathbb{A}(X * A, B) \cong \mathbb{V}(X, \mathbb{A}(A, B)).$$

They show that in the important case where $\mathbb{V}$ is a *right-closed* monoidal category, giving a category $\mathbb{A}$ together with an action of $\mathbb{V}$ such that each functor $- * A$ has a right adjoint is to endow $\mathbb{A}$ with the structure of a *tensored* $\mathbb{V}$-category.

Furthermore, they observe that in the case where $\mathbb{V}$ is a symmetric monoidal closed category, the existence of right adjoints for both $- * A$ and $X * -$ is equivalent to

$\mathbb{A}$ being a $\mathbb{V}$-category that is both *tensored* and *cotensored* [24].

2. **Gordon and Power (1997)** [17]: Gordon and Power study the theory of categories enriched over a bicategory $\mathcal{W}$, developing a framework that generalizes classical results such as Gabriel-Ulmer duality to this enriched setting. A central technical contribution of their work is the identification of an equivalence between the 2-category of $\mathcal{W}$-categories that admit tensors (i.e., copowers) and the 2-category of closed $\mathcal{W}$-representations. This equivalence holds under mild assumptions on the bicategory $\mathcal{W}$ and its subbicategories.

   They show that the existence of *tensors* (i.e., *copowers*) with respect to 1-cells of a locally dense subbicategory $\omega \subseteq \mathcal{W}$ allows one to embed $\mathcal{W}$-categories fully faithfully into a particular representation. Moreover, when both tensors and cotensors (i.e., both copowers and powers) exist, these enriched categories correspond to closed representations in the 2-category $\mathrm{Lax}(\mathcal{W}, \mathbf{Cat})$. The enriched version of Gabriel-Ulmer duality they develop depends crucially on the existence of such powers and copowers, showing that locally finitely presentable $\mathcal{W}$-categories can be described in terms of small $\mathcal{W}$-categories with finite colimits, as in the classical setting. This work emphasizes the foundational role of powers and copowers in enabling the passage between syntactic and semantic descriptions in enriched category theory over bicategories.

1.3.2 Related Work in Programming Languages

1. **Concurrent ML (John Reppy (1991) [31])**: Concurrent ML (CML) is an extension of Standard ML (SML) designed for high-level concurrent programming. CML integrates concurrency with SML's functional programming features, providing a model where threads are created dynamically and communicate synchronously

through typed channels. Channels are also created dynamically, and they are typed based on the value they carry, i.e. `'a chan` is the type of channel that carries value of type `'a`. Multiple threads can access the same channel–so they differ from channels in CaMPL.

Synchronous communication between threads is done using two primitive functions `send:'a chan * 'a -> unit` and `recv:'a chan -> 'a`. This means that a call `send(a, v)` will block until there is a matching `recv(a)` in another thread of computation. This is in contrast to CaMPL which is non-blocking.

A key shortcoming of CML is the lack of static guarantees against deadlocks or livelocks. CML relies entirely on the programmer to follow protocol conventions correctly. For instance, in a client-server interaction, the client must send a request before attempting to receive a reply, and must read exactly one reply before issuing another request. Violating these constraints can lead to out-of-sync communication and lack of progress (i.e. a deadlock).

CML introduces a model of higher-order concurrency based on first-class *event* values. These values represent potential communication actions and can be composed before being executed. Synchronization is not immediate; instead, it is explicitly triggered using the `sync` operation.

Basic operations such as `receive` and `transmit`, construct event values corresponding to input and output over channels. More expressive constructs are provided via higher-order combinators such as `wrap` and `choose` for composing and selecting events.

2. **Concurrent Haskell (S.P. Jones, A. Gordon and S. Finne (1996) [21])**: Concurrent Haskell is an extension to the purely-functional language Haskell that introduces concurrency for handling interactive and distributed systems.

The core of Concurrent Haskell revolves around two main additions: processes and atomically mutable state. Processes are spawned using the `forkIO` function, which initiates a concurrent action, allowing multiple processes to run independently. Communication between processes is done via *shared memory* which is implemented through a mutable structure called `MVar`, which holds a value that can be read or written atomically, ensuring safe interaction between processes. This model also allows the encapsulation of mutable state within the functional paradigm, enabling safe and controlled concurrency.

A significant aspect of Concurrent Haskell is its integration of concurrency with Haskell's existing monadic I/O system, which uses a state-transformer model to handle input and output operations. This makes the process of concurrency seamless within Haskell's functional framework. The language avoids the complexities of non-determinism, providing a clear separation between functional computations and concurrent operations.

The semantics of Concurrent Haskell are designed to maintain the determinism of functional programming while allowing for non-deterministic concurrency in I/O operations. This is achieved through a layered semantics that separates deterministic and concurrent behavior, ensuring that functional correctness is preserved even in concurrent applications.

3. **B. Toninho, L. Caires and F. Pfenning [32]**: This paper extends the authors' earlier work on session-type systems [8] by integrating higher-order message passing into a concurrent computation framework. In addition to the standard constructs of the π-calculus, their proposed language introduces a *contextual monad* to encapsulate open concurrent computations. These encapsulated computations can be both passed as values in functional programming and communicated between processes, supporting a uniform and symmetric treatment of higher-order functions

and higher-order processes.

Within this framework, a process P is viewed as offering a service of type A along a channel c, denoted $P :: c : A$. Processes may also consume services provided on other channels $c_1, \ldots, c_n$, each with its own session type A_i, yielding the linear typing judgment:

$$c_1 : A_1, \ldots, c_n : A_n \vdash P :: c : A$$

Here, each channel c_i must be used exactly once in P, respecting its corresponding session type A_i.

To support higher-order communication, they enrich the type system with *contextual monadic types*, written as $\{a : A \leftarrow a_i : A_i\}$, representing a process that offers session A along channel a, while making use of channels a_i at types A_i.

The key syntactic construct they introduce is the *internalization* of process expressions through the monadic form:

$$a \leftarrow \{P\} \leftarrow a_1, \ldots, a_n$$

This corresponds to the monadic `return` operation: a process P using channels $a_1, \ldots, a_n$ to provide a service along channel a. The monadic `bind` is represented as:

$$a \leftarrow M \leftarrow a_1, \ldots, a_n;\ P_a$$

which denotes the composition of a monadic object M (using channels $a_1, \ldots, a_n$) with a continuation process P_a that consumes the service on a. This construct spawns a new process to run M, providing a fresh channel a in parallel with P_a.

In CaMPL, their contextual monad can be modeled using the constructs `store`, `use`,

and `plug`. The monadic return corresponds to storing a process $p :: a_1, \ldots, a_n \Rightarrow a$, written as `store(p)`. The bind operation can be represented as:

```
plug
   use(p)( | a_1,...,a_n => a)
   p_a ( | a, b_1,...,b_n => c)
```

This code expresses the composition of a stored process `p` with a continuation `p_a` that uses channel `a` as input, closely mirroring the semantics of the monadic bind in their system. While the language provides a model for higher-order concurrency, it is based on a rather complex type theory and lacks a categorical semantics—unlike CaMPL, which is explicitly designed with a categorical semantics and a derived type system.

Chapter 2

Background

In this section, we first present the category theory definitions needed in this thesis, followed by a brief explanation of the Categorical Message Passing Language (CaMPL), highlighting its various constructs.

2.1 Categories

A category $\mathbb{C}$ consists of [27]

1. a collection of objects $\{A, B, C, ...\}$
2. a collection of maps (or morphisms) $f : A \to B, g : B \to C, ...$
3. an identity map $1_A : A \to A, \forall A \in \mathbb{C}$
4. and for each pair of maps $f : A \to B$ and $g : B \to C$, a composition morphism $fg : A \to C$

These data must satisfy the following axioms:

1. Identity Law: $\forall f : A \to B, 1_A f = f$ and $f 1_B = f$

2. Associativity Law $\forall f : A \to B, g : B \to C$ and $h : C \to D, (fg)h = f(gh)$

For any two objects $A, B \in Obj(\mathbb{C})$, the set of all morphisms from A to B is denoted as $\mathbb{C}(A, B)$ and is called the **hom-set**. A map $f : A \to B$, is called an **isomorphism** if there exists a map $f^{-1} : B \to A$ such that $ff^{-1} = 1_A$ and $f^{-1}f = 1_B$.

An example of a category is **Set**, in which the objects are sets and the maps are . The composition is the usual function composition, and the identity is the identity function.

For a category $\mathbb{C}$ the opposite category, referred to as $\mathbb{C}^{op}$, has the same objects as $\mathbb{C}$, but the direction of the maps is reversed, meaning that for any $f : A \to B$ in $\mathbb{C}$, $f^{op} : B \to A$ is in $\mathbb{C}^{op}$.

2.2 Functors

Given two categories $\mathbb{C}$ and $\mathbb{D}$, $F : \mathbb{C} \to \mathbb{D}$ is called a **functor** [27], and it consists of a map from the objects of $\mathbb{C}$ to the objects of $\mathbb{D}$ and a map from maps of $\mathbb{C}$ to the maps of $\mathbb{D}$, such that:

1. if $f : A \to B \in \mathbb{C}$ then $F(f) : F(A) \to F(B) \in \mathbb{D}$
2. $\forall A \in \mathbb{C}, F(1_A) = 1_{F(A)}$ (functors must preserve identity)
3. $\forall f : A \to B, g : B \to C \in \mathbb{C}, F(fg) = F(f)F(g)$ (functors must preserve composition)

An example of a functor is the list functor, which is an endofunctor on the category of sets, typically written as $L : \mathbf{Set} \to \mathbf{Set}$.

Lemma 2.2.1. *Categories and functors form a category* **Cat**.

Proof. **Cat** is a category in which objects are categories and maps are functors. Each category has an identity functor and the composition of functors is associative, so the lemma is immediate. □

If $F : \mathbb{C} \to \mathbb{D}$ is a functor, then $F^{\mathrm{op}} : \mathbb{C}^{\mathrm{op}} \to \mathbb{D}^{\mathrm{op}}$ is also a functor. A functor whose domain is the opposite category $\mathbb{C}^{\mathrm{op}}$ is called a **contravariant functor** from $\mathbb{C}$ as it reverses the direction of morphisms in $\mathbb{C}$. In contrast, a functor with domain $\mathbb{C}$ is sometimes referred to as a **covariant functor**, to emphasize that it preserves the direction of morphisms.

Functors may have a domain that is a product of two or more categories, which means they take more than one argument. Consider a functor $F : \mathbb{A} \times \mathbb{B} \to \mathbb{C}$. Fixing either argument of F at an object $A \in \mathbb{A}$ or $B \in \mathbb{B}$ yields new functors:

$$F_A = F(A, -) : \mathbb{B} \to \mathbb{C} \quad \text{and} \quad F_B = F(-, B) : \mathbb{A} \to \mathbb{C}$$

Conversely, given a family of such functors F_A and F_B we may reconstruct F.

Proposition 2.2.2. *To give a functor $F : \mathbb{A} \times \mathbb{B} \to \mathbb{C}$ is to give a family of functors $F_A : \mathbb{B} \to \mathbb{C}$ for each $A \in \mathbb{A}$ and $F_B : \mathbb{A} \to \mathbb{C}$ for each $B \in \mathbb{B}$ such that $F_A(B) = F_B(A)$ and for any $g : A \to A' \in \mathbb{A}$ and $f : B \to B' \in \mathbb{B}$, $F_A(f)F_{B'}(g) = F_B(g)F_{A'}(f)$.*

Proof. Let $F : \mathbb{A} \times \mathbb{B} \to \mathbb{X}$ be a functor, and define $F_A = F(A, -) : \mathbb{B} \to \mathbb{C}$ and $F_B = F(-, B) : \mathbb{A} \to \mathbb{C}$. Clearly $F_A(B) = F(A, B) = F_B(A)$ and $F_A(f)F'_B(g) = F(1, f)F(g, 1) = F(g, 1)F(1, f) = F_B(g)F'_A(f)$. Conversely let F_A and F_B be functors with the specified equalities. Define $F(f, g) = F_A(f)F_{B'}(g) = F_B(g)F_{A'}(f)$. We want to show that F is a functor. The proposed F satisfies the identity law because $F(1, 1) = F_A(1)F'_B(1) = F_B(1)F'_A(1) = 1$, and satisfies the composition law because:

$$\begin{aligned} F(f, g)F(f', g') &= F_A(f)F_{B'}(g)F_{A'}(f')F_{B''}(g') \\ &= F_A(f)F_A(f')F_{B''}(g)F_{B''}(g') \\ &= F_A(ff')F_{B''}(gg') \\ &= F(ff', gg') \end{aligned}$$

□

2.3 Natural Transformations

Given two functors $F, G : \mathbb{C} \to \mathbb{D}$, a natural transformation $\alpha : F \Rightarrow G$ is, for each $A \in \mathbb{C}$, a family of maps $\alpha_A : F(A) \to G(A)$ in $\mathbb{D}$ such that for any map $f : A \to A'$ in $\mathbb{C}$, the following diagram commutes:

$$\begin{array}{ccc} F(A) & \xrightarrow{\alpha_A} & G(A) \\ {\scriptstyle F(f)}\downarrow & & \downarrow{\scriptstyle G(f)} \\ F(A') & \xrightarrow[\alpha_A]{} & G(A') \end{array}$$

An example of a natural transformation is the flattening of a list written as $\text{flatten}_A : L(L(A)) \to L(A)$, which takes a list of lists and produces a single flattened list. It is natural because for any function $f : A \to B$, applying f to each element in a list of lists and then flattening is the same as first flattening the list and then applying f to each element; that is, the following diagram commutes:

$$\begin{array}{ccc} L(L(A)) & \xrightarrow{\text{flatten}_A} & L(A) \\ {\scriptstyle L(L(f))}\downarrow & & \downarrow{\scriptstyle L(f)} \\ L(L(B)) & \xrightarrow[\text{flatten}_B]{} & L(B) \end{array}$$

Proposition 2.3.1. $\mathbf{Cat}(\mathbb{C}, \mathbb{D})$ *is a category with functors as objects and natural transformations as maps.*

Proof. We want to show that the identity law holds and the composition is associative. We know that every functor has an identity transformation given by $1_{F(A)} : F(A) \to F(A)$. We define the composition of natural transformations simply as $(\alpha\beta)_A = \alpha_A \beta_A$. If this composition works then it is associative. It remains to check that the requirement on the

composite transformation holds. This can be seen by pasting the transformation squares together:

$$\begin{array}{ccc} F(C_1) & \xrightarrow{F(f)} & F(C_2) \\ {\scriptstyle \alpha_{C_1}}\downarrow & & \downarrow{\scriptstyle \alpha_{C_2}} \\ G(C_1) & \xrightarrow[G(f)]{} & G(C_2) \\ {\scriptstyle \beta_{C_1}}\downarrow & & \downarrow{\scriptstyle \beta_{C_2}} \\ H(C_1) & \xrightarrow[H(f)]{} & H(C_2) \end{array}$$

□

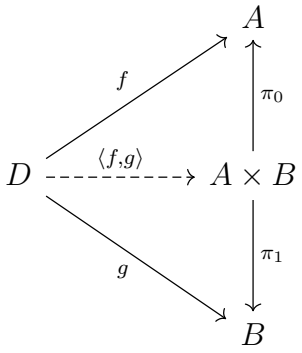
2.4 Initial and Final Objects

An **initial object** [10] in a category $\mathbb{C}$ is an object, often denoted by 0, such that for any object $A \in \mathbb{C}$, there is a unique map $?_A : 0 \to A$. Dually, a **final object** [10] in a category $\mathbb{C}$ is an object, often denoted by 1, such that for any object $A \in \mathbb{A}$, there is a unique map $!_A : A \to 1$.

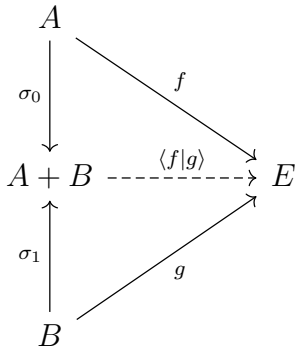
In **Set**, the initial object is the empty set, and the final object is the one element set. In **Cat**, the initial object is the empty category, and the final object is the category with one object and one map.

2.5 Products and Coproducts

Let A and B be any two objects in a category $\mathbb{C}$. The **product** [10] of A and B is an object $A \times B$ equipped with two maps $\pi_0 : A \times B \to A$ and $\pi_1 : A \times B \to B$ called the first and the second projection maps respectively such that, for any object $D \in \mathbb{C}$ with two maps $f : D \to A$ and $g : D \to B$, there is a unique map $\langle f, g \rangle : D \to A \times B$ such that the following diagram commutes:

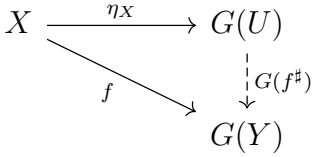


Dually, for two arbitrary objects A and B in a category $\mathbb{C}$, the **coproduct** [10] of A and B is an object $A + B$ in $\mathbb{C}$ equipped with two maps $\sigma_0 : A \to A + B$ and $\sigma_1 : B \to A + B$ called the first and the second coprojection maps respectively, such that, for any object $E \in \mathbb{C}$ with two maps $f : A \to E$ and $g : B \to E$, there is a unique map $\langle f|g \rangle : A + B \to E$ such that the following diagram commutes:



2.6 Adjoints

Let $G : \mathbb{Y} \to \mathbb{X}$ be a functor and X an object of $\mathbb{X}$, then a map $\eta_X : X \to G(U)$ has the **universal property** for the functor G at the object X if for any $f : X \to G(Y)$ there is a unique $f^\sharp : U \to Y$ such that the following diagram commutes:



An example of a universal property is the following [10]: let $\mathbb{X}$ denote the category of directed graphs and $\mathbb{Y}$ the category of categories. Consider the functor $G : \mathbb{Y} \to \mathbb{X}$,

which sends each category to its underlying directed graph by forgetting the composition and identity structure. The map that assigns to each directed graph the inclusion into the underlying graph of its free (or path) category—by identifying its edges with paths of length one—satisfies the universal property with respect to the functor G. That is, this inclusion is universal among all functors from $\mathbb{X}$ to $\mathbb{Y}$ that compose with G to yield the identity on $\mathbb{X}$.

Dual to the universal property, a map $\epsilon_Y : F(V) \to Y$, has the **couniversal property** for the functor F at the object Y if for any $g : F(X) \to Y$ there is a unique $g^\flat : X \to V$ such that the following diagram commutes:

$$\begin{array}{ccc} F(V) & \xrightarrow{\epsilon_Y} & Y \\ {\scriptstyle F(g^\flat)}\uparrow & \nearrow_{g} & \\ F(X) & & \end{array}$$

Let $F : \mathbb{X} \to \mathbb{Y}$ and $\mathbb{G} : \mathbb{Y} \to \mathbb{X}$ be functors, then F is a **left adjoin**t to G if there is a natural transformation $\eta_X : X \to G(F(X))$ with the universal property called the unit of the adjunction. Dually, G is right adjoint to F if there is a natural transformation $\epsilon_Y : F(G(Y)) \to Y$ with the couniversal property, called the counit of the adjunction.

In an adjunction $(\eta, \epsilon) : F \vdash G : \mathbb{X} \to \mathbb{Y}$, the following diagrams commute:

$$\begin{array}{ccc} G(Y) & \xrightarrow{\eta_{G(Y)}} & G(F(G(Y))) \\ & \searrow\!\!\searrow & \downarrow{\scriptstyle G(\epsilon)} \\ & & G(Y) \end{array} \qquad \begin{array}{ccc} F(X) & \xrightarrow{F(\eta_X)} & F(G(F(X))) \\ & \searrow\!\!\searrow & \downarrow{\scriptstyle \epsilon_{F(X)}} \\ & & F(X) \end{array}$$

these commuting diagrams result in two identities $\eta_{G(Y)}G(\epsilon_Y) = 1_{G(Y)}$ and $F(\eta_X)\epsilon_{F(X)} = 1_{F(X)}$ which are called the **triangle identities**.

Having an adjunction $(\eta, \epsilon) : F \dashv G : \mathbb{X} \to \mathbb{Y}$, is equivalent to having a $(-)^\sharp$ and $(-)^\flat$

operators where [10]:

$$\frac{X \xrightarrow{f=g^\flat} G(Y)}{F(X) \xrightarrow{g=f^\sharp} Y}$$

such that:

- $((f)^\sharp)^\flat = f$ and $((g)^\flat)^\sharp = g$,
- $(F(h)fk)^\flat = hf^\flat G(k)$ and $(hgG(k))^\sharp = F(h)g^\sharp k$.

Here is an example of an adjunction: If the category $\mathbb{C}$ has chosen binary products, the diagonal functor $\Delta : \mathbb{C} \to \mathbb{C} \times \mathbb{C}$ is left adjoint to the product functor, written as:

$$\Delta \dashv - \times - : \mathbb{C} \to \mathbb{C} \times \mathbb{C}$$

In this adjunction the unit is the diagonal map $\Delta : A \to A \times A$ and the counit is the pair of projections $(\pi_0, \pi_1) : (A \times B, A \times B) \to (A, B)$. The triangle identities in this adjunction can be expressed as follows:

$$\begin{array}{ccc} A \times B & \xrightarrow{\Delta} & (A \times B) \times (A \times B) \\ & \searrow & \downarrow{\scriptstyle \pi_0 \times \pi_1} \\ & & A \times B \end{array} \qquad \begin{array}{ccc} (A, A) & \xrightarrow{(\Delta, \Delta)} & (A \times A, A \times A) \\ & \searrow & \downarrow{\scriptstyle (\pi_0, \pi_1)} \\ & & (A, A) \end{array}$$

The left commuting diagram gives us the identity $\Delta(\pi_0, \pi_1) := \langle \pi_0, \pi_1 \rangle = 1_{A \times B}$, which is called "surjective pairing".

2.6.1 Parameterized Adjoints

Let $\mathbb{X}, \mathbb{Y}, \mathbb{Z}$ be categories. An **$\mathbb{X}$-parameterized adjunction** [27] between $\mathbb{Y}$ and $\mathbb{Z}$ consists of a pair of functors

$$F : \mathbb{Z} \times \mathbb{X} \to \mathbb{Y}, \qquad G : \mathbb{X}^{\mathrm{op}} \times \mathbb{Y} \to \mathbb{Z}$$

such that for every object $X \in \mathbb{X}$, the functor $F_X := F(-, X) : \mathbb{Z} \to \mathbb{Y}$ has a right adjoint $G_X := G(X, -) : \mathbb{Y} \to \mathbb{Z}$, and the adjunction bijection

$$\frac{F(Z, X) \to Y}{\overline{\overline{Z \to G(X, Y)}}}$$

is natural in $Y \in \mathbb{Y}$, and $Z \in \mathbb{Z}$.

Lemma 2.6.1. [1] *If $F : \mathbb{Z} \times \mathbb{X} \to \mathbb{Y}$ is a functor and if, for each $X \in \mathbb{X}$, $F(-, X)$ has a right adjoint such that $F(-, X) \dashv G(X, -) : \mathbb{Z} \to \mathbb{Y}$ then $G : \mathbb{X}^{op} \times \mathbb{Y} \to \mathbb{Z}$ is a functor where, for all $f \in \mathbb{X}$ and $g \in \mathbb{Y}$, $G(f, g) := (F(1, f)\epsilon g)^\flat$.*

Proof. We want to show that G is a functor, so we have to show that it satisfies the unit and composition laws.
We have:

$$\begin{aligned} G(1_X, 1_Z) &= (F(1_{G(X,Z)}, 1_X)\epsilon)^\flat \\ &= \epsilon^\flat \\ &= 1_{G(X,Z)} \end{aligned}$$

Which proves the unit law.
And if we have $f : X \to X'$, $f' : X' \to X''$ and $g : Y \to Y'$ and $g' : Y' \to Y''$, then we

[1] This lemma was proved in [27]

can write:

$$\begin{aligned}
G(ff', gg') &= (F(ff', 1)\epsilon gg')^\flat \\
&= (F(f, 1)(F(f', 1)\epsilon g)g')^\flat \\
&= (F(f, 1)G(f', g)^\sharp g')^\flat \\
&= (F(f, 1)F(1, G(f', g))\epsilon g')^\flat \\
&= (F(1, G(f', g))F(f, 1)\epsilon g')^\flat \\
&= G(f', g)(F(f, 1)\epsilon g')^\flat \\
&= G(f', g)G(f, g')
\end{aligned}$$

Which proves the composition law. □

Corollary 2.6.2. *In a parameterized adjunction, the unit η and the counit ϵ are dinatural, meaning that $F(f, 1)\epsilon = F(1, G(f, 1))\epsilon$ and $\eta G(f, 1) = \eta G(1, F(f, 1))$. That is the following diagrams commute:*

$$\begin{array}{ccc}
Y & \xrightarrow{\eta} & G(X', F(X', Y)) \\
{\scriptstyle\eta}\downarrow & & \downarrow{\scriptstyle G(f,1)} \\
G(X, F(X, Y)) & \xrightarrow[G(1,F(f,1))]{} & G(X, F(X', Y))
\end{array}
\qquad
\begin{array}{ccc}
F(X, G(X', Y)) & \xrightarrow{F(f,1)} & F(X', G(X', Y)) \\
{\scriptstyle F(1,G(f,1))}\downarrow & & \downarrow{\scriptstyle\epsilon} \\
F(X, G(X, Y)) & \xrightarrow[\epsilon]{} & Y
\end{array}$$

Proof. To prove this, it is enough to use Lemma 2.6.1 as shown:

$$\begin{aligned}
\eta G(1, F(f,1)) &= \eta(\epsilon F(f,1))^\flat && \text{(Lemma 2.6.1)} \\
&= \eta(F(1, G(1, F(f,1)))\epsilon)^\flat && \text{(naturality of } \epsilon) \\
&= \eta(F(1, (F(f,1)\epsilon)^\flat)\epsilon)^\flat && \text{(Lemma 2.6.1)} \\
&= \eta(F(1, f(\epsilon)^\flat)\epsilon)^\flat && \text{(property of } (-)^\flat) \\
&= \eta(F(1,f)\epsilon)^\flat && (\epsilon^\flat = 1) \\
&= \eta G(f,1) && \text{(Lemma 2.6.1)}
\end{aligned}$$

$$\begin{aligned}
F(f,1)\epsilon &= ((F(f,1)\epsilon)^\flat)^\sharp \\
&= (G(f,1))^\sharp \\
&= F(1, G(f,1))\epsilon
\end{aligned}$$

□

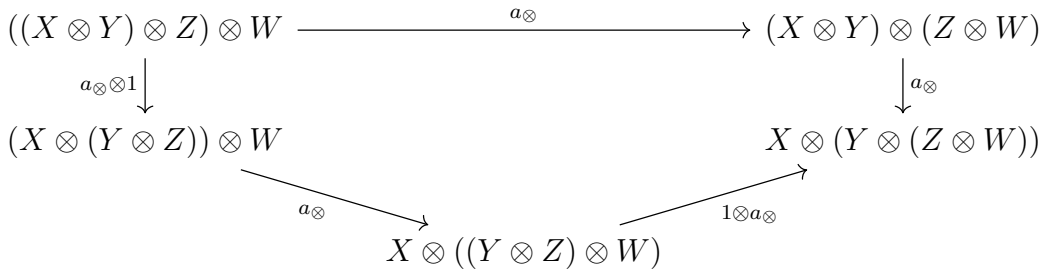
2.7 Monoidal Categories

A **monoidal category** [3] $\mathbb{X}$, is a category equipped with

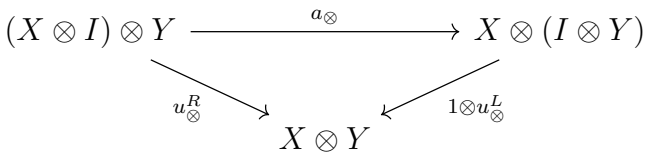
1. a functor $\otimes : \mathbb{X} \times \mathbb{X} \to \mathbb{X}$ called the "tensor product"
2. an object $I \in \mathbb{X}$ called the "tensor unit"
3. a natural isomorphism $a_\otimes : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$ called the "associator"
4. and two natural isomorphisms $u^L_\otimes : I \otimes X \to X$ and $u^R_\otimes : X \otimes I \to X$ called the "left unitor" and the "right unitor" respectively.

In addition, the following diagrams must commute:

1. Associativity of $a_\otimes$:

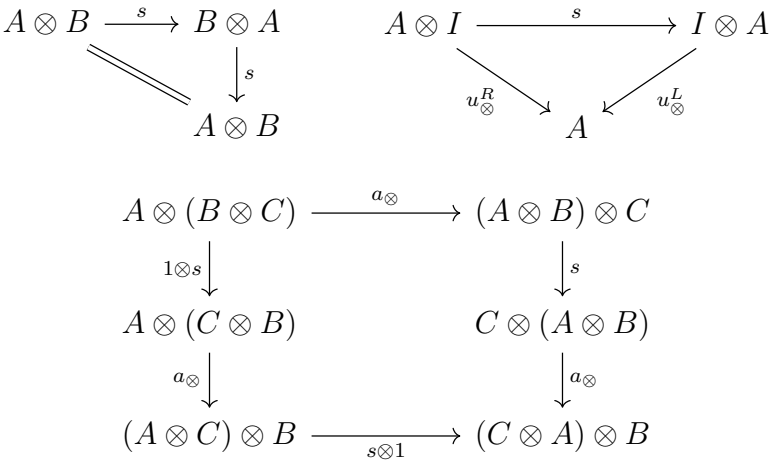


2. Identity:



An example of a monoidal category is **Set**, where the tensor product is given by the cartesian product and the unit object is the singleton set.

A monoidal category is **symmetric** if it also has a natural isomorphism $s : A \otimes B \to B \otimes A$ and the following diagrams commute:



2.7.1 Closed Monoidal Categories

A monoidal category $\mathbb{A}$ is said to be **right-closed** [3] if for any $A \in \mathbb{A}$, the functor $A \otimes -$ has a right adjoint which we write as $A \otimes - \dashv A \multimap - : \mathbb{A} \to \mathbb{A}$, meaning that there exists a bijective correspondence, natural in B and C,

$$\frac{A \otimes B \xrightarrow{f=\text{uncurry}(g)} C}{B \xrightarrow{g=\text{curry}(f)} A \multimap C}$$

In the **left-closed** case, for each $B \in \mathbb{B}$ there is a bijective correspondence, natural in A and C,

$$\frac{A \otimes B \xrightarrow{f=\text{uncurry}(g)} C}{A \xrightarrow{g=\text{curry}(f)} C \circ\!\!- B}$$

In the symmetric case, $B \multimap C \cong C \circ\!\!- B$.

2.7.2 Cartesian Categories

A **cartesian category** [15] is a monoidal category where the tensor product is given by the categorical product, and the unit object is the final object. For any object A in a cartesian category, there is a diagonal map $\Delta_A : A \to A \times A$, arising from the universal property of the product (see 2.6), and a map $e_A : A \to I$, where I is the unit object, since I is terminal. In the context of computer science, Δ can be interpreted as "duplicating data," while e corresponds to "deleting data."

2.7.3 Monoidal Functors and Natural Transformations

Let $(\mathbb{X}, \otimes_{\mathbb{X}}, I_{\mathbb{X}})$ and $(\mathbb{Y}, \otimes_{\mathbb{Y}}, I_{\mathbb{Y}})$ be two monoidal categories. A **monoidal functor** [14] functor between them is

1. a functor $F : \mathbb{X} \to \mathbb{Y}$

2. a natural transformation $m_\otimes : F(X) \otimes_\mathbb{Y} F(Y) \to F(X \otimes_\mathbb{X} Y)$

3. a map $m_I : I_\mathbb{Y} \to F(I_\mathbb{X})$

such that the following diagrams commute:

1. (Associativity)

$$\begin{array}{ccc}
(F(X) \otimes_\mathbb{Y} F(Y)) \otimes_\mathbb{Y} F(Z) & \xrightarrow{a_{\otimes_\mathbb{Y}}} & F(X) \otimes_\mathbb{Y} (F(Y) \otimes_\mathbb{Y} F(Z)) \\
\downarrow{\scriptstyle m_\otimes \otimes_\mathbb{Y} 1} & & \downarrow{\scriptstyle 1 \otimes_\mathbb{Y} m_\otimes} \\
F(X \otimes_\mathbb{X} Y) \otimes_\mathbb{Y} F(Z) & & F(X) \otimes_\mathbb{Y} F(Y \otimes_\mathbb{X} Z) \\
\downarrow{\scriptstyle m_\otimes} & & \downarrow{\scriptstyle m_\otimes} \\
F((X \otimes_\mathbb{X} Y) \otimes_\mathbb{X} Z) & \xrightarrow[F(a_{\otimes_\mathbb{X}})]{} & F(X \otimes_\mathbb{X} (Y \otimes_\mathbb{X} Z))
\end{array}$$

2. (Unitality)

$$\begin{array}{ccc}
I_\mathbb{Y} \otimes_Y F(X) & \xrightarrow{m_I \otimes_\mathbb{Y} 1} & F(I_\mathbb{X}) \otimes_Y F(X) \\
\downarrow{\scriptstyle u^L_{\otimes_\mathbb{Y}}} & & \downarrow{\scriptstyle m_\otimes} \\
F(X) & \xleftarrow[F(u^L_{\otimes_\mathbb{X}})]{} & F(I_\mathbb{X} \otimes_\mathbb{X} X)
\end{array}
\qquad
\begin{array}{ccc}
F(X) \otimes_Y I_\mathbb{Y} & \xrightarrow{1 \otimes_\mathbb{Y} m_I} & F(X) \otimes_Y F(I_\mathbb{X}) \\
\downarrow{\scriptstyle u^R_{\otimes_\mathbb{Y}}} & & \downarrow{\scriptstyle m_\otimes} \\
F(X) & \xleftarrow[F(u^R_{\otimes_\mathbb{X}})]{} & F(X \otimes_\mathbb{X} I_\mathbb{X})
\end{array}$$

When $m_\otimes$ and m_I are isomorphisms, F is called a **strong monoidal** functor.

Given two monoidal Functors $F, G : \mathbb{X} \to \mathbb{Y}$, a natural transformation $\alpha : F \Rightarrow G$ is a **monoidal natural transformation** if for any $X, Y \in \mathbb{X}$ the following diagrams commute:

$$\begin{array}{ccc}
F(X) \otimes_\mathbb{Y} F(Y) & \xrightarrow{\alpha_X \otimes_\mathbb{Y} \alpha_Y} & G(X) \otimes_\mathbb{Y} G(Y) \\
\downarrow{\scriptstyle m^F_\otimes} & & \downarrow{\scriptstyle m^G_\otimes} \\
F(X \otimes_\mathbb{X} Y) & \xrightarrow[\alpha_{X \otimes_\mathbb{X} Y}]{} & G(X \otimes_\mathbb{X} Y)
\end{array}
\qquad
\begin{array}{ccc}
 & I_\mathbb{Y} & \\
{\scriptstyle m^F_I} \swarrow & & \searrow {\scriptstyle m^G_I} \\
F(I_\mathbb{X}) & \xrightarrow[\alpha_{I_\mathbb{X}}]{} & G(I_\mathbb{X})
\end{array}$$

2.7.4 Monoidal Adjunctions

If $\mathbb{X}$ and $\mathbb{Y}$ are monoidal categories then a monoidal adjunction [23] between them is an ordinary adjunction $(\eta, \epsilon), F \dashv G : \mathbb{X} \to \mathbb{Y}$ in which both F and G are monoidal functors and η (unit) and ϵ (counit) are monoidal natural transformations. For η and ϵ to be monoidal natural transformations, the following diagrams must commute:

$$\begin{array}{ccc} X \otimes Y & \xrightarrow{\eta \otimes \eta} & G(F(X)) \otimes G(F(Y)) \\ {\scriptstyle \eta}\downarrow & & \downarrow{\scriptstyle m^G_\otimes} \\ G(F(X \otimes Y)) & \xleftarrow[G(m^F_\otimes)]{} & G(F(X) \otimes F(Y)) \end{array} \qquad \begin{array}{ccc} F(G(X)) \otimes F(G(Y)) & \xrightarrow{m^F_\otimes} & F(G(X) \otimes G(Y)) \\ {\scriptstyle \epsilon \otimes \epsilon}\downarrow & & \downarrow{\scriptstyle F(m^G_\otimes)} \\ X \otimes Y & \xleftarrow[\epsilon]{} & F(G(X \otimes Y)) \end{array}$$

$$\begin{array}{ccc} I & \xrightarrow{\eta} & G(F(I)) \\ & {\scriptstyle m^G_I}\searrow \quad \uparrow{\scriptstyle G(m^F_I)} & \\ & G(I) & \end{array} \qquad \begin{array}{ccc} F(G(I)) & \xrightarrow{\epsilon} & I \\ {\scriptstyle F(m^G_I)}\uparrow & \swarrow{\scriptstyle m^F_I} & \\ F(I) & & \end{array}$$

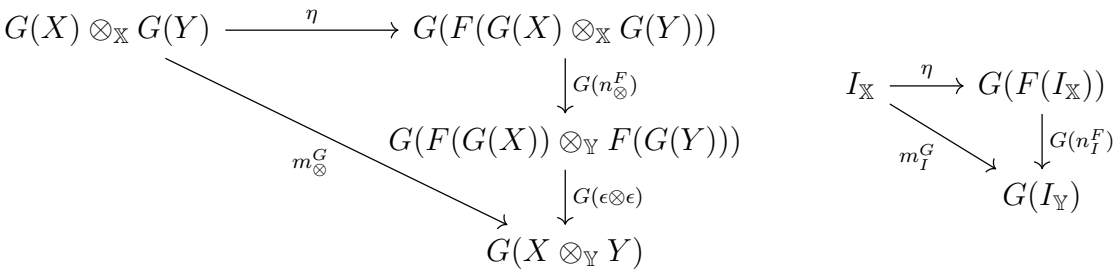
Proposition 2.7.1. *Given an adjunction $(\eta, \epsilon), F \dashv G : \mathbb{X} \to \mathbb{Y}$, in which $\mathbb{X}$ and $\mathbb{Y}$ are monoidal categories, $(G, m^G_\otimes, m^G_I)$ is monoidal if and only if $(F, n^F_\otimes, n^F_I)$ is comonoidal and $m^G_\otimes, n^F_\otimes$ and m^G_I, n^F_I are mates. Furthermore, η and ϵ satisfy the following coherences:*

$$\begin{array}{ccc} X \otimes Y & \xrightarrow{\eta \otimes \eta} & G(F(X)) \otimes G(F(Y)) \\ {\scriptstyle \eta}\downarrow & & \downarrow{\scriptstyle m^G_\otimes} \\ G(F(X \otimes Y)) & \xrightarrow[G(n^F_\otimes)]{} & G(F(X) \otimes F(Y)) \end{array} \qquad \begin{array}{ccc} F(G(X)) \otimes F(G(Y)) & \xleftarrow{n^F_\otimes} & F(G(X) \otimes G(Y)) \\ {\scriptstyle \epsilon \otimes \epsilon}\downarrow & & \downarrow{\scriptstyle F(m^G_\otimes)} \\ X \otimes Y & \xleftarrow[\epsilon]{} & F(G(X \otimes Y)) \end{array}$$

$$\begin{array}{ccc} I & \xrightarrow{\eta} & G(F(I)) \\ & {\scriptstyle m^G_I}\searrow \quad \downarrow{\scriptstyle G(n^F_I)} & \\ & G(I) & \end{array} \qquad \begin{array}{ccc} F(G(I)) & \xrightarrow{\epsilon} & I \\ {\scriptstyle F(m^G_I)}\uparrow & \nearrow{\scriptstyle n^F_I} & \\ F(I) & & \end{array}$$

Proof. We begin by proving the forward implication. Assuming that F is comonoidal, we define the monoidal structure maps of G to be the mates of $n^F_\otimes : F(X \otimes_{\mathbb{X}} Y) \to$

$F(X) \otimes_{\mathbb{Y}} F(Y)$ and $n_I^F : F(I_{\mathbb{X}}) \to I_{\mathbb{Y}}$ as below:



We first have to prove that $m_{\otimes}^G := \eta G(n_I^F) G(\epsilon \otimes \epsilon)$ and $m_I^G := \eta G(n_I^F)$ satisfy the conditions mentioned in Section 2.7.3. Specifically, we start by proving associativity of $m_{\otimes}^G$. We must prove $a_{\otimes}(1 \otimes m_{\otimes}^G) m_{\otimes}^G = (m_{\otimes}^G \otimes 1) m_{\otimes}^G G(a_{\otimes})$. We have:

$$
\begin{aligned}
a_\otimes(1 \otimes m^G_\otimes)m^G_\otimes &= a_\otimes(1 \otimes m^G_\otimes)\eta G(n^F_\otimes)G(\epsilon \otimes \epsilon) && \text{(definition of } m^G_\otimes\text{)} \\
&= a_\otimes \eta G(F(1 \otimes m^G_\otimes))G(n^F_\otimes)G(\epsilon \otimes \epsilon) && \text{(naturality of } \eta\text{)} \\
&= a_\otimes \eta G(F(1 \otimes m^G_\otimes)n^F_\otimes(\epsilon \otimes \epsilon) \\
&= a_\otimes \eta G(n^F_\otimes(1 \otimes F(m^G_\otimes))(\epsilon \otimes \epsilon) && \text{(naturality of } n^F_\otimes\text{)} \\
&= a_\otimes \eta G(n^F_\otimes(\epsilon \otimes F(m^G_\otimes)\epsilon) \\
&= a_\otimes \eta G(n^F_\otimes(\epsilon \otimes F(\eta G(n^F_\otimes)(\epsilon \otimes \epsilon))\epsilon) && \text{(definition of } m^G_\otimes\text{)} \\
&= a_\otimes \eta G(n^F_\otimes(\epsilon \otimes F(\eta)\epsilon n^F_\otimes(\epsilon \otimes \epsilon)) && \text{(naturality of } \epsilon\text{)} \\
&= a_\otimes \eta G(n^F_\otimes(\epsilon \otimes n^F_\otimes(\epsilon \otimes \epsilon)) && \text{(triangle identity)} \\
&= a_\otimes \eta G(n^F_\otimes(1^F_\otimes)(\epsilon \otimes (\epsilon \otimes \epsilon)) \\
&= \eta G(F(a_\otimes)n^F_\otimes(1^F_\otimes)(\epsilon \otimes (\epsilon \otimes \epsilon)) && \text{(naturality of } \eta\text{)} \\
&= \eta G(n^F_\otimes(n^F_\otimes \otimes 1)a_\otimes(\epsilon \otimes (\epsilon \otimes \epsilon)) && \text{(associativity of } n^F_\otimes\text{)} \\
&= \eta G(n^F_\otimes(n^F_\otimes \otimes 1)((\epsilon \otimes \epsilon) \otimes \epsilon)a_\otimes) && \text{(naturality of } a_\otimes\text{)} \\
&= \eta G(n^F_\otimes(F(\eta)^F_\otimes(\epsilon \otimes \epsilon) \otimes \epsilon)a_\otimes) && \text{(triangle identity)} \\
&= \eta G(n^F_\otimes(F(\eta)^F_\otimes(\epsilon \otimes \epsilon) \otimes \epsilon)a_\otimes) && \text{(triangle identity)} \\
&= \eta G(n^F_\otimes(F(\eta G(n^F_\otimes(\epsilon \otimes \epsilon)))\epsilon \otimes \epsilon)a_\otimes) && \text{(naturality of } \epsilon\text{)} \\
&= \eta G(n^F_\otimes(F(\eta G(n^F_\otimes(\epsilon \otimes \epsilon))) \otimes 1)(\epsilon \otimes \epsilon)a_\otimes) \\
&= \eta G(n^F_\otimes(F(\eta G(n^F_\otimes(\epsilon \otimes \epsilon))) \otimes F(1))(\epsilon \otimes \epsilon)a_\otimes) \\
&= \eta G(n^F_\otimes(F(\eta G(n^F_\otimes(\epsilon \otimes \epsilon))) \otimes F(1))(\epsilon \otimes \epsilon)a_\otimes) \\
&= \eta G(F(\eta G(n^F_\otimes(\epsilon \otimes \epsilon))) \otimes 1)n^F_\otimes(\epsilon \otimes \epsilon)a_\otimes) && \text{(naturality of } n^F_\otimes\text{)} \\
&= (\eta G(n^F_\otimes(\epsilon \otimes \epsilon)) \otimes 1)\eta G(n^F_\otimes(\epsilon \otimes \epsilon)a_\otimes) && \text{(naturality of } \eta\text{)} \\
&= (m^G_\otimes \otimes 1)m^G_\otimes G(a_\otimes) && \text{(definition of } m^G_\otimes\text{)}
\end{aligned}
$$

And next we will prove that $(m_I^G \otimes 1)m_\otimes^G G(u_\otimes^L) = u_\otimes^L$. We have:

$$\begin{aligned}
(m_I^G \otimes 1)m_\otimes^G G(u_\otimes^L) &= (m_I^G \otimes 1)\eta G(n_\otimes^F(\epsilon \otimes \epsilon)u_\otimes^L) && \text{(definition of } m_\otimes^G\text{)} \\
&= \eta G(F(m_I^G \otimes 1)n_\otimes^F(\epsilon \otimes \epsilon)u_\otimes^L) && \text{(naturality of } \eta\text{)} \\
&= \eta G(n_\otimes^F F(m_I^G \otimes F(1))(\epsilon \otimes \epsilon)u_\otimes^L) && \text{(naturality of } n_\otimes^F\text{)} \\
&= \eta G(n_\otimes^F F(m_I^G \epsilon \otimes 1)(1 \otimes \epsilon)u_\otimes^L) \\
&= \eta G(n_\otimes^F F(\eta G(n_I^F)\epsilon \otimes 1)(1 \otimes \epsilon)u_\otimes^L) && \text{(definition of } m_I^G\text{)} \\
&= \eta G(n_\otimes^F F(\eta)\epsilon(n_I^F \otimes 1)(1 \otimes \epsilon)u_\otimes^L) && \text{(naturality of } \epsilon\text{)} \\
&= \eta G(n_\otimes^F F(\eta)\epsilon(n_I^F \otimes 1)u_\otimes^L \epsilon) && \text{(naturality of } u_\otimes^L\text{)} \\
&= \eta G(n_\otimes^F (n_I^F \otimes 1)u_\otimes^L \epsilon) && \text{(triangle identity)} \\
&= \eta G(n_\otimes^F n_\otimes^F F(u_\otimes^L)(\epsilon) && ((n_I^F \otimes 1)u_\otimes^L = n_\otimes^F F(u_\otimes^L)) \\
&= \eta G(F(u_\otimes^L)\epsilon) \\
&= u_\otimes^L \eta G(\epsilon) && \text{(naturality of } \eta\text{)} \\
&= u_\otimes^L && \text{(triangle identity)}
\end{aligned}$$

and proving $(1 \otimes m_I^G)m_\otimes^G G(u_\otimes^R) = u_\otimes^R$ is done similarly.

Now we have to prove the second part of the proposition, which is to prove that η and ϵ satisfy the specified coherences.

For η we first have to show that $(\eta \otimes \eta)m_\otimes^G = \eta G(n_\otimes^F)$.

$$\begin{aligned}
(\eta \otimes \eta)m_\otimes^G &= (\eta \otimes \eta)\eta G(n_\otimes^F)G(\epsilon \otimes \epsilon) && \text{(definition of } m_\otimes^G\text{)} \\
&= \eta G(F(\eta \otimes \eta)n_\otimes^F(\epsilon \otimes \epsilon)) && \text{(naturality of } \eta\text{)} \\
&= \eta G(n_\otimes^F(F(\eta) \otimes F(\eta))(\epsilon \otimes \epsilon)) && \text{(naturality of } n_\otimes^F\text{)} \\
&= \eta G(n_\otimes^F) && \text{(triangle identity)}
\end{aligned}$$

We also have to show that $m_I^G = \eta G(n_I^F)$ which holds by the definition of m_I^G. For ϵ we first have to show that $F(m_\otimes^G)\epsilon = n_\otimes^F(\epsilon \otimes \epsilon)$:

$$\begin{aligned} F(m_\otimes^G)\epsilon &= F(\eta)F(G(n_\otimes^F(\epsilon \otimes \epsilon)))\epsilon && \text{(definition of } m_\otimes^G\text{)} \\ &= F(\eta)\epsilon n_\otimes^F(\epsilon \otimes \epsilon) && \text{(naturality of } \epsilon\text{)} \\ &= n_\otimes^F(\epsilon \otimes \epsilon) && \text{(triangle identity)} \end{aligned}$$

It remains to show that $F(m_I^G)\epsilon = n_I^F$:

$$\begin{aligned} F(m_I^G)\epsilon &= F(\eta)F(G(n_I^F))\epsilon && \text{(definition of } m_I^G\text{)} \\ &= F(\eta)\epsilon n_I^F && \text{(naturality of } \epsilon\text{)} \\ &= n_I^F && \text{(triangle identity)} \end{aligned}$$

Therefore, we proved that G is monoidal and η and ϵ satisfy the mentioned coherences. Dually, The backward implication of the proposition holds.

□

Corollary 2.7.2. *Given an adjunction $(\eta, \epsilon) : F \dashv G : \mathbb{X} \to \mathbb{Y}$, if F is strong monoidal, then G is monoidal with respect to the mates of the monoidal structure maps of F, and the adjunction is monoidal.*

2.7.5 Linear/Non-Linear Adjunctions

A (non-closed) linear/non-linear (LNL) adjunction [5] consists of

(i) a cartesian category $(\mathbb{C}, 1, \times)$

(ii) a symmetric monoidal category $(\mathbb{L}, I, \otimes)$

(iii) a pair of symmetric monoidal functors $(G, m^G_\otimes, m^G_I) : \mathbb{L} \to \mathbb{C}$ and $(F, n^F_\otimes, n^F_I) : \mathbb{C} \to \mathbb{L}$ between them which form a symmetric monoidal adjunction $F \dashv G$.

A **linear category** [5] is a symmetric monoidal category $(\mathbb{X}, \otimes, I)$ with a symmetric monoidal comonad $(!, \epsilon, \delta, m_\otimes, m_I)$ *on* $\mathbb{X}$ and two monoidal natural transformations $e_x : !X \to I$ and $d_X : !X \to !X \otimes !X$ such that

(i) Each $(!X, e_X, d_X)$ is a commutative comonoid.

(ii) e_X and d_X are coalgebra maps.

(iii) All coalgebra maps between free coalgebras preserve the comonoid structure.

In [5], Benton shows that in a linear/non-linear adjunction $(\eta, \epsilon), F \dashv G : \mathbb{A} \to \mathbb{X}$, the category $\mathbb{X}$ forms a linear category, where the comonad is given by $!X := F(G(X))$ which is clearly monoidal since F and G are monoidal, and the maps e_X and d_X are defined as follows and satisfy the required coherences.

$$\begin{array}{ccc} F(G(X)) & \xrightarrow{e_X} & I \\ {\scriptstyle F(*_{G(X)})}\downarrow & \nearrow_{(m^F_I)^{-1}} & \\ F(1) & & \end{array} \qquad \begin{array}{ccc} F(G(X)) & \xrightarrow{d_X} & F(G(X)) \otimes F(G(X)) \\ {\scriptstyle F(\Delta)}\downarrow & \nearrow_{(m^F_\otimes)^{-1}} & \\ F(G(X) \otimes G(X)) & & \end{array}$$

Conversely, he shows that a linear category $\mathbb{X}$ gives rise to a (non-closed) linear/non-linear adjunction by considering the Eilenberg–Moore category $\mathbb{X}^!$, which is cartesian, and showing that the adjunction between $\mathbb{X}^!$ and $\mathbb{X}$ is monoidal.

2.7.6 Linearly Distributive Categories

A **linearly distributive category** [11] is monoidal category with two monoidal structures $(\otimes,\ \top)$ and $(\oplus,\ \bot)$, called "tensor" and "par" respectively. The two monoidal structures are linked by two linear distributors, $d^{\otimes}_{\oplus}$ and $d^{\oplus}_{\otimes}$.

$$d^{\otimes}_{\oplus} : X \otimes (Y \oplus Z) \to (X \otimes Y) \oplus Z$$

$$d^{\oplus}_{\otimes} : (Y \oplus Z) \otimes X \to Y \oplus (Z \otimes X)$$

A linearly distributive category is symmetric if both the tensor and par are symmetric (by $c_{\otimes}$ and $c_{\oplus}$). Note that in this case, the following two additional linear distributors are induced by $d^{\otimes}_{\oplus}$, $d^{\oplus}_{\otimes}$, $c_{\otimes}$, and $c_{\oplus}$.

$$d^{\otimes\prime}_{\oplus} : X \otimes (Y \oplus Z) \to Y \oplus (X \otimes Z)$$

$$d^{\oplus\prime}_{\otimes} : (Y \oplus Z) \otimes X \to (Y \otimes X) \oplus Z$$

The above data must satisfy certain coherence conditions:

1. **Units and distributions:**

$$\begin{array}{ccc} \top \otimes (X \oplus Y) & & \\ \downarrow{\scriptstyle d^{\otimes}_{\oplus}} & \searrow^{u^L_{\otimes}} & \\ (\top \otimes X) \oplus Y & \xrightarrow[u^L_{\otimes}\oplus 1]{} & X \oplus Y \end{array} \qquad \begin{array}{ccc} (\top \oplus X) \otimes Y & & \\ \downarrow{\scriptstyle d^{\oplus}_{\otimes}} & \searrow^{u^L_{\oplus}\otimes 1} & \\ \top \oplus (X \otimes Y) & \xrightarrow[u^L_{\oplus}]{} & X \otimes Y \end{array}$$

$$\begin{array}{ccc} (X \oplus Y) \otimes \top & & \\ \downarrow{\scriptstyle d^{\otimes}_{\oplus}} & \searrow^{u^R_{\otimes}} & \\ (X \otimes \top) \oplus Y & \xrightarrow[u^R_{\otimes}\oplus 1]{} & X \oplus Y \end{array} \qquad \begin{array}{ccc} Y \otimes (X \oplus \top) & & \\ \downarrow{\scriptstyle d^{\oplus}_{\otimes}} & \searrow^{1\otimes u^R_{\oplus}} & \\ Y \oplus (X \otimes \top) & \xrightarrow[u^R_{\oplus}]{} & Y \otimes X \end{array}$$

2. **Associativity and distributions:**

$$\begin{array}{ccc}
(X \otimes Y) \otimes (Z \oplus W) & \xrightarrow{a_\otimes} & X \otimes (Y \otimes (Z \oplus W)) \\
\Big\downarrow{\scriptstyle d^\otimes_\oplus} & & \Big\downarrow{\scriptstyle 1 \otimes d^\otimes_\oplus} \\
 & & X \otimes ((Y \otimes Z) \oplus W) \\
 & & \Big\downarrow{\scriptstyle d^\otimes_\oplus} \\
((X \otimes Y) \otimes Z) \oplus W & \xrightarrow[a_\otimes \oplus 1]{} & (X \otimes (Y \otimes Z)) \oplus W
\end{array}$$

3. **Distributions and distributions:**

$$\begin{array}{ccc}
(X \oplus Y) \otimes (Z \oplus W) & \xrightarrow{d^\oplus_\otimes} & X \oplus (Y \otimes (Z \oplus W)) \\
\Big\downarrow{\scriptstyle d^\otimes_\oplus} & & \\
((X \oplus Y) \otimes Z) \oplus W & & \Big\downarrow{\scriptstyle 1 \oplus d^\otimes_\oplus} \\
\Big\downarrow{\scriptstyle d^\oplus_\otimes \oplus 1} & & \\
(X \oplus (Y \otimes Z)) \oplus W & \xrightarrow[a_\oplus]{} & X \oplus ((Y \otimes Z) \oplus W)
\end{array}$$

4. **Coassociativity and distributions:**

$$\begin{array}{ccc}
X \otimes ((Y \oplus Z) \oplus W) & \xrightarrow{1 \otimes a_\oplus} & X \otimes (Y \oplus (Z \oplus W)) \\
\Big\downarrow{\scriptstyle d^\otimes_\oplus} & & \Big\downarrow{\scriptstyle d^{\otimes\prime}_\oplus} \\
(X \otimes (Y \oplus Z)) \oplus W & & Y \oplus (X \otimes (Z \oplus W)) \\
\Big\downarrow{\scriptstyle d^{\otimes\prime}_\oplus \oplus 1} & & \Big\downarrow{\scriptstyle 1 \oplus d^\otimes_\oplus} \\
(Y \oplus (X \otimes Z)) \oplus W & \xrightarrow[a_\oplus]{} & Y \oplus ((X \otimes Z) \oplus W)
\end{array}$$

An example of a linearly distributive category is distributive lattice where the objects are elements and maps are comparisons. In this case, $\otimes$ is the meet $\wedge$ and $\oplus$ is the join $\vee$.

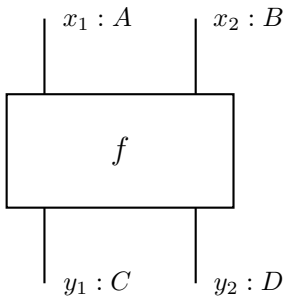


Figure 2.1: Typed Circuit

Circuit Diagrams for Linearly Distributive Categories

The use of string diagrams was introduced in [22] to represent morphisms in monoidal categories. This graphical approach was further extended in [7] to capture the relationship between proof theory and linearly distributive categories.

A circuit diagram in this context is built from a set of types and components. Each component is treated as a black box with a fixed number of typed input and output ports. Wires are attached to these ports to interconnect components, forming a circuit. The types of the wires must match the types of the ports they connect to.

As an example, Figure 2.1 indicates a component f with two input ports of types A and B, and two output ports of types C and D. Suppose wires x_1 and x_2 (of types A and B, respectively) are attached to the inputs, and wires y_1 and y_2 (of types C and D) are attached to the outputs. The circuit expression representing this component is written as:

$$[x_1, x_2]f[y_1, y_2].$$

Multiple circuit expressions can be composed by concatenating them. For instance:

$$[x_2, x_3]f[y_2, y_4][x_1, y_2]g[y_1, y_3]$$

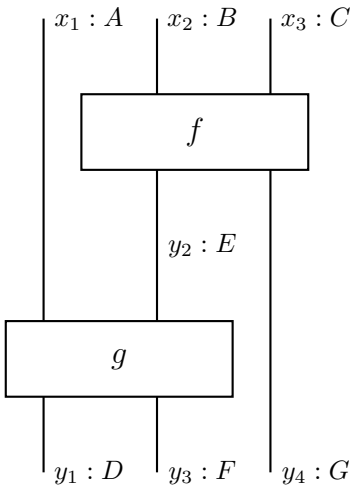


Figure 2.2: circuit diagram of composing f and g

represents the circuit shown in Figure 2.2.

Recall that linearly distributive categories have two tensor products: tersor ($\otimes$) and par ($\oplus$). Circuits in these settings are introduced in [1]. Using components from a set $\mathbf{S}$, with ports labeled by formulas built from atomic types in a set $\mathbf{P}$ as:

(i) Any atomic type $A \in \mathbf{P}$ is a formula,

(ii) If A and B are formulas, then $A \otimes B$ and $A \oplus B$ are also formulas,

(iii) The units $\top$ and $\bot$ are formulas.

Special components, referred to as *links*, are used in the circuits for linearly distributive categories to indicate the tensor and par structures [7]. These include:

$$
\begin{array}{rl}
[A, B] \otimes I[A \otimes B] & \otimes\text{-introduction} \\
[A \otimes B] \otimes E[A, B] & \otimes\text{-elimination} \\
[A, B] \oplus I[A \oplus B] & \oplus\text{-introduction} \\
[A \oplus B] \oplus E[A, B] & \oplus\text{-elimination} \\
[\,]\top I[\top] & \text{unit introduction} \\
[A, \top]\top E[A] & \text{unit elimination} \\
[\,]\bot E[\bot] & \text{counit introduction} \\
[A]\bot I[A, \bot] & \text{counit elimination}
\end{array}
$$

Figure 2.3 illustrates the graphical forms of these links, where the $\otimes E$ and $\oplus I$ links are "switchable" links in the sense of Girard [16]. From the proof-theoretic standpoint, $\otimes I$ and $\otimes E$ act as the introduction and elimination rules for tensor, whereas $\oplus I$ and $\oplus E$ serve as the introduction and elimination rules for par.

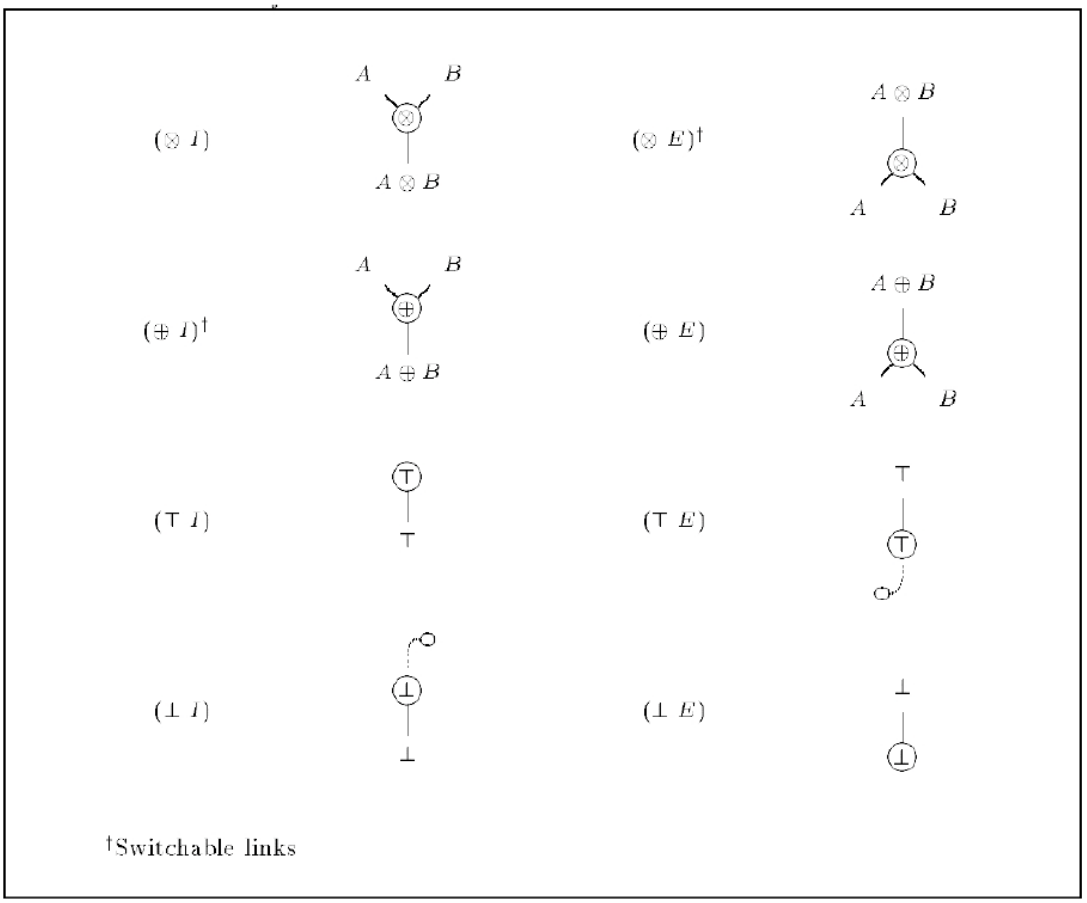


Figure 2.3: Introduction and elimination rules for $\otimes, \otimes, \top, \bot$ (adapted from [6])

To construct a valid circuit, one must adhere to the judgment rules of the type theory which determine the well-formed proofs. Girard [16] proposed a correctness criterion for circuits: a circuit is valid if, for every choice of "switch settings" for switchable links, the resulting graph is acyclic and connected. This ensures that once a switching is chosen, there exists a unique path between any two components.

However, checking Girard's criterion directly is computationally expensive. If a circuit has n switchable links, then 2^n different switchings must be tested. Danos and Regnier [13] introduced a more efficient method using *sequentialization*. A circuit involving $\otimes$ and $\oplus$ is sequential if it can be reduced to a single sequent box via a sequence of reduction

steps [7]. This reduction demonstrates the validity of the proof. Figure 2.4 illustrates the sequentialization procedure. These rules allow us to group non-switching components into boxes, apply cut rules, and eliminate switching links. According to [7], a sequential circuit (also called a proof net) corresponds to a morphism in a linearly distributive category.

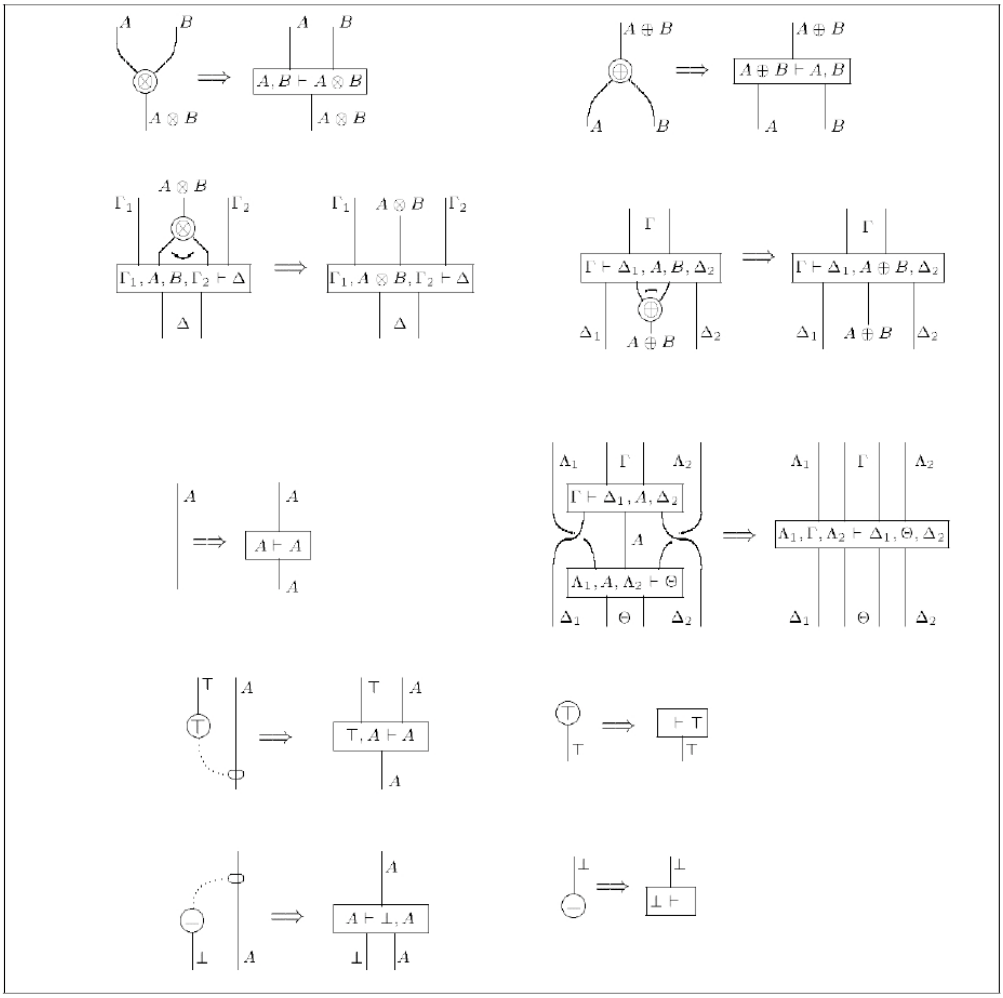


Figure 2.4: Sequentialization Procedure (adapted from [6])

The equivalence of proof nets is governed by a rewriting system consisting of three rule types: *reductions*, *expansions*, and *equivalences*, as shown in Figure 2.5. Reduction rules simplify circuits, expansion rules unpack wires, and equivalence rules adjust unit

links. These transformations are only valid if they preserve the legality of the circuit.

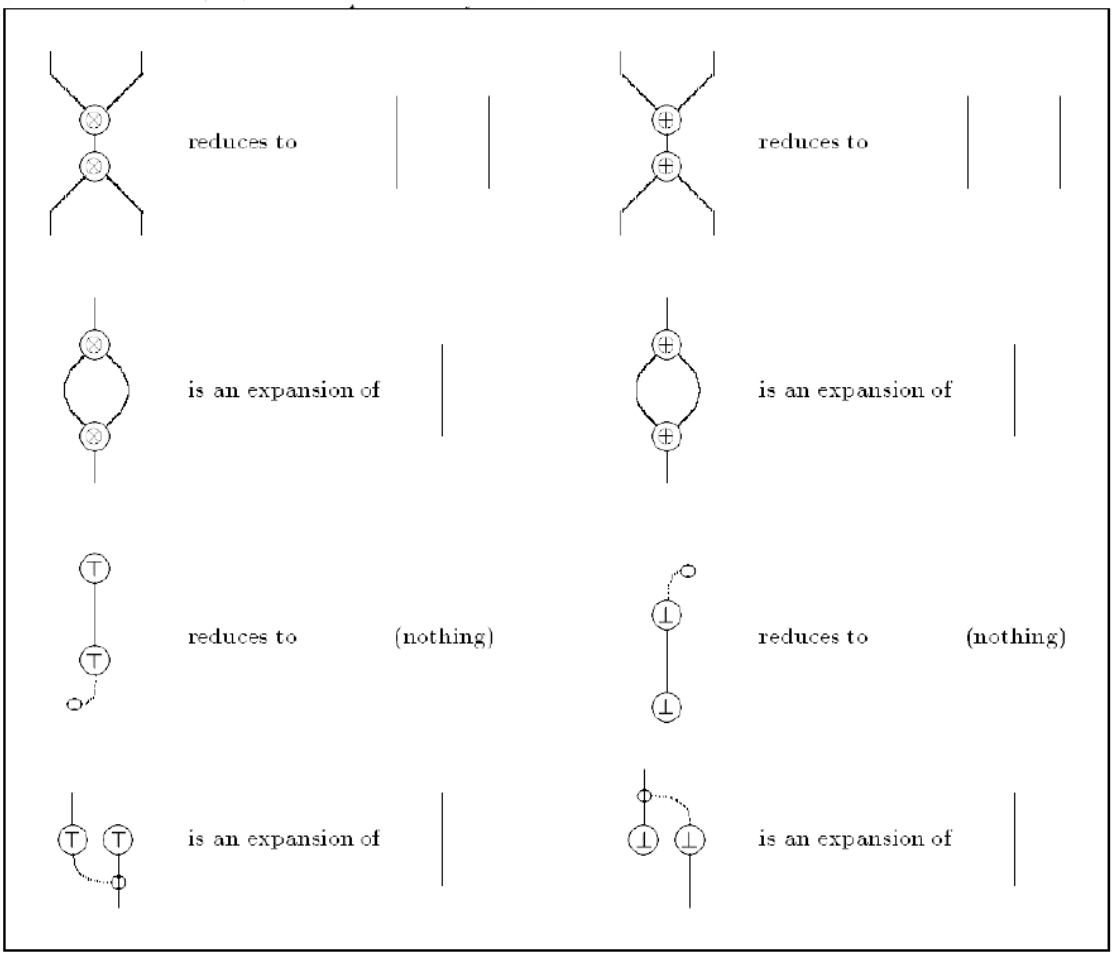


Figure 2.5: Reduction, expansion, and equivalence rules for $\otimes$, $\oplus$, $\top$, $\bot$ (adapted from [6])

2.8 Enrichment

A category $\mathbb{X}$ is enriched [24] in the monoidal category $(\mathbb{A}, \otimes, I)$ if:

1. For every pair of objects $X, Y \in \mathbb{X}$, there is a hom-object $\mathrm{Hom}(X, Y) \in \mathbb{A}$

2. For every triple $X, Y, Z \in \mathbb{X}$, a "composition" morphism exists in $\mathbb{A}$:

$$m_{XYZ} : \mathrm{Hom}(X, Y) \otimes \mathrm{Hom}(Y, Z) \to \mathrm{Hom}(X, Z)$$

3. For every object $X \in \mathbb{X}$ there is a "unit" morphism in $\mathbb{A}$:

$$id : I \to \mathrm{Hom}(X, X)$$

Such that the following diagrams commute:

(a) Associativity of the composition:

$$\begin{array}{ccc} \mathrm{Hom}(X,Y) \otimes \mathrm{Hom}(Y,Z) \otimes \mathrm{Hom}(Z,W) & \xrightarrow{m_{XYZ} \otimes 1_{\mathrm{Hom}(Z,W)}} & \mathrm{Hom}(X,Z) \otimes \mathrm{Hom}(Z,W) \\ \downarrow{\scriptstyle 1_{\mathrm{Hom}(X,Y)} \otimes m_{YZW}} & & \downarrow{\scriptstyle m_{XZW}} \\ \mathrm{Hom}(X,Y) \otimes \mathrm{Hom}(Y,W) & \xrightarrow[m_{XYW}]{} & \mathrm{Hom}(X,W) \end{array}$$

(b) Identity on the left:

$$\begin{array}{ccc} I \otimes \mathrm{Hom}(X,Y) & \xrightarrow{id_X \otimes 1_{\mathrm{Hom}(X,Y)}} & \mathrm{Hom}(X,X) \otimes \mathrm{Hom}(X,Y) \\ & \searrow^{u^L_\otimes} & \downarrow{\scriptstyle m_{XXY}} \\ & & \mathrm{Hom}(X,Y) \end{array}$$

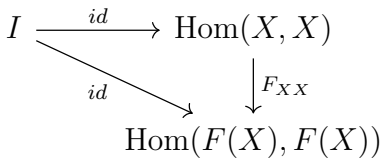
(c) Identity on the right:

$$\begin{array}{ccc} \mathrm{Hom}(X,Y) \otimes I & \xrightarrow{1_{\mathrm{Hom}(X,Y)} \otimes id_Y} & \mathrm{Hom}(X,Y) \otimes \mathrm{Hom}(Y,Y) \\ & \searrow^{u^R_\otimes} & \downarrow{\scriptstyle m_{XYY}} \\ & & \mathrm{Hom}(X,Y) \end{array}$$

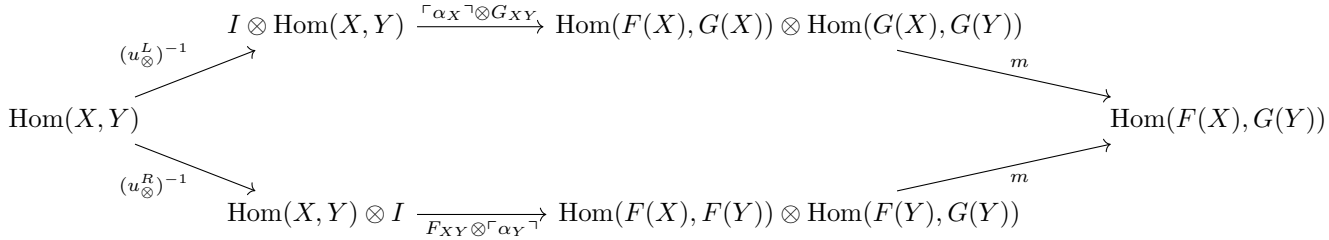
2.8.1 Enriched Functors and Natural Transformations

For $\mathbb{A}$-enriched categories $\mathbb{X}$ and $\mathbb{Y}$, an **$\mathbb{A}$-functor** F [24] consists of a map $F : obj(\mathbb{X}) \to obj(\mathbb{Y})$, and for each $X, Y \in \mathbb{X}$ a map $F_{XY} : \mathrm{Hom}(X,Y) \to \mathrm{Hom}(F(X), F(Y))$ in $\mathbb{A}$, such that the following diagrams commute:

$$\begin{array}{ccc} \mathrm{Hom}(X,Y) \otimes \mathrm{Hom}(Y,Z) & \xrightarrow{m} & \mathrm{Hom}(X,Z) \\ \downarrow{\scriptstyle F_{XY} \otimes F_{YZ}} & & \downarrow{\scriptstyle F_{XZ}} \\ \mathrm{Hom}(F(X),F(Y)) \otimes \mathrm{Hom}(F(Y),F(Z)) & \xrightarrow[m]{} & \mathrm{Hom}(F(X),F(Z)) \end{array}$$



For $\mathbb{A}$-functors F, G: $\mathbb{X} \to \mathbb{Y}$, an $\mathbb{A}$-natural transformation $\alpha : F \Rightarrow G$ [24] is a family of components $\ulcorner \alpha_X \urcorner : I \to \mathrm{Hom}(F(X), G(X))$ in $\mathbb{A}$, satisfying:



2.8.2 The Underlying Ordinary Category

The **underlying ordinary category** [24] of the $\mathbb{A}$-enriched category $\mathbb{X}$ is denoted by $\ulcorner \mathbb{X} \urcorner$ and has the same objects as $\mathbb{X}$ while a map $f : X \to Y$ in $\ulcorner \mathbb{X} \urcorner$ is just an element $\ulcorner f \urcorner : I \to \mathrm{Hom}(X, Y)$. The composition of two maps $f : X \to Y$ and $g : Y \to Z$ in $\ulcorner \mathbb{X} \urcorner$ is given by:

$$I \xrightarrow{(u^R_\otimes)^{-1}} I \otimes I \xrightarrow{\ulcorner f \urcorner \otimes \ulcorner g \urcorner} \mathrm{Hom}(X,Y) \otimes \mathrm{Hom}(Y,Z) \xrightarrow{m} \mathrm{Hom}(X,Z)$$

and the identity $1_A : X \to X$ is $id : I \to \mathrm{Hom}(X, X)$.

2.8.3 Behavior of the Hom Functor

Let $\mathbb{X}$ be an $\mathbb{A}$-enriched category, the **hom functor** is a functor of the form:

$$\mathrm{Hom} : \ulcorner \mathbb{X} \urcorner^{op} \times \ulcorner \mathbb{X} \urcorner \to \mathbb{A}$$

that sends objects $X, Y \in \mathbb{X}$ to the hom-object $\mathrm{Hom}(X, Y) \in \mathbb{A}$. Fixing an object $X \in \mathbb{X}$, the hom functor induces a functor

$$\mathrm{Hom}(X, -) : \ulcorner \mathbb{X} \urcorner \to \mathbb{A},$$

which we will use extensively throughout this thesis. This functor acts on objects by sending $Y \in \mathbb{X}$ to $\mathrm{Hom}(X, Y)$, and on morphisms as follows.

Given a morphism $f : Y \to Z$ in $\mathbb{X}$, the induced morphism $\mathrm{Hom}(1_X, f) : \mathrm{Hom}(X, Y) \to \mathrm{Hom}(X, Z)$ is defined via the enriched composition as:

$$\mathrm{Hom}(1_X, f) = (u^R_\otimes)^{-1}(1 \otimes \ulcorner f \urcorner)m$$

This action can be illustrated by the following commutative diagram:

$$\begin{array}{ccccc} \mathrm{Hom}(X,Y) & \xrightarrow{(u^R_\otimes)^{-1}} & \mathrm{Hom}(X,Y) \otimes I & \xrightarrow{1 \otimes \ulcorner f \urcorner} & \mathrm{Hom}(X,Y) \otimes \mathrm{Hom}(Y,Z) \\ & \searrow^{\mathrm{Hom}(1,f)} & & & \downarrow m \\ & & & & \mathrm{Hom}(X,Z) \end{array}$$

This diagram reflects how morphisms in the enriched category are composed internally in $\mathbb{A}$, with identities and composition behaving coherently with the monoidal structure. In particular, this construction ensures that the enriched category $\mathbb{X}$ behaves analogously to an ordinary category, but with morphisms generalized to objects in the base monoidal category $\mathbb{A}$.

The following lemma about the behavior of the hom functor and composition morphism is used frequently in proofs throughout this thesis.

Lemma 2.8.1. *In any $\mathbb{A}$-enriched category $\mathbb{X}$, for all $f : Z \to Z'$ we have* $(1 \otimes \mathrm{Hom}(1, f))m = m\mathrm{Hom}(1, f)$.

$$\begin{array}{ccc} \mathrm{Hom}(X,Y)\otimes\mathrm{Hom}(Y,Z) & \xrightarrow{1\otimes\mathrm{Hom}(1,f)} & \mathrm{Hom}(X,Y)\otimes\mathrm{Hom}(Y,Z') \\ \downarrow{\scriptstyle m} & & \downarrow{\scriptstyle m} \\ \mathrm{Hom}(X,Z) & \xrightarrow[\mathrm{Hom}(1,f)]{} & \mathrm{Hom}(X,Z') \end{array}$$

Proof.

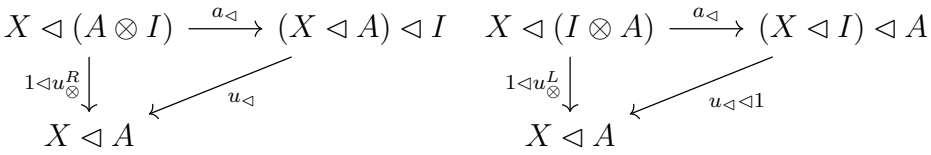
$$\begin{aligned} m\mathrm{Hom}(1,f) &= m(u^R_\otimes)^{-1}(1\otimes\ulcorner f\urcorner)m \\ &= (u^R_\otimes)^{-1}(m\otimes 1)(1\otimes\ulcorner f\urcorner)m \\ &= (u^R_\otimes)^{-1}(m\otimes\ulcorner f\urcorner)m \\ &= (u^R_\otimes)^{-1}((1\otimes 1)m\otimes\ulcorner f\urcorner)m \\ &= (u^R_\otimes)^{-1}a_\otimes(1\otimes(1\otimes\ulcorner f\urcorner)m \\ &= (1\otimes(u^R_\otimes)^{-1}(1\otimes\ulcorner f\urcorner)m)m \\ &= (1\otimes\mathrm{Hom}(1,f))m \end{aligned}$$

□

2.9 Actegories

A **right $\mathbb{A}$-actegory** consists of a monoidal category $(\mathbb{A},\otimes,I)$, a category $\mathbb{X}$, an action functor $\lhd:\mathbb{X}\times\mathbb{A}\to\mathbb{X}$ and two natural isomorphism $a_\lhd : X\lhd(A\otimes B)\to(X\lhd A)\lhd B$ and $u_\lhd : X\lhd I\to X$ satisfying the following coherences [9]:

$$\begin{array}{ccc} X\lhd(A\otimes(B\otimes C)) & \xrightarrow{1\lhd a_\otimes} & X\lhd((A\otimes B)\otimes C) \\ \downarrow{\scriptstyle a_\lhd} & & \downarrow{\scriptstyle a_\lhd} \\ (X\lhd A)\lhd(B\otimes C) & & (X\lhd(A\otimes B))\lhd C \\ \downarrow{\scriptstyle a_\lhd} & \swarrow{\scriptstyle a_\lhd\lhd 1} & \\ ((X\lhd A)\lhd B)\lhd C & & \end{array}$$



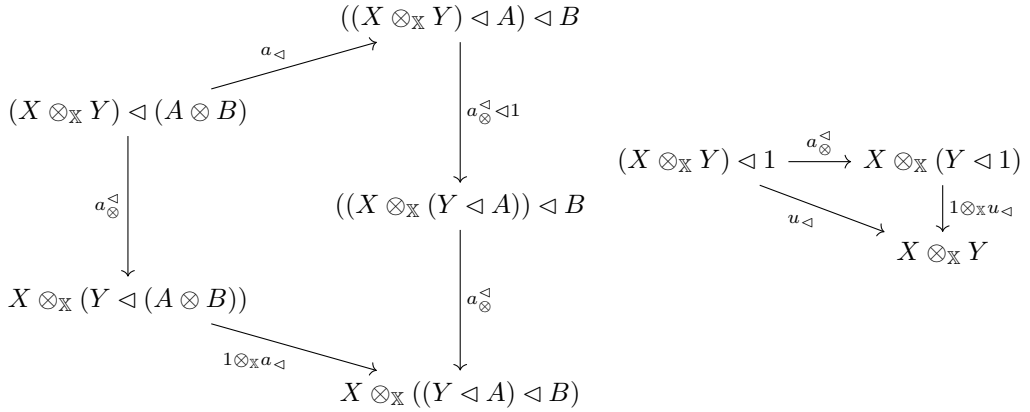
An example of an actegory, is **Set**, with an action on itself given by the cartesian product. More generally any monoidal category acts on itself by the tensor product.

Similar to right $\mathbb{A}$-actegories, left $\mathbb{A}$-actegories are equipped with an action functor $\rhd : \mathbb{A} \times \mathbb{X} \to \mathbb{X}$ along with two natural isomorphisms $a_{\rhd} : (A \otimes B) \rhd X \to A \rhd (B \rhd X)$ and $u_{\rhd} : I \rhd X \to X$ satisfying similar coherences.

A right $\mathbb{A}$-actegory is precisely a **left** $\mathbb{A}^{rev}$-actegory, where $\mathbb{A}^{rev}$ is the monoidal category obtained by equipping the underlying category of $\mathbb{A}$ with the tensor product $A \otimes^{rev} B := B \otimes A$ and appropriately adjusting the rest of the structure.

2.9.1 Monoidal Actegories

A **monoidal right $\mathbb{A}$-actegory** is a monoidal category $(\mathbb{X}, \otimes_{\mathbb{X}}, I_{\mathbb{X}})$ with a functor $\lhd : \mathbb{X} \times \mathbb{A} \to \mathbb{X}$ and two monoidal natural transformations $a_{\lhd}$ and $u_{\lhd}$ defined as in Definition 2.9, along with a natural map $a^{\lhd}_{\otimes} : (X \otimes_{\mathbb{X}} Y) \lhd A \to X \otimes_{\mathbb{X}} (Y \lhd A)$ such that the following diagrams commute



An example of a monoidal actegory is a monoidal category, which acts on itself, and the map $a^{\lhd}_{\otimes}$ in this case is the associator $a_{\otimes} : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$.

In a right monoidal actegory $\mathbb{X}$, if $\mathbb{X}$ is symmetric monoidal, then the action functor is monoidal. When the monoidal category $\mathbb{X}$ is symmetric, then there is an isomorphism $s : X \otimes_{\mathbb{X}} Y \simeq Y \otimes_{\mathbb{X}} X$. Using this map, we can construct $a^{\triangleleft\prime}_{\otimes} : (X \otimes_{\mathbb{X}} Y) \triangleleft A \to (X \triangleleft A) \otimes_{\mathbb{X}} Y$ as $a^{\triangleleft\prime}_{\otimes} = (s \triangleleft 1) a^{\triangleleft}_{\otimes} s$. We can define the monoidal structure for $\triangleleft$ by defining $m^{\triangleleft}_I : I_{\mathbb{X}} \to I_{\mathbb{X}} \triangleleft I_{\mathbb{A}}$ as $m^{\triangleleft}_I := u^{-1}_{\triangleleft}$ and $m^{\triangleleft}_{\otimes} : (X \otimes_{\mathbb{X}} Y) \triangleleft (A \otimes_{\mathbb{A}} B) \to (X \triangleleft A) \otimes_{\mathbb{X}} (Y \triangleleft B)$ by the following composite.

$$\begin{array}{ccc} (X \otimes_{\mathbb{X}} Y) \triangleleft (A \otimes B) & \xrightarrow{a_{\triangleleft}} & ((X \otimes_{\mathbb{X}} Y) \triangleleft A) \triangleleft B \\ & \searrow^{m^{\triangleleft}_{\otimes}} & \downarrow{\scriptstyle a^{\triangleleft\prime}_{\otimes} \triangleleft 1} \\ & & ((X \triangleleft A) \otimes_{\mathbb{X}} Y) \triangleleft B \\ & & \downarrow{\scriptstyle a^{\triangleleft}_{\otimes}} \\ & & (X \triangleleft A) \otimes_{\mathbb{X}} (Y \triangleleft B) \end{array}$$

This is how a monoidal actegory is defined in [9].

An actegory is called **strong monoidal** when $a^{\triangleleft}_{\otimes}$ is an isomorphism, since in this case, the action functor is strong monoidal.

2.9.2 Linear Actegories

A **(symmetric) linear actegory** [19] is a monoidal actegory in which a monoidal category $\mathbb{A}$ acts on a linearly distributive category $\mathbb{X}$ both covarient and controvariantly. A linear actegory consists of the following data:

- a symmetric monoidal category $(\mathbb{A}, *, I, a_*, u^L_*, u^R_*)$
- a linearly distributive category $\mathbb{X}$ as in 2.7.6
- two action functors:

$$\circ : \mathbb{A} \times \mathbb{X} \to \mathbb{X} \quad \text{and} \quad \bullet : \mathbb{A}^{op} \times \mathbb{X} \to \mathbb{X}$$

such that $\circ$ is paramterized left adjoint of $\bullet$, i.e. $\forall A \in \mathbb{A}$, $A \circ - \dashv A \bullet - : \mathbb{X} \to \mathbb{X}$

- The following natural isomorphisms in $\mathbb{X}$ for all $A, B \in \mathbb{A}$ and $X, Y \in \mathbb{X}$:

$$u_\circ : I \circ X \to X, \quad u_\bullet : X \to I \bullet X,$$

$$a^*_\circ : (A * B) \circ X \to A \circ (B \circ X), \quad a^*_\bullet : A \bullet (B \bullet X) \to (A * B) \bullet X,$$

$$a^\circ_\otimes : A \circ (X \otimes Y) \to (A \circ X) \otimes Y, \quad a^\bullet_\oplus : (A \bullet X) \oplus Y \to A \bullet (X \oplus Y),$$

- The following natural morphisms in $\mathbb{X}$ for all $A, B \in \mathbb{A}$ and $X, Y \in \mathbb{X}$:

$$d^\circ_\oplus : A \circ (X \oplus Y) \to (A \circ X) \oplus Y, \quad d^\bullet_\otimes : (A \bullet X) \otimes Y \to A \bullet (X \otimes Y),$$

$$d^\circ_\bullet : A \circ (B \bullet X) \to B \bullet (A \circ X).$$

- The following natural isomorphisms from the symmetries of $*$, $\otimes$, and $\oplus$:

$$a^{*\prime}_\circ : (A * B) \circ X \to B \circ (A \circ X), \quad a^{*\prime}_\bullet : B \bullet (A \bullet X) \to (A * B) \bullet X,$$

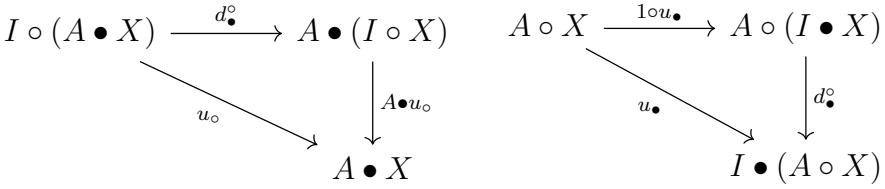
$$a^\circ_{\otimes'} : A \circ (X \otimes Y) \to X \otimes (A \circ Y), \quad a^\bullet_{\oplus'} : X \oplus (A \bullet Y) \to A \bullet (X \oplus Y),$$

$$d^\circ_{\oplus'} : A \circ (X \oplus Y) \to X \oplus (A \circ Y), \quad d^\bullet_{\otimes'} : X \otimes (A \bullet Y) \to A \bullet (X \otimes Y).$$

This data must satisfy several coherence conditions besides the ones that are required by being a strong monoidal actegory. These coherences are expressed below.

- Unit and distributivity:

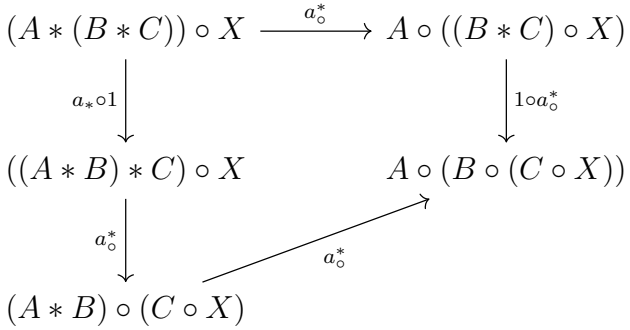
$$\begin{array}{ccc} I \circ (X \oplus Y) & \xrightarrow{d^\circ_\oplus} & (I \circ X) \oplus Y \\ & \searrow^{u_\circ} & \downarrow{\scriptstyle u_\circ \oplus 1} \\ & & X \oplus Y \end{array} \qquad \begin{array}{ccc} I \circ (X \otimes Y) & \xrightarrow{a^\circ_\otimes} & (I \circ X) \otimes Y \\ & \searrow^{u_\circ} & \downarrow{\scriptstyle u_\circ \otimes 1} \\ & & X \otimes Y \end{array}$$

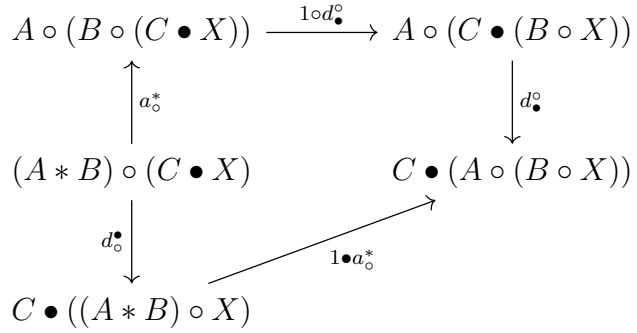


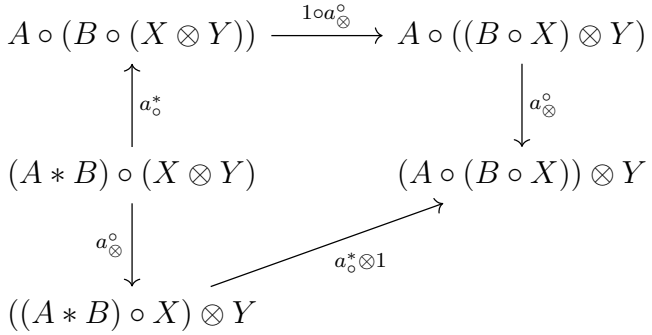
which result in the following equalities:

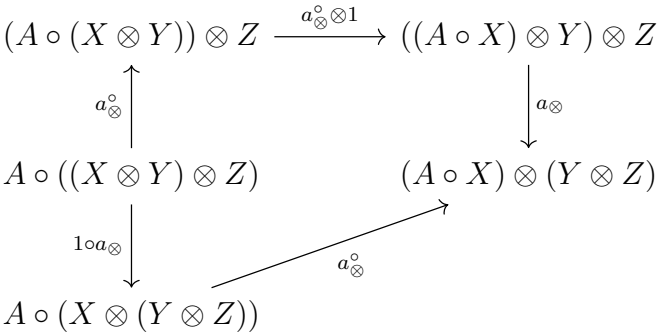
$$d^{\circ}_{\oplus}(u_{\circ} \oplus 1) = u_{\circ} \quad (u_{\bullet} \otimes Y)d^{\bullet}_{\otimes} = u_{\bullet}$$
$$d^{\circ'}_{\oplus}(1 \oplus u_{\circ}) = u_{\circ} \quad (1 \otimes u_{\bullet})d^{\bullet'}_{\otimes} = u_{\bullet}$$
$$a^{\circ}_{\otimes}(u_{\circ} \otimes 1) = u_{\circ} \quad (u_{\bullet} \oplus 1)a^{\bullet}_{\oplus} = u_{\bullet}$$
$$a^{\circ'}_{\otimes}(1 \otimes u_{\circ}) = u_{\circ} \quad (1 \oplus u_{\bullet})a^{\bullet'}_{\oplus} = u_{\bullet}$$
$$d^{\circ}_{\circ}(1 \bullet u_{\circ}) = u_{\circ} \quad (1 \circ u_{\bullet})d^{\circ}_{\bullet} = u_{\bullet}$$
$$d^{\circ'}_{\circ}(1 \bullet u_{\circ}) = u_{\circ} \quad (1 \circ u_{\bullet})d^{\circ'}_{\bullet} = u_{\bullet}$$
$$(1 \circ u_{\bullet})d^{\circ}_{\bullet} = u_{\bullet} \quad d^{\circ}_{\bullet}(1 \bullet u_{\circ}) = u_{\circ}$$

- Associativity:

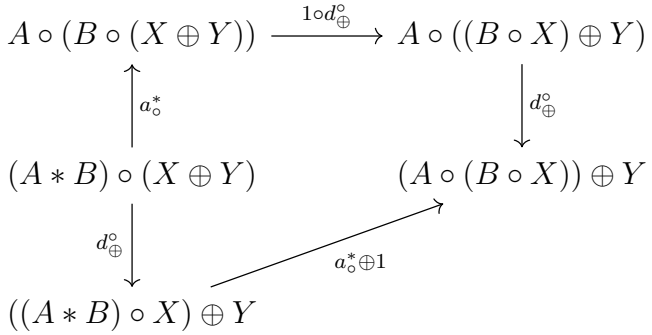


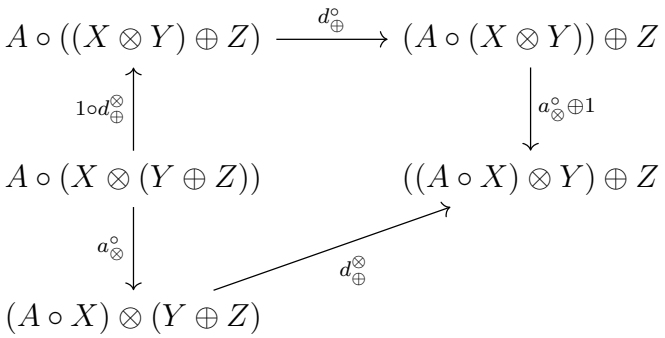


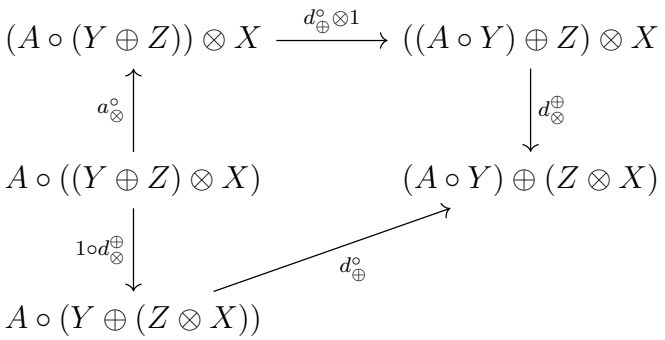


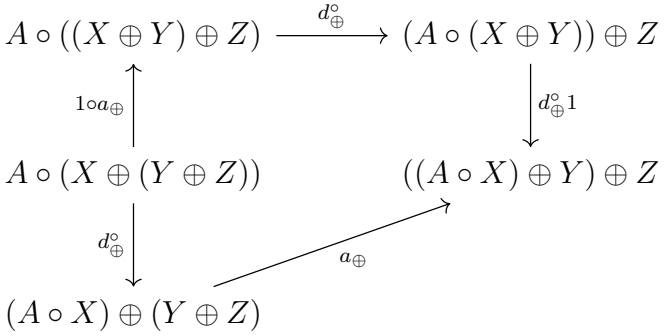


- Distributivity and associativity:

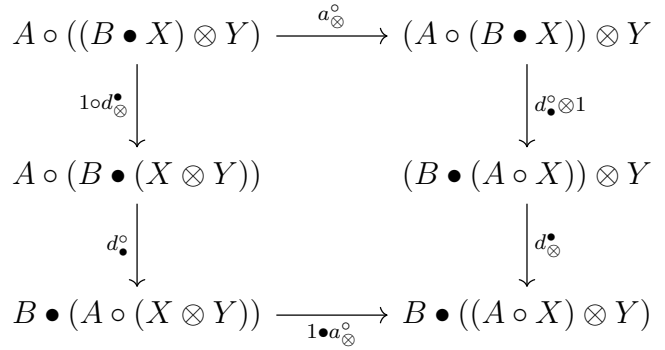


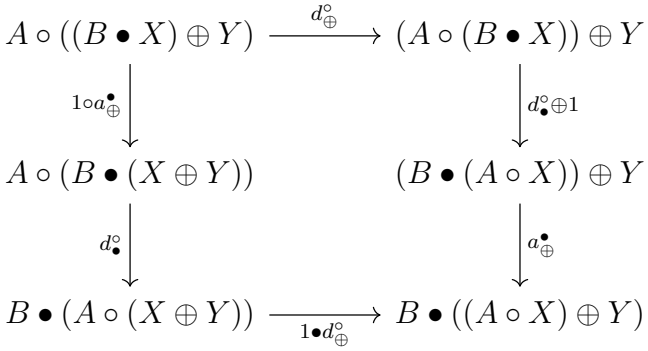






- And the remaining two are





Any monoidal closed category (regarded as a compact linearly distributive category) acts on itself via the tensor product $\otimes$ and the internal hom $\multimap$. However, these actions do not in general give a linear actegory, as the required isomorphism $a^{\bullet}_{\oplus} : (A \bullet X) \otimes Y \to A \bullet (X \oplus Y)$ may not exist. Although, if we restrict the action to the compact objects (that is, those objects for which the map $(A \multimap I) \otimes B \to A \multimap B$ is an isomorphism), then the structure becomes a linear actegory with the same actions.

Chapter 3

Actegories and Copowers

3.1 Introduction

A right $\mathbb{A}$-actegory $\mathbb{X}$ consists of a monoidal category $\mathbb{A}$ acting on a category $\mathbb{X}$ in a manner compatible with the monoidal structure. This may be formulated as a strong monoidal functor $F : \mathbb{A} \to End(\mathbb{X})$ where $End(\mathbb{X})$ is the category of endofunctors of $\mathbb{X}$ with the monoidal structure given by composition. Alternatively, an $\mathbb{A}$-actegory may be viewed more directly with an action functor $\lhd : \mathbb{X} \times \mathbb{A} \to \mathbb{X}$ as in Definition 2.9 above. In an $\mathbb{A}$-actegory, if the action functor $\lhd : \mathbb{X} \times \mathbb{A} \to \mathbb{X}$ admits a right adjoint written as $X \lhd - \dashv \mathrm{Hom}(X, -) : \mathbb{A} \to \mathbb{X}$, then $\mathbb{X}$ is referred to as a right $\mathbb{A}$-actegory with hom-objects.

A category $\mathbb{X}$ is right $\mathbb{A}$-enriched, if for every pair of objects $X, Y \in \mathbb{X}$, an object $\mathrm{Hom}(X, Y)$ exists in $\mathbb{A}$, and there are composition $m : \mathrm{Hom}(X, Y) \otimes \mathrm{Hom}(Y, Z) \to \mathrm{Hom}(Z, W)$ and unit $id : I \to \mathrm{Hom}(X, X)$ morphisms satisfying coherenece s specified in Section 2.8. A right $\mathbb{A}$-enriched category $\mathbb{X}$ is said to be copowered [28], if for any $A \in \mathbb{A}$ and $X \in \mathbb{X}$, there exists an object $X \lhd A \in \mathbb{X}$ and a morphism $\eta : A \to \mathrm{Hom}(X, X \lhd A)$ in $\mathbb{A}$ referred to as the *unit* of the copower, such that for each $B \in \mathbb{A}$, $Y \in \mathbb{X}$ and map

$f : B \to \mathrm{Hom}(X \lhd A, Y) \in \mathbb{A}$, there exists a bijective correspondence

$$\frac{B \xrightarrow{f} \mathrm{Hom}(X \lhd A, Y)}{A \otimes B \xrightarrow{(\eta \otimes f)m} \mathrm{Hom}(X, Y)}$$

where m is the composition morphism in $\mathbb{A}$.

The primary goal of this work is to establish an equivalence between right $\mathbb{A}$-actegories with hom-objects and right $\mathbb{A}$-enriched categories with copowers. Specifically, we prove the following key theorem:

Theorem 3.1.1. *To give a right actegory with hom-objects is to give a right enriched category with copowers.*

The proof proceeds as follows:

1. We show that given a right $\mathbb{A}$-actegory with hom-objects, one can construct a right enrichment over $\mathbb{A}$. We then show that copowers are given by the action.

2. Conversely, we demonstrate that a right $\mathbb{A}$-enriched category with copowers induces a right $\mathbb{A}$-actegory structure with right hom-objects.

3. One structure determines the other structure. The copower is the action and the Hom functor is the Hom of enrichment, and thus going back and forth is the identity.

Then, we establish the dual theorem by carefully changing right actegory to left actegory, right enrichment to left enrichment, transferring to $\mathbb{A}^{rev}$, and going to the opposite categories which will give us an equivalence between having a left $(\mathbb{A}^{rev})^{op}$-actegory $\mathbb{X}^{op}$ with hom-objects and a left $(\mathbb{A}^{rev})^{op}$-category with powers.

In the end we prove that in a right $\mathbb{A}$-actegory $\mathbb{X}$ with hom-objects, if $\forall A \in \mathbb{A}$, the action functor $- \lhd A$ has a right adjoint as $- \lhd A \dashv A \blacktriangleright - : \ulcorner \mathbb{X} \urcorner \to \ulcorner \mathbb{X} \urcorner$, then $\mathbb{X}$ is a left $(\mathbb{A}^{rev})^{op}$-actegory with hom-objects.

3.2 Copowers

Generally, copowers are defined for a right monoidal-closed category $\mathbb{A}$ and an $\mathbb{A}$-enriched category $\mathbb{X}$ by an $\mathbb{A}$-natural isomorphism:

$$\gamma : \mathrm{Hom}(X \lhd A, Y) \to A \multimap \mathrm{Hom}(X, Y)$$

However in this thesis we will give a definition that works when $\mathbb{A}$ is not right monoidal-closed.

Definition 3.2.1. An $\mathbb{A}$-enriched category $\mathbb{X}$ has **copowers** [28] if, for any $A \in \mathbb{A}$ and $X \in \mathbb{X}$, there exists an object $X \lhd A \in \mathbb{X}$ and a morphism $\eta : A \to \mathrm{Hom}(X, X \lhd A)$ in $\mathbb{A}$ (called the unit of the copower) such that for any $B \in \mathbb{A}$ and $Y \in \mathbb{X}$ there is a bijective correspondence:

$$\frac{B \xrightarrow{f} \mathrm{Hom}(X \lhd A, Y)}{A \otimes B \xrightarrow{(\eta \otimes f)m} \mathrm{Hom}(X, Y)}$$

Later in this section, we show that this definition is equivalent to having an $\mathbb{A}$-natural isomorphism $\gamma : \mathrm{Hom}(A \lhd X, Y) \to A \multimap \mathrm{Hom}(X, Y)$ when $\mathbb{A}$ is right monoidal-closed. We first prove the following lemma.

Lemma 3.2.1. *Given a bijective correspondence*

$$\frac{B \xrightarrow{f=(g)^{\Uparrow}} Hom(X \lhd A, Y)}{A \otimes B \xrightarrow{g=(f)^{\Downarrow}} Hom(X, Y)}$$

with the two combinators $(-)^{\Downarrow}$ and $(-)^{\Uparrow}$, the followings are equivalent:

(i) $\exists\ \eta : A \to Hom(X, X \lhd A)$ *and* $(f)^{\Downarrow} = (\eta \otimes f)m$

(ii) The Following properties hold for $(-)^{\Downarrow}$ and $(-)^{\Uparrow}$:

(a) $(1 \otimes h)(f)^{\Downarrow} = (hf)^{\Downarrow}$

(b) $h(f)^{\Uparrow} = ((1 \otimes h)f)^{\Uparrow}$

(c) $((f)^{\Downarrow} \otimes g)m = a_{\otimes}((f \otimes g)m)^{\Downarrow}$

(d) $((f)^{\Uparrow} \otimes g)m = (a_{\otimes}(f \otimes g)m)^{\Uparrow}$

Proof. We start with proving $(ii) \Rightarrow (i)$. We can define $\eta := (u^R_{\otimes})^{-1}(id)^{\Downarrow}$ using the given bijection as below:

$$\dfrac{\dfrac{I \xrightarrow{id} \mathrm{Hom}(X \lhd A, X \lhd A)}{A \otimes I \xrightarrow{(id)^{\Downarrow}} \mathrm{Hom}(X, X \lhd A)}}{A \xrightarrow{(u^R_{\otimes})^{-1}} A \otimes I \xrightarrow{(id)^{\Downarrow}} \mathrm{Hom}(X, X \lhd A)}$$

We have:

$$\begin{aligned}
(\eta \otimes f)m &= (((u^R_{\otimes})^{-1}(id)^{\Downarrow}) \otimes f)m \\
&= ((u^R_{\otimes})^{-1} \otimes 1)(1 \otimes f)((id)^{\Downarrow} \otimes 1)m \\
&= ((u^R_{\otimes})^{-1} \otimes 1)(1 \otimes f)a_{\otimes}((id \otimes 1)m)^{\Downarrow} \\
&= ((u^R_{\otimes})^{-1} \otimes 1)a_{\otimes}(1 \otimes f)((id \otimes 1)m)^{\Downarrow} \\
&= ((u^R_{\otimes})^{-1} \otimes 1)a_{\otimes}(1 \otimes f)((u^L_{\otimes})^{-1})^{\Downarrow} \\
&= ((u^R_{\otimes})^{-1} \otimes 1)a_{\otimes}(f(u^L_{\otimes})^{-1})^{\Downarrow} \\
&= (1 \otimes (u^L_{\otimes})^{-1})(f(u^L_{\otimes})^{-1})^{\Downarrow} \\
&= ((u^L_{\otimes})^{-1}f(u^L_{\otimes})^{-1})^{\Downarrow} \\
&= (f(u^L_{\otimes})^{-1}u^L_{\otimes})^{\Downarrow} \\
&= (f)^{\Downarrow}
\end{aligned}$$

Next, we prove $(i) \Rightarrow (ii)$. We have:

(a)

$$\begin{aligned}(1 \otimes h)(f)^{\Downarrow} &= (1 \otimes h)(\eta \otimes f)m \\ &= (\eta \otimes hf)m \\ &= (hf)^{\Downarrow}\end{aligned}$$

(b)

$$\begin{aligned}h(g)^{\Uparrow} &= ((h(g)^{\Uparrow})^{\Downarrow})^{\Uparrow} \\ &= ((\eta \otimes h(g)^{\Uparrow})m)^{\Uparrow} \\ &= ((1 \otimes h)(\eta \otimes (g)^{\Uparrow})m)^{\Uparrow} \\ &= ((1 \otimes h)((g)^{\Uparrow})^{\Downarrow})^{\Uparrow} \\ &= ((1 \otimes h)g)^{\Uparrow}\end{aligned}$$

(c)

$$\begin{aligned}((f)^{\Downarrow} \otimes g)m &= ((\eta \otimes f)m \otimes g)m \\ &= a_{\otimes}(\eta \otimes (f \otimes g)m)m \\ &= a_{\otimes}((f \otimes g)m)^{\Downarrow}\end{aligned}$$

(d)

$$\begin{aligned}((f)^{\Uparrow} \otimes g)m &= ((((f)^{\Uparrow} \otimes g)m)^{\Downarrow})^{\Uparrow} \\ &= ((\eta \otimes ((f)^{\Uparrow} \otimes g)m)m)^{\Uparrow} \\ &= (a_{\otimes}((\eta \otimes (f)^{\Uparrow})m \otimes g)m)^{\Uparrow} \\ &= (a_{\otimes}(((f)^{\Uparrow})^{\Downarrow} \otimes g)m)^{\Uparrow} \\ &= (a_{\otimes}(f \otimes g)m)^{\Uparrow}\end{aligned}$$

□

Corollary 3.2.2. *Given the bijective correspondence discussed in Lemma 3.2.1, the following properties hold for* $(-)^{\Uparrow}$ *and* $(-)^{\Downarrow}$*:*

(i) $(1 \otimes h)(f)^{\Downarrow} Hom(1,k) = (hf\, Hom(1,k))^{\Downarrow}$

(ii) $h(g)^{\Uparrow} Hom(1,k) = ((1 \otimes h)g Hom(1,k))^{\Uparrow}$

The preceding lemma plays a crucial role, as it provides an alternative characterization of copowers using the combinators $(-)^{\Uparrow}$ and $(-)^{\Downarrow}$. This reformulation is particularly valuable as these combinators offer a more flexible and compositional perspective on copowers. Throughout this chapter, they will be used extensively in the development of subsequent results, including the proof of the main theorem.

Lemma 3.2.3. *In a right monoidal-closed category* $\mathbb{A}$*, if* $\gamma : Hom(X \lhd A, Y) \to A \multimap Hom(X,Y)$ *is* $\mathbb{A}$*-natural in* Y*, then* $(\mathrm{uncurry}(\gamma) \otimes 1)\mathrm{m} = \mathrm{a}_{\otimes}(1 \otimes \mathrm{m})\mathrm{uncurry}(\gamma)$.

Proof.

γ is a natural transformation between two $\mathbb{A}$-functors $G := \mathrm{Hom}(X \lhd A, -) : \mathbb{X} \to \mathbb{A}$ and $F := A \multimap \mathrm{Hom}(X, -) : \mathbb{X} \to \mathbb{A}$. Since it is $\mathbb{A}$-natural in Y, the following diagram commutes:

$$\begin{array}{ccc}
\mathrm{Hom}(X,Y)\otimes I & \xrightarrow{G\otimes\ulcorner\gamma\urcorner} & (\mathrm{Hom}(X\lhd A,X)\multimap\mathrm{Hom}(X\lhd A,Y))\otimes(\mathrm{Hom}(X\lhd A,Y)\multimap(A\multimap\mathrm{Hom}(X,Y))) \\
{\scriptstyle (u^R_\otimes)^{-1}}\uparrow & & \downarrow{\scriptstyle m^{\multimap}} \\
\mathrm{Hom}(X,Y) & & \mathrm{Hom}(X\lhd A,X)\multimap(A\multimap\mathrm{Hom}(X,Y)) \\
{\scriptstyle (u^L_\otimes)^{-1}}\downarrow & & \uparrow{\scriptstyle m^{\multimap}} \\
I\otimes\mathrm{Hom}(X,Y) & \xrightarrow[\ulcorner\gamma\urcorner\otimes F]{} & (\mathrm{Hom}(X\lhd A,Y)\multimap(A\multimap\mathrm{Hom}(X,Y)))\otimes((A\multimap\mathrm{Hom}(X,X))\multimap(A\multimap\mathrm{Hom}(X,Y)))
\end{array}$$

Which means

$$(u^R_\otimes)^{-1}(G\otimes\ulcorner\gamma\urcorner)m^{\multimap}=(u^L_\otimes)^{-1}(\ulcorner\gamma\urcorner\otimes F)m^{\multimap} \tag{3.1}$$

We can define $\ulcorner\gamma\urcorner:=\mathrm{curry}(u^R_\otimes\gamma)$ as:

$$\begin{array}{ccc}
\mathrm{Hom}(X\lhd A,Y)\otimes I & \xrightarrow{\mathrm{uncurry}(\ulcorner\gamma\urcorner)} & A\multimap\mathrm{Hom}(X,Y) \\
{\scriptstyle u^R_\otimes}\downarrow & \nearrow_{\gamma} & \\
\mathrm{Hom}(X\lhd A,Y) & &
\end{array}$$

We can define $m^{\multimap}:=\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)\epsilon)$ as below, in which $\epsilon:A\otimes(A\multimap B)\to B$ is the counit of adjunction between the $A\multimap -$ and $A\otimes -$ (known as the "evaluation" map).

$$\begin{array}{ccc}
A\otimes(A\multimap B\otimes B\multimap C) & \xrightarrow{\mathrm{uncurry}(m^{\multimap})} & C \\
{\scriptstyle (a_\otimes)^{-1}}\downarrow & & \uparrow{\scriptstyle \epsilon} \\
(A\otimes A\multimap B)\otimes B\multimap C & \xrightarrow[\epsilon\otimes 1]{} & B\otimes B\multimap C
\end{array}$$

We can define $F:=\mathrm{curry}(\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)m))$ as below:

$$\begin{array}{ccc} A \otimes ((A \multimap \mathrm{Hom}(X,X)) \otimes \mathrm{Hom}(X,Y)) & \xrightarrow{\mathrm{uncurry}(\mathrm{uncurry}((F))} & \mathrm{Hom}(X,Y) \\ \downarrow{\scriptstyle (a_\otimes)^{-1}} & & \uparrow{\scriptstyle m} \\ (A \otimes (A \multimap Hom(X,X)) \otimes \mathrm{Hom}(X,Y) & \xrightarrow[\epsilon\otimes 1]{} & \mathrm{Hom}(X,X) \otimes \mathrm{Hom}(X,Y) \end{array}$$

Finally, we can define $G := \mathrm{curry}(m)$ as below:

$$\mathrm{Hom}(X \lhd A, X) \otimes \mathrm{Hom}(X,Y) \xrightarrow{m=\mathrm{uncurry}(G)} \mathrm{Hom}(X \lhd A, Y)$$

Using the definitions above, we can rewrite the left hand side identity we have due to naturality as below:

$$\begin{aligned}
(u^R_\otimes)^{-1}(G \otimes \ulcorner\gamma\urcorner)m^{\multimap} &= (u^R_\otimes)^{-1}(\mathrm{curry}(m) \otimes \mathrm{curry}(u^R_\otimes\gamma))\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)\epsilon) \\
&= \mathrm{curry}(m)(u^R_\otimes)^{-1}(1 \otimes \mathrm{curry}(u^R_\otimes\gamma))\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)\epsilon) \\
&\qquad\qquad \text{(naturality of } (u^R_\otimes)^{-1}\text{)} \\
&= \mathrm{curry}(m)(u^R_\otimes)^{-1}\mathrm{curry}((1 \otimes (1 \otimes \mathrm{curry}(u^R_\otimes\gamma)))(a_\otimes)^{-1}(\epsilon\otimes 1)\epsilon) \\
&= \mathrm{curry}(m)(u^R_\otimes)^{-1}\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)(1 \otimes \mathrm{curry}(u^R_\otimes\gamma))\epsilon) \\
&\qquad\qquad \text{(naturality of } (a_\otimes)^{-1}\text{)} \\
&= \mathrm{curry}(m)(u^R_\otimes)^{-1}\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)(u^R_\otimes\gamma)) \\
&= \mathrm{curry}(m)\mathrm{curry}((1 \otimes (u^R_\otimes)^{-1})(a_\otimes)^{-1}(\epsilon\otimes 1)(u^R_\otimes\gamma)) \\
&= \mathrm{curry}(m)\mathrm{curry}((1 \otimes (u^R_\otimes)^{-1})(a_\otimes)^{-1}u^R_\otimes(\epsilon\gamma)) \quad \text{(naturality of } u^R_\otimes\text{)} \\
&= \mathrm{curry}(m)\mathrm{curry}(\epsilon\gamma) \\
&= \mathrm{curry}((1 \otimes \mathrm{curry}(m))\epsilon\gamma) \\
&= \mathrm{curry}(m\gamma)
\end{aligned}$$

And on the right hand side we have:

$$
\begin{aligned}
(u^L_\otimes)^{-1}(\ulcorner\gamma\urcorner \otimes F)m^{-\circ} &= (u^L_\otimes)^{-1}(1\otimes F)(\mathrm{curry}(u^R_\otimes\gamma)\otimes 1)\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)\epsilon) \\
&= F(u^L_\otimes)^{-1}(\mathrm{curry}(u^R_\otimes\gamma)\otimes 1)\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)\epsilon) \\
&\qquad\qquad \text{(naturality of } (u^L_\otimes)^{-1}) \\
&= F(u^L_\otimes)^{-1}\mathrm{curry}((1\otimes(\mathrm{curry}(u^R_\otimes\gamma)\otimes 1))(a_\otimes)^{-1}(\epsilon\otimes 1)\epsilon) \\
&= F(u^L_\otimes)^{-1}\mathrm{curry}((a_\otimes)^{-1}(\mathrm{curry}(u^R_\otimes\gamma)\otimes 1)(\epsilon\otimes 1)\epsilon) \\
&\qquad\qquad \text{(naturality of } (a_\otimes)^{-1}) \\
&= F(u^L_\otimes)^{-1}\mathrm{curry}((a_\otimes)^{-1}((1\otimes\mathrm{curry}(u^R_\otimes\gamma))\epsilon\otimes 1)\epsilon) \\
&\qquad\qquad \text{(naturality of } (a_\otimes)^{-1}) \\
&= F(u^L_\otimes)^{-1}\mathrm{curry}((a_\otimes)^{-1}(u^R_\otimes\gamma\otimes 1)\epsilon) \qquad \text{(naturality of } (a_\otimes)^{-1}) \\
&= F\mathrm{curry}((1\otimes(u^L_\otimes)^{-1})(a_\otimes)^{-1}(u^R_\otimes\gamma\otimes 1)\epsilon) \\
&= F\mathrm{curry}(((u^L_\otimes)^{-1}\otimes 1)(u^R_\otimes\gamma\otimes 1)\epsilon) \\
&= F\mathrm{curry}((\gamma\otimes 1)\epsilon) \\
&= \mathrm{curry}(\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)m))\mathrm{curry}((\gamma\otimes 1)\epsilon) \\
&= \mathrm{curry}((\gamma\otimes 1)(1\otimes\mathrm{curry}(\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)m)))\epsilon) \\
&= \mathrm{curry}((\gamma\otimes 1)\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)m))
\end{aligned}
$$

So we can re-write 3.1 as:

$$\mathrm{curry}(m\gamma) = \mathrm{curry}((\gamma\otimes 1)\mathrm{curry}((a_\otimes)^{-1}(\epsilon\otimes 1)m))$$

We may now apply the "uncurry" operation twice to both sides of the identity, yielding:

$$\text{uncurry}(\text{uncurry}(\text{curry}(m\gamma))) = \text{uncurry}(\text{uncurry}(\text{curry}((\gamma \otimes 1)\text{curry}((a_{\otimes})^{-1}(\epsilon \otimes 1)m))))$$

Now we simplify both sides again. On the left-hand side we have:

$$\begin{aligned}\text{uncurry}(\text{uncurry}(\text{curry}(m\gamma))) &= \text{uncurry}(m\gamma) \\ &= (1 \otimes m)\text{uncurry}(\gamma)\end{aligned}$$

and on the right-hand side we have:

$$\begin{aligned}\text{uncurry}(\text{uncurry}(\text{curry}((\gamma \otimes 1)\text{curry}((a_{\otimes})^{-1}(\epsilon \otimes 1)m)))) &= \text{uncurry}((\gamma \otimes 1)\text{curry}((a_{\otimes})^{-1}(\epsilon \otimes 1)m)) \\ &= (1 \otimes (\gamma \otimes 1))\text{curry}((a_{\otimes})^{-1}(\epsilon \otimes 1)m) \\ &= (1 \otimes (\gamma \otimes 1))(a_{\otimes})^{-1}(\epsilon \otimes 1)m \\ &= (a_{\otimes})^{-1}((1 \otimes \gamma)\epsilon \otimes 1)m \\ &= (a_{\otimes})^{-1}(\text{uncurry}(\gamma) \otimes 1)m\end{aligned}$$

Which means that we can re-write the identity 3.1 as:

$$(1 \otimes m)\text{uncurry}(\gamma) = (a_{\otimes})^{-1}(\text{uncurry}(\gamma) \otimes 1)m$$

And by pre-composing $a_{\otimes}$ on both sides, we get

$$a_{\otimes}(1 \otimes m)\text{uncurry}(\gamma) = (\text{uncurry}(\gamma) \otimes 1)m$$

□

We now make use of Lemma 3.2.3 to prove the following proposition. This result demonstrates that our definition of copowers in the non-closed setting is equivalent to the definition of copowers provided by [20] in the context of a closed monoidal category. This equivalence highlights the compatibility of the two approaches and justifies our choice of definition in the more general, non-closed setting.

Proposition 3.2.4. *For an $\mathbb{A}$-enriched category $\mathbb{X}$, where $\mathbb{A}$ is a right monoidal-closed category, the following statements are equivalent:*

(i) There exists an isomorphism $\gamma : Hom(X \lhd A, Y) \to A \multimap Hom(X, Y)$ which is $\mathbb{A}$-natural in Y.

(ii) There exist a bijective correspondence

$$\frac{B \xrightarrow{f=(g)^{\Uparrow}} Hom(X \lhd A, Y)}{A \otimes B \xrightarrow{g=(f)^{\Downarrow}} Hom(X, Y)}$$

and a map $\eta : A \to Hom(X, X \lhd A)$ such that $(f)^{\Downarrow} = (\eta \otimes f)m$

Proof. We start with proving $(i) \Rightarrow (ii)$. Using γ we can write:

$$\frac{\dfrac{A \otimes B \xrightarrow[(f)^{\Downarrow}=\mathrm{uncurry}(f\gamma)]{g} \mathrm{Hom}(X, Y)}{B \xrightarrow[f\gamma]{\mathrm{curry}(g)} A \multimap \mathrm{Hom}(X, Y)}}{B \xrightarrow[f]{(g)^{\Uparrow}=\mathrm{curry}(g)\gamma^{-1}} \mathrm{Hom}(X \lhd A, Y)}$$

We have to show that $((g)^{\Uparrow})^{\Downarrow} = g$ and $((f)^{\Downarrow})^{\Uparrow} = f$

$$\begin{aligned}((g)^{\Uparrow})^{\Downarrow} &= (\text{curry}(g)\gamma^{-1})^{\Downarrow} \\ &= \text{uncurry}(\text{curry}(g)\gamma^{-1}\gamma) \\ &= \text{uncurry}(\text{curry}(g)) \\ &= g\end{aligned}$$

$$\begin{aligned}((f)^{\Downarrow})^{\Uparrow} &= (\text{uncurry}(f\gamma))^{\Uparrow} \\ &= \text{curry}(\text{uncurry}(f\gamma))\gamma^{-1} \\ &= f\gamma\gamma^{-1} \\ &= f\end{aligned}$$

We can define $\eta := (u^R_{\otimes})^{-1}(1 \otimes id)\text{uncurry}(\gamma)$ as below:

$$\dfrac{\dfrac{\text{Hom}(X \lhd A, X \lhd A) \xrightarrow{\gamma} A \multimap \text{Hom}(X, X \lhd A)}{A \otimes \text{Hom}(X \lhd A, X \lhd A) \xrightarrow{\text{uncurry}(\gamma)} \text{Hom}(X, X \lhd A)}}{A \xrightarrow{(u^R_{\otimes})^{-1}} A \otimes I \xrightarrow{(1 \otimes id)} A \otimes \text{Hom}(X \lhd A, X \lhd A) \xrightarrow{\text{uncurry}(\gamma)} \text{Hom}(X, X \lhd A)}$$

We have to prove that the defined η satisfies $(f)^{\Downarrow} = (\eta \otimes f)m$:

$$\begin{aligned}
(\eta \otimes f)m &= ((u^R_{\otimes})^{-1}(1 \otimes id)\text{uncurry}(\gamma) \otimes f)m && \text{(def of } \eta\text{)} \\
&= (1 \otimes f)((u^R_{\otimes})^{-1} \otimes 1)((1 \otimes id) \otimes 1)(\text{uncurry}(\gamma) \otimes 1)m \\
&= (1 \otimes f)((u^R_{\otimes})^{-1} \otimes 1)((1 \otimes id) \otimes 1)a_{\otimes}(1 \otimes m)\text{uncurry}(\gamma) && \text{(Lemma 3.2.3)} \\
&= (1 \otimes f)((u^R_{\otimes})^{-1} \otimes 1)a_{\otimes}(1 \otimes (id \otimes 1))(1 \otimes m)\text{uncurry}(\gamma) \\
&= (1 \otimes f)((u^R_{\otimes})^{-1} \otimes 1)a_{\otimes}(1 \otimes (id \otimes 1)m)\text{uncurry}(\gamma) \\
&= (1 \otimes f)((u^R_{\otimes})^{-1} \otimes 1)a_{\otimes}(1 \otimes u^L_{\otimes})\text{uncurry}(\gamma) \\
&= (1 \otimes f)((u^R_{\otimes})^{-1} \otimes 1)(u^R_{\otimes} \otimes 1)\text{uncurry}(\gamma) \\
&= (1 \otimes f)\text{uncurry}(\gamma) \\
&= \text{uncurry}(f\gamma) \\
&= (f)^{\Downarrow}
\end{aligned}$$

Next we have to prove $(ii) \Rightarrow (i)$. We define the map $\alpha : A \multimap \text{Hom}(X, Y) \to \text{Hom}(X \lhd A, Y)$ as:

$$\frac{\dfrac{A \multimap \text{Hom}(X,Y) \xrightarrow{\alpha=(ev)^{\Uparrow}} \text{Hom}(X \lhd A, Y)}{A \otimes A \multimap \text{Hom}(X,Y) \xrightarrow{ev} \text{Hom}(X,Y)}}{A \multimap \text{Hom}(X,Y) \xrightarrow{1} A \multimap \text{Hom}(X,Y)}$$

and the map $\beta : \text{Hom}(X \lhd A, Y) \to A \multimap \text{Hom}(X, Y)$ as:

$$\frac{\dfrac{\text{Hom}(X \lhd A, Y) \xrightarrow{\beta=\text{curry}(1^{\Downarrow})} A \multimap \text{Hom}(X,Y)}{A \otimes \text{Hom}(X \lhd A, Y) \xrightarrow{1^{\Downarrow}} \text{Hom}(X,Y)}}{\text{Hom}(X \lhd A, Y) \xrightarrow{1} \text{Hom}(X \lhd A, Y)}$$

Then we have:

$$\begin{aligned}(1 \otimes \alpha\beta)ev &= (1 \otimes \alpha)(1 \otimes \beta)ev \\ &= (1 \otimes (ev)^{\Uparrow})(1 \otimes \mathrm{curry}(1^{\Downarrow}))ev \\ &= (1 \otimes (ev)^{\Uparrow})(1^{\Downarrow}) \\ &= ((ev)^{\Uparrow})^{\Downarrow} \\ &= ev\end{aligned}$$

So $\alpha\beta = 1$. We also have:

$$\begin{aligned}\alpha\beta &= \mathrm{curry}(1^{\Downarrow})(ev)^{\Uparrow} \\ &= ((1 \otimes \mathrm{curry}(1^{\Downarrow}))ev)^{\Uparrow} \\ &= (1^{\Downarrow})^{\Uparrow} \\ &= 1\end{aligned}$$

So $\beta\alpha = 1$. So α and β are mutually inverse and thus, isomorphisms.

It remains to prove that $(\mathrm{uncurry}(\gamma) \otimes 1)m = a_{\otimes}(1 \otimes m)\mathrm{uncurry}(\gamma)$.

$$\begin{aligned}(\mathrm{uncurry}(\gamma) \otimes 1)m &= (\mathrm{uncurry}(\mathrm{curry}((1)^{\Downarrow}) \otimes 1)m \\ &= ((1)^{\Downarrow} \otimes 1)m \\ &= a_{\otimes}(m)^{\Downarrow} \\ &= a_{\otimes}(1 \otimes m)(1)^{\Downarrow} \\ &= a_{\otimes}(1 \otimes m)\mathrm{uncurry}(\mathrm{curry}((1)^{\Downarrow}) \\ &= a_{\otimes}(1 \otimes m)\mathrm{uncurry}(\gamma)\end{aligned}$$

□

3.3 Right actegories with hom-objects have copowers

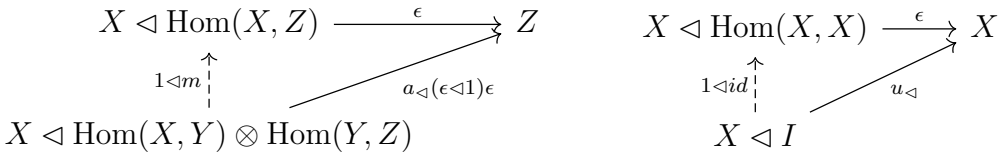
In this section, we establish the first direction of Theorem 3.1.1. Specifically, we aim to prove the following proposition, which serves as the forward implication of the theorem and constitutes a key step in the overall argument.

Proposition 3.3.1. *If $\mathbb{X}$ is a right $\mathbb{A}$-actegory with hom-objects, that is there is an adjunction*

$$\frac{A \to Hom(X,Y)}{X \lhd A \to Y}$$

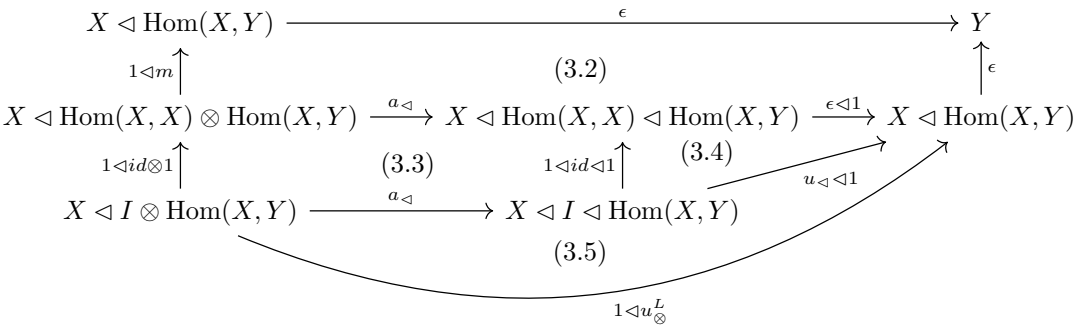
then $\mathbb{X}$ is an $\mathbb{A}$-enriched category with copowers.

Proof. We start with proving the enrichment. In the $\mathbb{A}$-actegory $\mathbb{X}$, we take 'm' to be the composition morphism and 'id' to be the unit as below:



In order to have an enrichment, we have to show the associativity of the composition and identity rules for id:

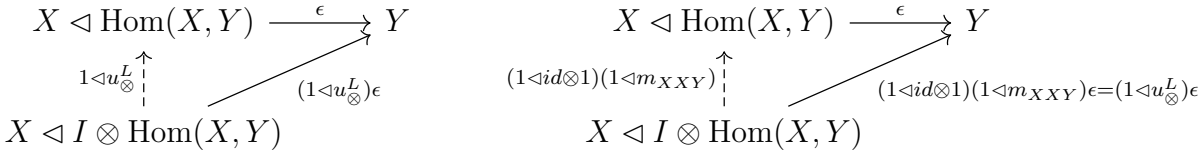
1. Left identity:



Square 3.2 is the definition of m, square 3.3 commutes because $a_{\lhd}$ is natural, triangle 3.4 is the definition of id and finally 3.5 commutes because of section 2.9. So the whole diagram above commutes,

$$(1 \lhd id_X \otimes 1)(1 \lhd m_{XXY})\epsilon = (1 \lhd u^L_{\otimes})\epsilon$$

Since ϵ is the counit of the adjunction, based on the couniversal property we have:

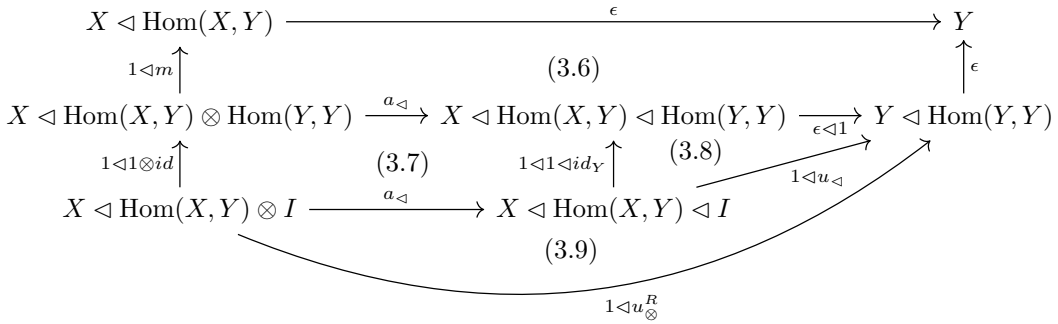


Since the map from $X \lhd I \otimes \mathrm{Hom}(X,Y)$ to $X \lhd \mathrm{Hom}(X,Y)$ must be unique, then we have:

$$1 \lhd u^L_{\otimes} = (1 \lhd id \otimes 1)(1 \lhd m_{XXY})$$

which proves the left identity.

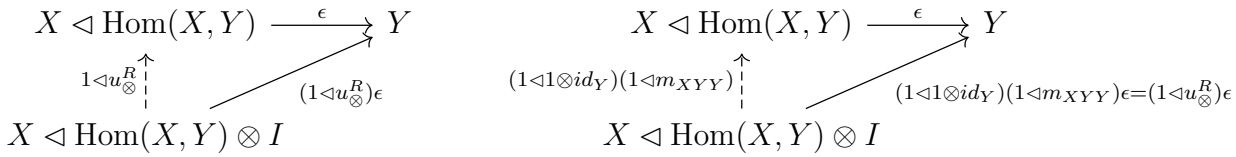
2. Right identity:



Square 3.6 is the definition of m, square 3.7 commutes because $a_{\lhd}$ is natural, triangle 3.8 is the definition of id and finally 3.8 commutes because of section 2.9. Thus the whole diagram above commutes, so we have

$$(1 \lhd 1 \otimes id_Y)(1 \lhd m_{XYY})\epsilon = (1 \lhd u^R_{\otimes})\epsilon$$

Since ϵ is the counit of the adjunction, based on the couniversal property we have:



Since the map from $X \lhd \mathrm{Hom}(X,Y) \otimes I$ to $X \lhd \mathrm{Hom}(X,Y)$ must be unique, then we have:

$$1 \lhd u^R_\otimes = (1 \lhd 1 \otimes id_Y)(1 \lhd m_{XYY})$$

which proves the identity on the right.

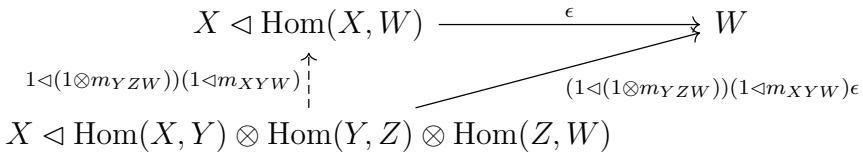
3. Associativity of composition:

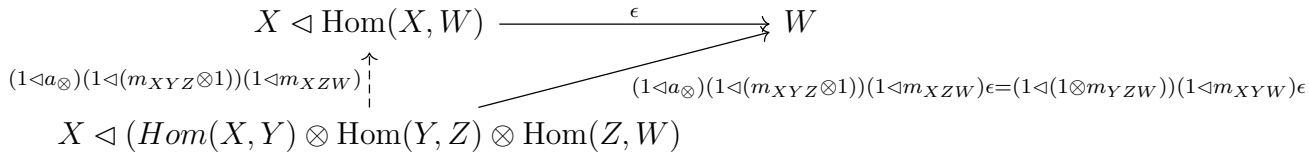
$$\begin{aligned}
(1 \lhd (1 \otimes m))(1 \lhd m)\epsilon &= (1 \lhd (1 \otimes m))a_{\lhd}(\epsilon \lhd 1)\epsilon && \text{(def of } m\text{)} \\
&= a_{\lhd}(1 \lhd m)(\epsilon \lhd 1)\epsilon && \text{(naturality of } a_{\lhd}\text{)} \\
&= a_{\lhd}(\epsilon \lhd 1)(1 \lhd m)\epsilon \\
&= a_{\lhd}(\epsilon \lhd 1)a_{\lhd}(\epsilon \lhd 1)\epsilon && \text{(def of } m\text{)} \\
&= a_{\lhd}a_{\lhd}((\epsilon \lhd 1) \lhd 1)(\epsilon \lhd 1)\epsilon && \text{(naturality of } a_{\lhd}\text{)} \\
&= (1 \lhd a_{\otimes})a_{\lhd}(a_{\lhd} \lhd 1)((\epsilon \lhd 1) \lhd 1)(\epsilon \lhd 1)\epsilon \\
& && \text{(associativity of } a_{\lhd}\text{)} \\
&= (1 \lhd a_{\otimes})a_{\lhd}(a_{\lhd}(\epsilon \lhd 1) \lhd 1)(\epsilon \lhd 1)\epsilon \\
&= (1 \lhd a_{\otimes})a_{\lhd}((1 \lhd m) \lhd 1)(\epsilon \lhd 1)\epsilon && \text{(def of } m\text{)} \\
&= (1 \lhd a_{\otimes})(1 \lhd (m \otimes 1))a_{\lhd}(\epsilon \lhd 1)\epsilon && \text{(naturality of } a_{\lhd}\text{)} \\
&= (1 \lhd a_{\otimes})(1 \lhd (m \otimes 1))(1 \lhd m)\epsilon && \text{(def of } m\text{)}
\end{aligned}$$

so we have:

$$(1 \lhd (1 \otimes m))(1 \lhd m)\epsilon = (1 \lhd a_{\otimes})(1 \lhd (m \otimes 1))(1 \lhd m)\epsilon$$

Since ϵ is the counit of the adjunction, based on the couniversal property we have:





Since the map from $X \lhd (Hom(X, Y) \otimes \mathrm{Hom}(Y, Z) \otimes \mathrm{Hom}(Z, W)$ to $X \lhd \mathrm{Hom}(X, W)$ must be unique, we have:

$$(1 \lhd a_{\otimes})(1 \lhd (m_{XYZ} \otimes 1))(1 \lhd m_{XZW}) = (1 \lhd (1 \otimes m_{YZW}))(1 \lhd m_{XYW})$$

which proves the associativity of the composition.

Above we have shown that we have an enrichment of the category $\mathbb{X}$ in the category $\mathbb{A}$.

Next, we prove that $\mathbb{X}$ has copowers over $\mathbb{A}$. Using the adjunction, we define the $(-)^{\Uparrow}$ and $(-)^{\Downarrow}$ combinators as:

$$\frac{\dfrac{B \xrightarrow{f} \mathrm{Hom}(X \lhd A, Y)}{X \lhd A \otimes B \xrightarrow{a_{\lhd}} X \lhd A \lhd B \xrightarrow{f^{\sharp}} Y}}{A \otimes B \xrightarrow{(a_{\lhd} f^{\sharp})^{\flat}} \mathrm{Hom}(X, Y)} \qquad \frac{\dfrac{A \otimes B \xrightarrow{g} \mathrm{Hom}(X, Y)}{X \lhd A \lhd B \xrightarrow{a_{\lhd}^{-1}} X \lhd A \otimes B \xrightarrow{g^{\sharp}} Y}}{B \xrightarrow{(a_{\lhd}^{-1} g^{\sharp})^{\flat}} \mathrm{Hom}(X \lhd A, Y)}$$

$(f)^{\Downarrow} = (a_{\lhd} f^{\sharp})^{\flat}$ and $(g)^{\Uparrow} = (a_{\lhd}^{-1} g^{\sharp})^{\flat}$

We have to prove that $((f)^{\Downarrow})^{\Uparrow} = f$ and $((g)^{\Uparrow})^{\Downarrow} = g$

$$\begin{aligned}((g)^{\Uparrow})^{\Downarrow} &= ((a_{\lhd}^{-1} g^{\sharp})^{\flat})^{\Downarrow} \\ &= (a_{\lhd}((a_{\lhd}^{-1} g^{\sharp})^{\flat})^{\sharp})^{\flat} \\ &= (a_{\lhd} a_{\lhd}^{-1} g^{\sharp})^{\flat} \\ &= (g^{\sharp})^{\flat} \\ &= g\end{aligned} \qquad \begin{aligned}((f)^{\Downarrow})^{\Uparrow} &= ((a_{\lhd} f^{\sharp})^{\flat})^{\Uparrow} \\ &= (a_{\lhd}^{-1}((a_{\lhd} f^{\sharp})^{\flat})^{\sharp})^{\flat} \\ &= (a_{\lhd}^{-1} a_{\lhd} f^{\sharp})^{\flat} \\ &= (f^{\sharp})^{\flat} \\ &= f\end{aligned}$$

We take $\eta := 1^{\flat}$ (the unit of the ajdunction). We have to prove that $(f)^{\Downarrow} = (\eta \otimes f)m$

$$\begin{aligned}(f)^{\Downarrow} &= (a_{\lhd} f^{\sharp})^{\flat} \\ &= (a_{\lhd}(1 \lhd f)\epsilon)^{\flat} && (f^{\sharp} = (1 \lhd f)\epsilon) \\ &= ((1 \lhd (1 \otimes f))a_{\lhd}\epsilon)^{\flat} && (\text{naturality of } a_{\lhd}) \\ &= (1 \otimes f)a_{\lhd}^{\flat}\mathrm{Hom}(1, \epsilon) \\ &= (1 \otimes f)((1 \lhd (\eta \otimes \eta)m)\epsilon)^{\flat}\mathrm{Hom}(1, \epsilon) \\ &= (1 \otimes f)(\eta \otimes \eta)m\epsilon^{\flat}\mathrm{Hom}(1, \epsilon) \\ &= (1 \otimes f)(\eta \otimes \eta)m(1^{\sharp})^{\flat}\mathrm{Hom}(1, \epsilon) \\ &= (1 \otimes f)(\eta \otimes \eta)m\mathrm{Hom}(1, \epsilon) \\ &= (\eta \otimes f)(1 \otimes \eta\mathrm{Hom}(1, \epsilon))m \\ &= (\eta \otimes f)m && (\text{triangle identity})\end{aligned}$$

□

3.4 Categories with copowers are right actegories with hom-object

In the previous section, we established that any right actegory equipped with hom-objects admits copowers. In this section, we prove the converse direction of Theorem 3.1.1. This will be accomplished by proving the following proposition, which demonstrates that the existence of copowers (together with certain coherence conditions) implies the presence of a right actegory structure with hom-objects.

Proposition 3.4.1. *In any $\mathbb{A}$-enriched category $\mathbb{X}$ with copowers, for each $X \in \mathbb{X}$ there is an adjunction*

$$\frac{A \to Hom(X, Y)}{X \lhd A \to Y}$$

and $\mathbb{X}$ is an $\mathbb{A}$-actegory, with an action $\lhd : \mathbb{X} \times \mathbb{A} \to \mathbb{X}$ and two morphisms $a_\lhd := (1 \lhd (\eta \otimes \eta)m)\epsilon : X \lhd A \otimes B \to X \lhd A \lhd B$ and $u_\lhd := (1 \lhd id)\epsilon : X \lhd I \to X$.

Proof. We start with proving that $Hom(X, -)$ is a functor. We have:

$$\begin{aligned} \mathrm{Hom}(1,1) &= (u^R_\otimes)^{-1}(1 \otimes \ulcorner 1 \urcorner)m \\ &= (u^R_\otimes)^{-1}(1 \otimes id)m \\ &= (u^R_\otimes)^{-1}u^R_\otimes \\ &= 1 \end{aligned}$$

So $\mathrm{Hom}(X,-)$ satisfies the identity, and we have:

$$\begin{aligned}
\mathrm{Hom}(1,fg) &= (u^R_\otimes)^{-1}(1\otimes \ulcorner fg \urcorner)m \\
&= (u^R_\otimes)^{-1}(1\otimes((u^L_\otimes)^{-1}(\ulcorner f\urcorner\otimes\ulcorner g\urcorner)m))m \\
&= (u^R_\otimes)^{-1}(1\otimes(u^L_\otimes)^{-1})(1\otimes(\ulcorner f\urcorner\otimes\ulcorner g\urcorner)m)m \\
&= (u^R_\otimes)^{-1}(1\otimes(u^L_\otimes)^{-1})a_\otimes((1\otimes\ulcorner f\urcorner)m\otimes\ulcorner g\urcorner)m \\
&= (u^R_\otimes)^{-1}((u^R_\otimes)^{-1}\otimes 1)((1\otimes\ulcorner f\urcorner)m\otimes\ulcorner g\urcorner)m \\
&= (u^R_\otimes)^{-1}((u^R_\otimes)^{-1}(1\otimes\ulcorner f\urcorner)m\otimes\ulcorner g\urcorner)m \\
&= (u^R_\otimes)^{-1}(\mathrm{Hom}(1,f)\otimes\ulcorner g\urcorner)m \\
&= \mathrm{Hom}(1,f)\mathrm{Hom}(g,f)
\end{aligned}$$

Which shows that $\mathrm{Hom}(X,-)$ also satisfies the composition, and thus is a functor.

Next we prove that the unit of the copower $\eta : A \to \mathrm{Hom}(X, X\lhd A)$ has the universal property.

Using the property of copowers, we have $\ulcorner f^\sharp\urcorner = (u^R_\otimes f)^{\Uparrow}$:

$$\frac{A\otimes I \xrightarrow{u^R_\otimes} A \xrightarrow{f} \mathrm{Hom}(X,y)}{I \xrightarrow{\ulcorner f^\sharp\urcorner=(u^R_\otimes f)^{\Uparrow}} \mathrm{Hom}(X\lhd A, Y)}$$

So We have:

$$\begin{aligned}
\eta\mathrm{Hom}(1, f^\sharp) &= (u^R_\otimes)^{-1}(\eta \otimes \ulcorner f^\sharp \urcorner)m \\
&= (u^R_\otimes)^{-1}(\eta \otimes (u^R_\otimes f)^{\Uparrow})m \\
&= (u^R_\otimes)^{-1}((u^R_\otimes f)^{\Uparrow})^{\Downarrow} \\
&= (u^R_\otimes)^{-1}u^R_\otimes f \\
&= f
\end{aligned}$$

Which proves that the diagram below commutes:

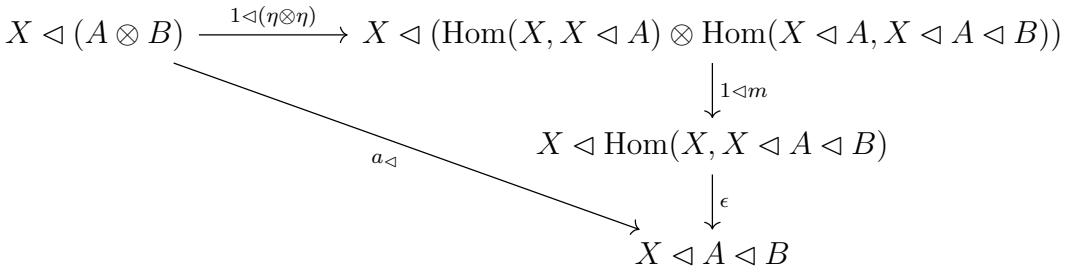
$$\begin{array}{ccc}
A & \xrightarrow{\eta} & \mathrm{Hom}(X, X \lhd A) \\
& \searrow^{f} & \downarrow{\scriptstyle \mathrm{Hom}(1,f^\sharp)} \\
& & \mathrm{Hom}(X, Y)
\end{array}$$

It remains to prove that for any f, $\eta\mathrm{Hom}(1, f^\sharp)$ is unique. Suppose there is another map $\mathrm{Hom}(1, g) : \mathrm{Hom}(X, Y) \to \mathrm{Hom}(X, X \lhd A)$, such that $\eta\mathrm{Hom}(1, g) = f$. We have:

$$\begin{aligned}
\eta\mathrm{Hom}(1, g) &= (u^R_\otimes)^{-1}(\eta \otimes \ulcorner g \urcorner)m \\
&= (u^R_\otimes)^{-1}(\ulcorner g \urcorner)^{\Downarrow} = f
\end{aligned}$$

which means $(\ulcorner g \urcorner)^{\Downarrow} = u^R_\otimes f$. By applying $(-)^{\Uparrow}$ on both sides we get $\ulcorner g \urcorner = (u^R_\otimes f)^{\Uparrow} = \ulcorner f^\sharp \urcorner$ which means $\mathrm{Hom}(1, g) = \mathrm{Hom}(1, f^\sharp)$ so this proves the uniqueness of $\mathrm{Hom}(1, f^\sharp)$. Therefore, η has the universal property. As a result, the adjunction exists, $X \lhd - : \mathbb{A} \to \mathbb{X}$ is a functor and $\epsilon : X \lhd \mathrm{Hom}(X, Y) \to Y$ is the counit of the adjunction.

Next we prove that $\mathbb{X}$ is an $\mathbb{A}$-actegory. Let $a_{\lhd} : X \lhd (A \otimes B) \to X \lhd A \lhd B$ be defined as:



$a^{-1}_{\lhd} : X \lhd A \lhd B \to X \lhd (A \otimes B)$ can be expressed as a map $\ulcorner a^{-1}_{\lhd} \urcorner : I \to \mathrm{Hom}(X \lhd A \lhd B, X \lhd (A \otimes B))$ in category $\ulcorner \mathbb{X} \urcorner$, which can be obtained using the property of the copower as:

$$\dfrac{\dfrac{\dfrac{A \otimes B \xrightarrow{\eta} \mathrm{Hom}(X, X \lhd (A \otimes B))}{B \xrightarrow{(\eta)^{\Uparrow}} \mathrm{Hom}(X \lhd A, X \lhd (A \otimes B))}}{B \otimes I \xrightarrow{u^R_{\otimes}(\eta)^{\Uparrow}} \mathrm{Hom}(X \lhd A, X \lhd (A \otimes B))}}{I \xrightarrow{(u^R_{\otimes}(\eta)^{\Uparrow})^{\Uparrow}} \mathrm{Hom}(X \lhd A \lhd B, X \lhd (A \otimes B))}$$

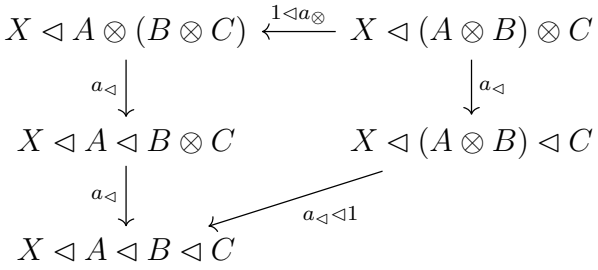
Thus,

$$
\begin{aligned}
a_{\lhd}a_{\lhd}^{-1} &= ((1 \lhd (\eta \otimes \eta)m)\epsilon)a_{\lhd}^{-1} \\
&= (1 \lhd (\eta \otimes \eta)m\mathrm{Hom}(1, a_{\lhd}^{-1}))\epsilon && \text{(naturality of } \epsilon\text{)} \\
&= (1 \lhd (\eta \otimes \eta\mathrm{Hom}(1, a_{\lhd}^{-1}))m)\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \lhd (\eta \otimes ((u_{\otimes}^{R})^{-1}(\eta \otimes \ulcorner a_{\lhd}^{-1\urcorner})m))m)\epsilon && \text{(corollary 3.2.2)} \\
&= (1 \lhd (\eta \otimes ((u_{\otimes}^{R})^{-1}(\ulcorner a_{\lhd}^{-1\urcorner})^{\Downarrow})m)\epsilon && \text{(definition 3.2.1)} \\
&= (1 \lhd (\eta \otimes ((u_{\otimes}^{R})^{-1}((u_{\otimes}^{R}(\eta)^{\Uparrow})^{\Uparrow})^{\Downarrow})m)\epsilon && \text{(def of } \ulcorner a_{\lhd}^{-1\urcorner}\text{)} \\
&= (1 \lhd (\eta \otimes ((u_{\otimes}^{R})^{-1}u_{\otimes}^{R}(\eta)^{\Uparrow})m)\epsilon && (((f)^{\Uparrow})^{\Downarrow} = f) \\
&= (1 \lhd (\eta \otimes (\eta)^{\Uparrow})m)\epsilon \\
&= (1 \lhd ((\eta)^{\Uparrow})^{\Downarrow})\epsilon && \text{(definition 3.2.1)} \\
&= (1 \lhd \eta)\epsilon \\
&= 1 && \text{(triangle identity)}
\end{aligned}
$$

$$
\begin{aligned}
a_{\lhd}^{-1}a_{\lhd} &= (1 \lhd ((u_{\otimes}^{R})^{-1}(\eta \otimes \ulcorner a_{\lhd}^{-1} \urcorner)m))\epsilon a_{\lhd} \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}(\eta \otimes \ulcorner a_{\lhd}^{-1} \urcorner)m\mathrm{Hom}(1, a_{\lhd}))\epsilon && \text{(naturality of } \epsilon) \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((\eta \otimes \ulcorner a_{\lhd}^{-1} \urcorner\mathrm{Hom}(1, a_{\lhd}^{-1}))m)\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((\eta \otimes (u_{\otimes}^{R}(\eta)^{\Uparrow})^{\Uparrow}\mathrm{Hom}(1, a_{\lhd}))m)\epsilon && \text{(def of } \ulcorner a_{\lhd}^{-1} \urcorner) \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((\eta \otimes (u_{\otimes}^{R}(\eta)^{\Uparrow}\mathrm{Hom}(1, a_{\lhd}))^{\Uparrow})m)\epsilon && \text{(corollary 3.2.2)} \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((\eta \otimes (u_{\otimes}^{R}(\eta\mathrm{Hom}(1, a_{\lhd}))^{\Uparrow})^{\Uparrow})m)\epsilon && \text{(corollary 3.2.2)} \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((\eta \otimes (u_{\otimes}^{R}(\eta\mathrm{Hom}(1, (1 \lhd (\eta)^{\Downarrow})\epsilon)))^{\Uparrow})^{\Uparrow})m)\epsilon && \text{(def of } a_{\lhd}) \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((\eta \otimes (u_{\otimes}^{R}((\eta)^{\Downarrow}\eta\mathrm{Hom}(1, \epsilon)))^{\Uparrow})^{\Uparrow})m)\epsilon && \text{(naturality of } \eta) \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((\eta \otimes (u_{\otimes}^{R}((\eta)^{\Downarrow})^{\Uparrow})^{\Uparrow})m)\epsilon && \text{(triangle identity)} \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((\eta \otimes (u_{\otimes}^{R}\eta)^{\Uparrow})m)\epsilon && (((f)^{\Downarrow})^{\Uparrow} = f) \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}((u_{\otimes}^{R}\eta)^{\Uparrow})^{\Downarrow})\epsilon && \text{(definition 3.2.1)} \\
&= (1 \lhd ((u_{\otimes}^{R})^{-1}u_{\otimes}^{R}\eta)\epsilon && (((f)^{\Downarrow})^{\Uparrow} = f) \\
&= (1 \lhd \eta)\epsilon \\
&= 1 && \text{(triangle identity)}
\end{aligned}
$$

Therefore we can conclude that $a_{\lhd}$ is an isomorphism.

In order to have an actegory, the following coherence for $a_\triangleleft$ must hold:



Based on our definition of $a_\triangleleft$, on the left-hand-side of the diagram we have:

$$
\begin{aligned}
(1 \triangleleft a_\otimes)a_\triangleleft a_\triangleleft &= (1 \triangleleft a_\otimes)(1 \triangleleft (\eta \otimes \eta)m)\epsilon a_\triangleleft \\
&= (1 \triangleleft a_\otimes(\eta \otimes \eta)m)\epsilon a_\triangleleft \\
&= (1 \triangleleft a_\otimes(\eta \otimes \eta)m\mathrm{Hom}(1, a_\triangleleft))\epsilon && \text{(naturality of } \epsilon\text{)} \\
&= (1 \triangleleft a_\otimes(\eta \otimes \eta\mathrm{Hom}(1, a_\triangleleft))m)\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \triangleleft a_\otimes(\eta \otimes \eta\mathrm{Hom}(1, (1 \triangleleft (\eta \otimes \eta)m)\epsilon))m)\epsilon && \text{(def of } a_\triangleleft\text{)} \\
&= (1 \triangleleft a_\otimes(\eta \otimes (\eta \otimes \eta)m\eta\mathrm{Hom}(1, \epsilon))m)\epsilon && \text{(naturality of } \eta\text{)} \\
&= (1 \triangleleft a_\otimes(\eta \otimes (\eta \otimes \eta)m)m)\epsilon && \text{(triangle identity)}
\end{aligned}
$$

And on the right-hand-side we have:

$$
\begin{aligned}
a_{\triangleleft}(a_{\triangleleft} \triangleleft 1) &= (1 \triangleleft (\eta \otimes \eta)m)\epsilon(a_{\triangleleft} \triangleleft 1) && \text{(definition of } a_{\triangleleft}\text{)} \\
&= (1 \triangleleft (\eta \otimes \eta)m\mathrm{Hom}(1, a_{\triangleleft} \triangleleft 1))\epsilon && \text{(naturality of } \epsilon\text{)} \\
&= (1 \triangleleft (\eta \otimes \eta\mathrm{Hom}(1, a_{\triangleleft} \triangleleft 1))m)\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \triangleleft (\eta \otimes \eta\mathrm{Hom}(a_{\triangleleft}, 1))m)\epsilon && \text{(corollary 2.6.2)} \\
&= (1 \triangleleft (\eta \otimes ((u^R_{\otimes})^{-1}(\ulcorner a_{\triangleleft} \urcorner \otimes \eta)m))m)\epsilon \\
&= (1 \triangleleft ((u^R_{\otimes})^{-1}(\eta \otimes (\ulcorner a_{\triangleleft} \urcorner \otimes \eta))m)m)\epsilon \\
&= (1 \triangleleft ((u^L_{\otimes})^{-1}(\eta \otimes \ulcorner a_{\triangleleft} \urcorner)m \otimes \eta))m)\epsilon && \text{(associativity of } m\text{)} \\
&= (1 \triangleleft ((\eta\mathrm{Hom}(1, a_{\triangleleft}) \otimes \eta))m)\epsilon \\
&= (1 \triangleleft ((\eta\mathrm{Hom}(1, (1 \triangleleft (\eta \otimes \eta)m)\epsilon) \otimes \eta))m)\epsilon && \text{(def of } a_{\triangleleft}\text{)} \\
&= (1 \triangleleft (((\eta \otimes \eta)m\eta\mathrm{Hom}(1, \epsilon) \otimes \eta))m)\epsilon && \text{(naturality of } \eta\text{)} \\
&= (1 \triangleleft (((\eta \otimes \eta)m \otimes \eta))m)\epsilon && \text{(triangle identity)} \\
&= (1 \triangleleft (((\eta \otimes (\eta \otimes \eta)m)m)\epsilon && \text{(associativity of } m\text{)}
\end{aligned}
$$

Therefore the right-hand-side and the left-hand-side of the diagram are equal, so the diagram commutes and the coherence for $a_{\triangleleft}$ holds.

One of the most effective ways to prove such identities is through the use of **circuit diagrams**. Figure 3.1 illustrates a simplified circuit diagram corresponding to the right-hand side of the identity discussed above, and Figure 3.2 depicts the left-hand side. As shown, the simplified diagrams are identical, confirming the equality of the two sides.

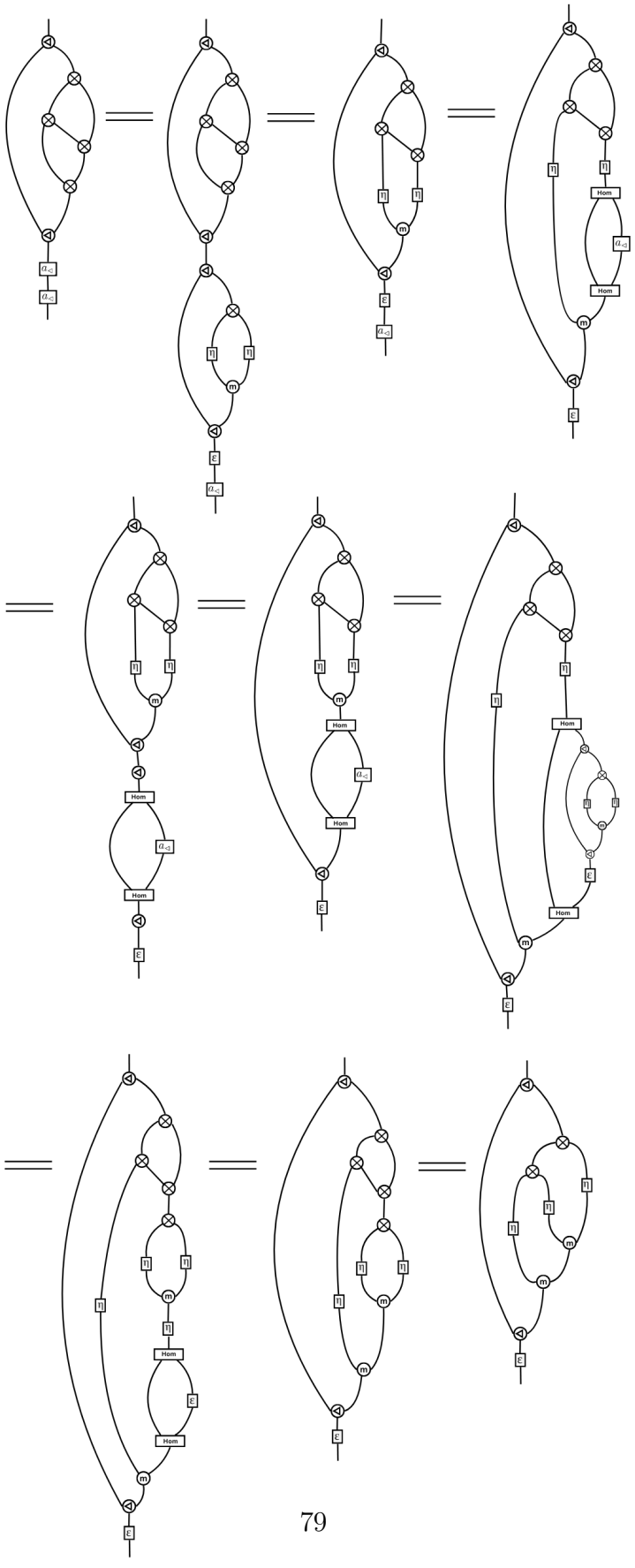


Figure 3.1: Circuit diagram proof for $(1 \lhd a_{\otimes})a_{\lhd}a_{\lhd} = (1 \lhd a_{\otimes}(\eta \otimes (\eta \otimes \eta)m)m)\epsilon$

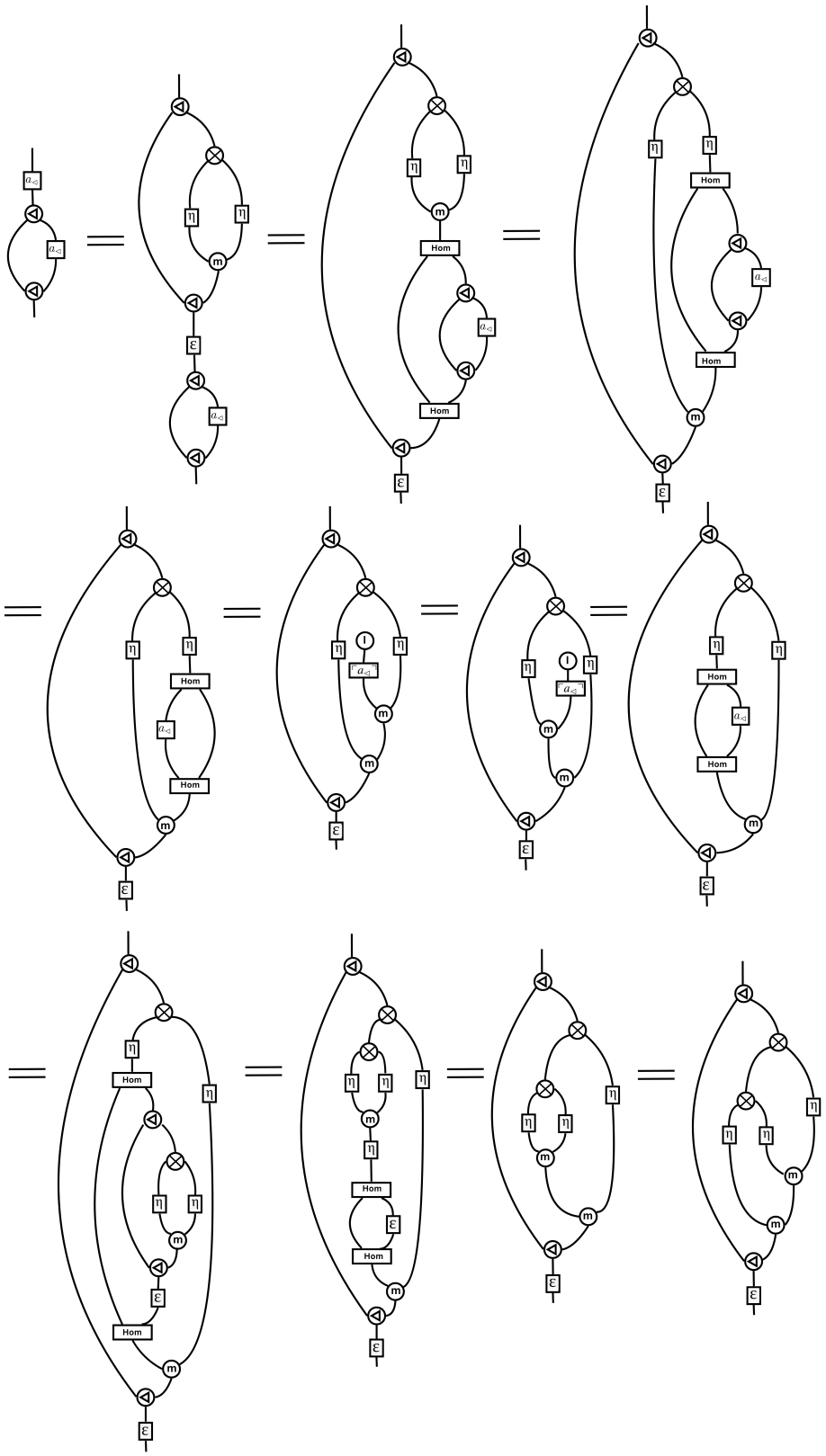
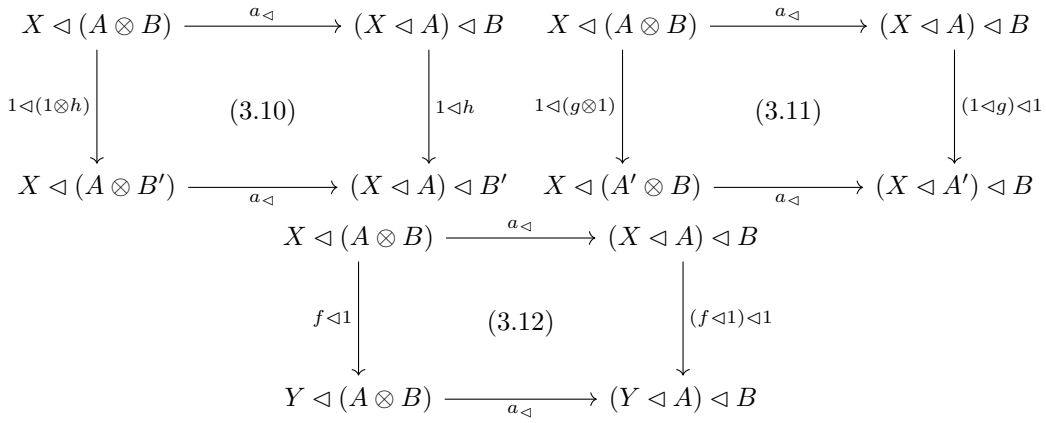


Figure 3.2: Circuit diagram proof for $a_{\triangleleft}(1 \triangleleft a_{\triangleleft}) = (1 \triangleleft a_{\otimes}(\eta \otimes (\eta \otimes \eta)m)m)\epsilon$

We also need $a_{\lhd}$ to be natural in every argument. Therefore, the following diagrams should commute:



For (3.10) we have:

$$
\begin{aligned}
a_{\lhd}(1 \lhd h) &= (1 \lhd (\eta \otimes \eta)m)\epsilon(1 \lhd h) \\
&= (1 \lhd (\eta \otimes \eta)m\mathrm{Hom}(1, 1 \lhd h))\epsilon && \text{(naturality of } \epsilon\text{)} \\
&= (1 \lhd (\eta \otimes (\eta\mathrm{Hom}(1, 1 \lhd h)))m)\epsilon \\
&= (1 \lhd (\eta \otimes (h\eta))m)\epsilon && \text{(naturality of } \eta\text{)} \\
&= (1 \lhd (1 \otimes h)(\eta \otimes \eta)m)\epsilon \\
&= (1 \lhd (1 \otimes h))(1 \lhd (\eta \otimes \eta)m)\epsilon \\
&= (1 \lhd (1 \otimes h))a_{\lhd}
\end{aligned}
$$

For (3.11) we have:

$$
\begin{aligned}
a_{\lhd}((1 \lhd g) \lhd 1) &= (1 \lhd (\eta \otimes \eta)m)\epsilon((1 \lhd g) \lhd 1) \\
&= (1 \lhd (\eta \otimes \eta)m\mathrm{Hom}(1, ((1 \lhd g) \lhd 1)))\epsilon && \text{(naturality of } \epsilon\text{)} \\
&= (1 \lhd (\eta \otimes (\eta\mathrm{Hom}(1, ((1 \lhd g) \lhd 1))))m)\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \lhd (\eta \otimes (\eta\mathrm{Hom}((1 \lhd g), 1)))m)\epsilon && \text{(corollary 2.6.2)} \\
&= (1 \lhd (\eta \otimes ((u^R_{\otimes})^{-1}(\ulcorner 1 \lhd g \urcorner \otimes \eta)m))m)\epsilon \\
&= (1 \lhd (((u^L_{\otimes})^{-1}(\eta \otimes \ulcorner 1 \lhd g \urcorner)m) \otimes \eta))m)\epsilon && \text{(associativity of } m\text{)} \\
&= (1 \lhd (\eta\mathrm{Hom}(1, 1 \lhd g) \otimes \eta)m)\epsilon \\
&= (1 \lhd (g\eta \otimes \eta)m)\epsilon && \text{(naturality of } \eta\text{)} \\
&= (1 \lhd (g \otimes 1)(\eta \otimes \eta)m)\epsilon \\
&= (1 \lhd (g \otimes 1))(1 \lhd (\eta \otimes \eta)m)\epsilon \\
&= (1 \lhd (g \otimes 1))a_{\lhd}
\end{aligned}
$$

For (3.12) we have:

$$
\begin{aligned}
a_{\lhd}((f \lhd 1) \lhd 1) &= (1 \lhd (\eta \otimes \eta)m)\epsilon((f \lhd 1) \lhd 1) \\
&= (1 \lhd (\eta \otimes \eta)m\mathrm{Hom}(1, (f \lhd 1) \lhd 1))\epsilon && \text{(naturality of } \epsilon) \\
&= (1 \lhd (\eta \otimes (\eta\mathrm{Hom}(1, (f \lhd 1) \lhd 1)))m)\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \lhd (\eta \otimes (\eta\mathrm{Hom}(f \lhd 1, 1)))m)\epsilon && \text{(corollary 2.6.2)} \\
&= (1 \lhd (\eta \otimes ((u^L_{\otimes})^{-1}(\ulcorner f \lhd 1 \urcorner \otimes \eta)m))m)\epsilon \\
&= (1 \lhd (1 \otimes (u^L_{\otimes})^{-1})(\eta \otimes ((\ulcorner f \lhd 1 \urcorner \otimes \eta)m))m)\epsilon \\
&= (1 \lhd (1 \otimes (u^L_{\otimes})^{-1})a_{\otimes}((\eta \otimes \ulcorner f \lhd 1 \urcorner)m \otimes \eta)m)\epsilon && \text{(associativity of } m) \\
&= (1 \lhd ((u^R_{\otimes})^{-1}(\eta \otimes \ulcorner f \lhd 1 \urcorner)m \otimes \eta)m)\epsilon \\
&= (1 \lhd ((\eta\mathrm{Hom}(1, f \lhd 1)) \otimes \eta)m)\epsilon \\
&= (1 \lhd ((\eta\mathrm{Hom}(f, 1)) \otimes \eta)m)\epsilon && \text{(corollary 2.6.2)} \\
&= (1 \lhd (((u^R_{\otimes})^{-1}(\ulcorner f \urcorner \otimes \eta)m) \otimes \eta)m)\epsilon \\
&= (1 \lhd (u^R_{\otimes})^{-1}(\ulcorner f \urcorner \otimes (\eta \otimes \eta)m)m)\epsilon && \text{(associativity of } m) \\
&= (1 \lhd (u^R_{\otimes})^{-1}(\ulcorner f \urcorner \otimes (\eta \otimes \eta)m\eta\mathrm{Hom}(1, \epsilon))m)\epsilon && \text{(triangle identity)} \\
&= (1 \lhd (u^R_{\otimes})^{-1}(\ulcorner f \urcorner \otimes \eta\mathrm{Hom}(1, 1 \lhd ((\eta \otimes \eta)m)\epsilon))m)\epsilon && \text{(naturality of } \eta) \\
&= (1 \lhd (u^R_{\otimes})^{-1}(\ulcorner f \urcorner \otimes \eta\mathrm{Hom}(1, a_{\lhd}))m)\epsilon && \text{(def of } a_{\lhd}) \\
&= (1 \lhd (u^R_{\otimes})^{-1}(\ulcorner f \urcorner \otimes \eta)m\mathrm{Hom}(1, a_{\lhd}))\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \lhd (u^R_{\otimes})^{-1}(\ulcorner f \urcorner \otimes \eta)m)\epsilon a_{\lhd} && \text{(naturality of } \epsilon) \\
&= (1 \lhd (\eta\mathrm{Hom}(f, 1)))\epsilon a_{\lhd} \\
&= (f \lhd 1)a_{\lhd}
\end{aligned}
$$

Therefore $a_{\lhd}$ is natural.

The other map required to form an actegory is $u_\triangleleft : X \triangleleft I \to I$ which is defined by $u_\triangleleft := (1 \triangleleft id)\epsilon$:

$$\begin{array}{ccc} X \triangleleft I & \xrightarrow{1 \triangleleft id} & X \triangleleft \mathrm{Hom}(X, X) \\ & \searrow^{u_\triangleleft} & \downarrow \epsilon \\ & & X \end{array}$$

We can also express $u_\triangleleft$ as a map $\ulcorner u_\triangleleft \urcorner : I \to \mathrm{Hom}(X \triangleleft I, X)$ in the category $\ulcorner \mathbb{X} \urcorner$ which can be obtained using the property of copowers as:

$$\frac{I \xrightarrow{u_\triangleleft = (u^L_\otimes id)^{\Uparrow}} \mathrm{Hom}(X \triangleleft I, X)}{I \otimes I \xrightarrow{u^L_\otimes} I \xrightarrow{id} \mathrm{Hom}(X, X)}$$

The inverse of $\ulcorner u_\triangleleft \urcorner$ can be written as $\ulcorner u^{-1}_\triangleleft \urcorner := \eta_I : I \to \mathrm{Hom}(X, X \triangleleft I)$. We must show that $u^{-1}_\triangleleft$ is inverse to $u_\triangleleft$ that is equivalently the following:

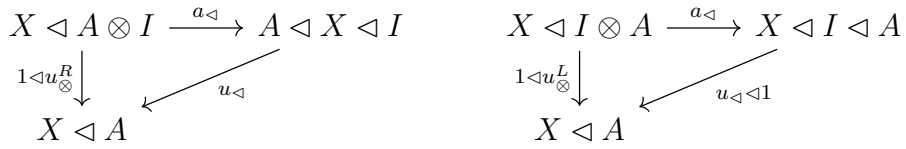
$$\begin{aligned} u_\triangleleft u^{-1}_\triangleleft &= (u^R_\otimes)^{-1}(\ulcorner u_\triangleleft \urcorner \otimes \ulcorner u^{-1}_\triangleleft \urcorner)m \\ &= (u^R_\otimes)^{-1}((u^L_\otimes id)^{\Uparrow} \otimes \eta)m \\ &= (u^R_\otimes)^{-1}(((u^L_\otimes id) \otimes \eta)m)^{\Uparrow} \\ &= ((1 \otimes (u^R_\otimes)^{-1})(u^L_\otimes \otimes 1)(id \otimes \eta)m)^{\Uparrow} \\ &= (u^L_\otimes (u^L_\otimes)^{-1}(id \otimes \eta)m)^{\Uparrow} \\ &= (u^L_\otimes \eta)^{\Uparrow} \\ &= id \end{aligned}$$

and

$$\begin{aligned} u_{\triangleleft}^{-1}u_{\triangleleft} &= (u_{\otimes}^{L})^{-1}(\ulcorner u_{\triangleleft}^{-1}\urcorner \otimes \ulcorner u_{\triangleleft}\urcorner)m \\ &= (u_{\otimes}^{L})^{-1}(\eta \otimes (u_{\otimes}^{L}id)^{\Uparrow})m \\ &= (u_{\otimes}^{L})^{-1}((u_{\otimes}^{L}id)^{\Uparrow})^{\Downarrow} \\ &= (u_{\otimes}^{L})^{-1}u_{\otimes}^{L}id \\ &= id \end{aligned}$$

Therefore $u_{\triangleleft}$ is an isomorphism.

Next, we show that $u_{\triangleleft}$ satisfies the following coherences:



$$\begin{aligned} a_{\triangleleft}u_{\triangleleft} &= (1 \triangleleft (\eta \otimes \eta)m)\epsilon u_{\triangleleft} \\ &= (1 \triangleleft (\eta \otimes \eta)m\mathrm{Hom}(1, u_{\triangleleft}))\epsilon && \text{(naturality of } \epsilon\text{)} \\ &= (1 \triangleleft (\eta \otimes (\eta\mathrm{Hom}(1, u_{\triangleleft})))m)\epsilon && \text{(lemma 2.8.1)} \\ &= (1 \triangleleft (\eta \otimes (\eta\mathrm{Hom}(1, (1 \triangleleft id)\epsilon)))m)\epsilon && \text{(def of } u_{\triangleleft}\text{)} \\ &= (1 \triangleleft (\eta \otimes (id\eta\mathrm{Hom}(1, \epsilon)))m)\epsilon && \text{(naturality of } \eta\text{)} \\ &= (1 \triangleleft (\eta \otimes id)m)\epsilon && \text{(triangle identity)} \\ &= (1 \triangleleft u_{\otimes}^{R}\eta)\epsilon \\ &= (1 \triangleleft u_{\otimes}^{R})(1 \triangleleft \eta)\epsilon \\ &= 1 \triangleleft u_{\otimes}^{R} && \text{(triangle identity)} \end{aligned}$$

$$
\begin{aligned}
a_{\triangleleft}(u_{\triangleleft} \triangleleft 1) &= (1 \triangleleft (\eta \otimes \eta)m)\epsilon(u_{\triangleleft} \triangleleft 1) \\
&= (1 \triangleleft (\eta \otimes \eta)m\mathrm{Hom}(1, u_{\triangleleft} \triangleleft 1))\epsilon && \text{(naturality of } \epsilon\text{)} \\
&= (1 \triangleleft (\eta \otimes (\eta\mathrm{Hom}(1, u_{\triangleleft} \triangleleft 1)))m)\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \triangleleft (\eta \otimes (\eta\mathrm{Hom}(u_{\triangleleft}, 1)))m)\epsilon && \text{(corollary 2.6.2)} \\
&= (1 \triangleleft (\eta \otimes ((u^L_{\otimes})^{-1}(\ulcorner u_{\triangleleft} \urcorner \otimes \eta)m))m)\epsilon \\
&= (1 \triangleleft (1 \otimes (u^L_{\otimes})^{-1})(\eta \otimes (\ulcorner u_{\triangleleft} \urcorner \otimes \eta)m)m)\epsilon \\
&= (1 \triangleleft (1 \otimes (u^L_{\otimes})^{-1})a_{\otimes}((\eta \otimes \ulcorner u_{\triangleleft} \urcorner)m \otimes \eta)m)\epsilon && \text{(associativity of } m\text{)} \\
&= (1 \triangleleft ((u^R_{\otimes})^{-1} \otimes 1)((\eta \otimes \ulcorner u_{\triangleleft} \urcorner)m \otimes \eta)m)\epsilon \\
&= (1 \triangleleft ((u^R_{\otimes})^{-1}(\eta \otimes \ulcorner u_{\triangleleft} \urcorner)m \otimes \eta)m)\epsilon \\
&= (1 \triangleleft (\eta\mathrm{Hom}(1, u_{\triangleleft}) \otimes \eta)m)\epsilon \\
&= (1 \triangleleft (\eta\mathrm{Hom}(1, (1 \triangleleft id)\epsilon) \otimes \eta)m)\epsilon && \text{(def of } u_{\triangleleft}\text{)} \\
&= (1 \triangleleft (id\eta\mathrm{Hom}(1, \epsilon) \otimes \eta)m)\epsilon && \text{(naturality of } \eta\text{)} \\
&= (1 \triangleleft (id \otimes \eta)m)\epsilon && \text{(triangle identity)} \\
&= (1 \triangleleft u^L_{\otimes}\eta)\epsilon \\
&= (1 \triangleleft u^L_{\otimes})(1 \triangleleft \eta)\epsilon \\
&= 1 \triangleleft u^L_{\otimes} && \text{(triangle identity)}
\end{aligned}
$$

Finally, $u_\triangleleft$ has to be natural, so the following diagaram must commute:

$$\begin{array}{ccc} X \triangleleft I & \xrightarrow{u_\triangleleft} & X \\ {\scriptstyle f\triangleleft 1}\downarrow & & \downarrow{\scriptstyle f} \\ Y \triangleleft I & \xrightarrow[u_\triangleleft]{} & Y \end{array}$$

$$\begin{aligned}
(f \triangleleft 1)u_\triangleleft &= (1 \triangleleft (\eta \mathrm{Hom}(f,1)))\epsilon u_\triangleleft \\
&= (1 \triangleleft ((u^R_\otimes)^{-1}(\ulcorner f \urcorner \otimes \eta)m))\epsilon u_\triangleleft \\
&= (1 \triangleleft (((u^R_\otimes)^{-1}(\ulcorner f \urcorner \otimes \eta)m)\mathrm{Hom}(1, u_\triangleleft)))\epsilon && \text{(naturality of } \epsilon\text{)} \\
&= (1 \triangleleft (((u^R_\otimes)^{-1}(\ulcorner f \urcorner \otimes (\eta \mathrm{Hom}(1, u_\triangleleft)))m)))\epsilon && \text{(lemma 2.8.1)} \\
&= (1 \triangleleft (((u^R_\otimes)^{-1}(\ulcorner f \urcorner \otimes (\eta \mathrm{Hom}(1, (1 \triangleleft id)\epsilon)))m)))\epsilon && \text{(def of } u_\triangleleft\text{)} \\
&= (1 \triangleleft (((u^R_\otimes)^{-1}(\ulcorner f \urcorner \otimes (id\eta \mathrm{Hom}(1, \epsilon)))m)))\epsilon && \text{(naturality of } \eta\text{)} \\
&= (1 \triangleleft (((u^R_\otimes)^{-1}(\ulcorner f \urcorner \otimes id)m)))\epsilon && \text{(triangle of identity)} \\
&= (1 \triangleleft (((u^R_\otimes)^{-1}(id \otimes \ulcorner f \urcorner)m)))\epsilon \\
&= (1 \triangleleft id\mathrm{Hom}(1, f))\epsilon \\
&= (1 \triangleleft id)\epsilon f && \text{(naturality of } \epsilon\text{)} \\
&= u_\triangleleft f
\end{aligned}$$

□

By proving Propositions 3.3.1 and 3.4.1, we prove the Theorem 3.1.1 and establish the equivalence between right $\mathbb{A}$-actegories with hom-objects and right $\mathbb{A}$-enriched categories with copowers. This result has not previously been proven in the general case where $\mathbb{A}$ is neither monoidal closed nor symmetric.

3.5 Powers

In the previous sections, we discussed copowers and the equivalence between right actegories with hom-objects and right enriched categories with copowers. In this section, we introduce the dual concept of copowers, known as powers, and obtain the dual of Theorem 3.1.1. Finally, we show that if the action functor in a right actegory with hom-objects admits a right adjoint, this adjoint gives a power, just like the action gives a copower.

3.5.1 Obtaining The Dual Theorem

In this section we establish the dual of Theorem 3.1.1 by first changing right actegory to left actegory, second changing right enrichment to left enrichment and transferring to $\mathbb{A}^{rev}$, and finally going to the opposite categories. This will give us an equivalence between having a left $\mathbb{A}^{op}$-actegory with hom-objects and a left-enriched category with powers. This processes can be shown as followes:

(i) To give a right $\mathbb{A}$-actegory with hom-objects that is

$$X \lhd - \dashv \mathrm{Hom}(X, -) : \mathbb{A} \to \mathbb{X}$$

is to give a right $\mathbb{A}$-enriched category with copowers, that is the follwoing bijective correspondence exists in $\mathbb{A}$:

$$\frac{B \xrightarrow{f} \mathrm{Hom}(X \lhd A, Y)}{A \otimes B \xrightarrow{(\eta \otimes f)m} \mathrm{Hom}(X, Y)}$$

(ii) To give a left $\mathbb{A}^{rev}$-actegory with hom-objects that is

$$- \rhd X \dashv \mathrm{Hom}(X, -) : \mathbb{A}^{rev} \to \mathbb{X}$$

is to give a left $\mathbb{A}^{rev}$-enriched category with copowers, that is the follwoing bijective correspondence exists in $\mathbb{A}^{rev}$:

$$\frac{B \xrightarrow{f} \mathrm{Hom}(A \triangleright X, Y)}{B \otimes A \xrightarrow{(f \otimes \eta)m} \mathrm{Hom}(X, Y)}$$

(iii) To give a left $(\mathbb{A}^{rev})^{op}$-actegory with hom-objects that is

$$\mathrm{Hom}^{op}(X, -) \dashv - \blacktriangleright X : (\mathbb{A}^{rev})^{op} \to \mathbb{X}^{op}$$

is to give a left $(\mathbb{A}^{rev})^{op}$-coenriched category with powers, that is the following bijective correspondence exists in $(\mathbb{A}^{rev})^{op}$:

$$\frac{B \xleftarrow{f} \mathrm{Hom}^{op}(A \blacktriangleright X, Y)}{B \otimes A \xleftarrow{(f^{op} \otimes \epsilon)m^{op}} \mathrm{Hom}^{op}(X, Y)}$$

(iv) To give a left $(\mathbb{A}^{rev})^{op}$-actegory with hom-objects that is

$$\mathrm{Hom}(-, X) \dashv - \blacktriangleright X : (\mathbb{A}^{rev})^{op} \to \mathbb{X}^{op}$$

is to give a left $\mathbb{A}^{rev}$-enriched category with powers, that is the following bijection exists in $\mathbb{A}^{rev}$:

$$\frac{B \xrightarrow{f} \mathrm{Hom}(Y, A \blacktriangleright X)}{B \otimes A \xrightarrow{(f \otimes \epsilon)m} \mathrm{Hom}(Y, X)}$$

So we can express the dual theorem as:

Theorem 3.5.1. *To give a left $(\mathbb{A}^{rev})^{op}$-actegory with hom-objects, is to give an $\mathbb{A}^{rev}$-enriched category with powers.*

We now show that in a right actegory with hom-objects, if the action functor has a right adjoint (in its other argument) the right adjoint action yields powers just as the action yields copowers. More importantly, in this case, both powers and copowers can have the same hom-object, demonstrating that they induce the same enrichment up to equivalence.

Theorem 3.5.2. *A right $\mathbb{A}$-actegory $\mathbb{X}$ with hom-objects is a left $(\mathbb{A}^{rev})^{op}$-actegory with hom-objects if and only if $\forall A \in \mathbb{A}$, the action functor has a right adjoint as $- \lhd A \dashv A \blacktriangleright - : \mathbb{X} \to \mathbb{X}$.*

Proof. We start with proving a right $\mathbb{A}$-actegory $\mathbb{X}$ with right hom-objects is a left $(\mathbb{A}^{rev})^{op}$-actegory with left hom-objects provided that the action functor has a right adjoint $(\eta', \epsilon') : - \lhd A \dashv A \blacktriangleright - : \mathbb{X} \to \mathbb{X}$. Since $\mathbb{X}$ is a right $\mathbb{A}$-actegory with hom-objects there is an adjunction $(\eta, \epsilon) : X \lhd - \dashv \mathrm{Hom}(X, -) : \mathbb{A} \to \mathbb{X}$. Using the two adjunctions we obtain the following bijective correspondence:

$$\dfrac{\dfrac{\dfrac{\mathrm{Hom}(X,Y) \xrightarrow{f^{op}} A \quad in\ (\mathbb{A}^{rev})^{op}}{A \xrightarrow{f} \mathrm{Hom}(X,Y) \quad in\ \mathbb{A}}}{X \lhd A \xrightarrow{(1 \lhd f)\epsilon} Y}}{X \xrightarrow{\eta'(1 \blacktriangleright ((1 \lhd f)\epsilon))} A \blacktriangleright Y}$$

Therefore, we can define the $(-)^\flat$ operator as $f^\flat = \eta'(1 \blacktriangleright ((1 \lhd f)\epsilon))$. It suffices to prove that $(\mathrm{Hom}(k,1) f^{op} h)^\flat = k f^\flat (h \blacktriangleright 1)$ which means $(h f \mathrm{Hom}(k,1))^\flat = k f^\flat (h \blacktriangleright 1)$. We

have:

$$\begin{aligned}
(hf\mathrm{Hom}(k,1))^\flat &= \eta'(1 \blacktriangleright ((1 \triangleleft hf\mathrm{Hom}(k,1))\epsilon)) \\
&= \eta'(h \blacktriangleright (1 \triangleleft f\mathrm{Hom}(k,1))\epsilon) \qquad \text{(dinaturality of } \eta', \text{ (Lemma 2.6.2))} \\
&= \eta'(h \blacktriangleright ((k \triangleleft f)\epsilon)) = k(\eta'(h \blacktriangleright ((1 \triangleleft f)\epsilon))) \\
&= k(\eta'(1 \blacktriangleright ((1 \triangleleft f)\epsilon)))(h \blacktriangleright 1) = kf^\flat(h \blacktriangleright 1)
\end{aligned}$$

Consequently, $\mathrm{Hom}(X,-) \dashv - \blacktriangleright X : \mathbb{X} \to (\mathbb{A}^{rev})^{op}$ which means that an $(\mathbb{A}^{rev})^{op}$-actegory has hom-objects. Now, we prove the other direction of the theorem. Using the two adjunctions $(\eta,\epsilon) : X \triangleleft - \dashv \mathrm{Hom}(X,-) : \mathbb{A} \to \mathbb{X}$ and $(\eta'',\epsilon'') : \mathrm{Hom}(X,-) \dashv - \blacktriangleright X : \mathbb{X} \to (\mathbb{A}^{rev})^{op}$ we have the following bijcective correspondence:

$$\dfrac{\dfrac{\dfrac{X \triangleleft A \xrightarrow{f} Y}{A \xrightarrow{\eta\mathrm{Hom}(1,f)} \mathrm{Hom}(X,Y) \quad in\ \mathbb{A}}}{\mathrm{Hom}(X,Y) \xrightarrow{\mathrm{Hom}(f,1)\eta} A \quad in\ (\mathbb{A}^{rev})^{op}}}{X \xrightarrow{\eta'(\mathrm{Hom}(f,1)\eta \blacktriangleright 1)} A \blacktriangleright Y}$$

So we can define the $(-)^\flat$ operator as $f^\flat = \eta'(\mathrm{Hom}(f,1)\eta \blacktriangleright 1)$. It suffices to show that $((h \triangleleft 1)fk)^\flat = hf^\flat(1 \blacktriangleright k)$. We have:

$$\begin{aligned}
hf^\flat(1 \blacktriangleright k) &= h\eta'(\mathrm{Hom}(f,1)\eta \blacktriangleright 1)(1 \blacktriangleright k) = h\eta'(1 \blacktriangleright k)(\mathrm{Hom}(f,1)\eta \blacktriangleright 1) \\
&= h\eta'(\mathrm{Hom}(kf,1)\eta \blacktriangleright 1) \qquad \text{(dinaturality of } \eta' \text{ (Corollary 2.6.2))} \\
&= \eta'(\mathrm{Hom}(kf,1)\mathrm{Hom}(1,h)\eta \blacktriangleright 1) \qquad \text{(naturality of } \eta') \\
&= \eta'(\mathrm{Hom}(kf(h \triangleleft 1),1)\eta \blacktriangleright 1) \qquad \text{(dinaturality of } \eta \text{ (Corollary 2.6.2))} \\
&= ((h \triangleleft 1)fk)^\flat
\end{aligned}$$

This proves that there is an adjunction $- \lhd A \dashv A \blacktriangleright - : \mathbb{X} \to \mathbb{X}$. □

Using Theorems 3.5.2 and 3.5.1, we can conclude that *a category is both powered and copowered if and only if the power and the copower form an adjoint pair.*

3.6 Right Actegory with Hom-Objects as a Linear/Non-Linear Adjunction

In this section, we delve deeper into the structure of actegories enriched with hom-objects. Specifically, we consider a strong monoidal $\mathbb{A}$-actegory with hom-objects in which the acting category $\mathbb{A}$ is cartesian. This gives rise to a linear/non-linear adjunction as introduced in Section 2.7.5.

Theorem 3.6.1. *A strong monoidal $\mathbb{A}$-actegory with hom-objects in which category $\mathbb{A}$ is cartesian, gives a (non-closed) linear/non-linear adjunction.*

Proof. In order for such a an actegory to be a linear/non-linear adjunction, we have to show that there is a monoidal adjunction between $\mathbb{A}$ and $\mathbb{X}$. In this setting, we have two functors $F := I \lhd - : \mathbb{A} \to \mathbb{X}$ and $G := \mathrm{Hom}(I, -) : \mathbb{X} \to \mathbb{A}$. We have an adjunction $F \dashv G : \mathbb{X} \to \mathbb{A}$ and we prove that this adjunction is monoidal. We define $n_I^F : I \to I \lhd 1$ as $n_I^F := u_{\lhd}^{-1}$ and $m_{\otimes}^F : (I \lhd A) \otimes (I \lhd B) \to I \lhd (A \times B)$ by the following composite:

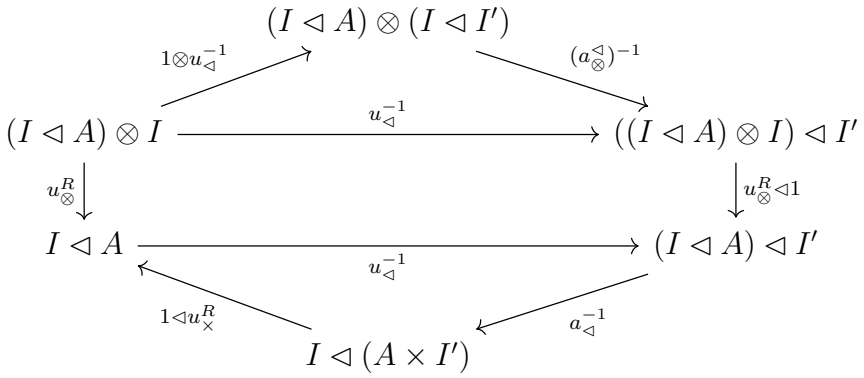
$$\begin{array}{ccc} (I \lhd A) \otimes (I \lhd B) & \xrightarrow{(a_{\otimes}^{\lhd})^{-1}} & ((I \lhd A) \otimes I) \lhd B \\ & \searrow^{m_{\otimes}^F} & \downarrow{\scriptstyle u_{\otimes}^R \lhd 1} \\ & & (I \lhd A) \lhd B \\ & & \downarrow{\scriptstyle a_{\lhd}^{-1}} \\ & & I \lhd (A \times B) \end{array}$$

It is clear that $m_{\otimes}^F$ is an isomorphism because it is a composition of isomorphisms. The map n_I^F is also clearly an isomorphism because $u_{\lhd}$ is an isomorphism. We have to

show that they satisfy the coherence conditions described in Section 2.7.3 which means we have to prove that $(m^F_\otimes \otimes 1)m^F_\otimes(1 \lhd a_\times) = a_\otimes(1 \otimes m^F_\otimes)m^F_\otimes$, $(n^F_I \otimes 1)m^F_\otimes(1 \lhd u^L_\times) = u^L_\otimes$ and $(1 \otimes n^F_I)m^F_\otimes(1 \lhd u^R_\times) = u^R_\otimes$. For the first one we have:

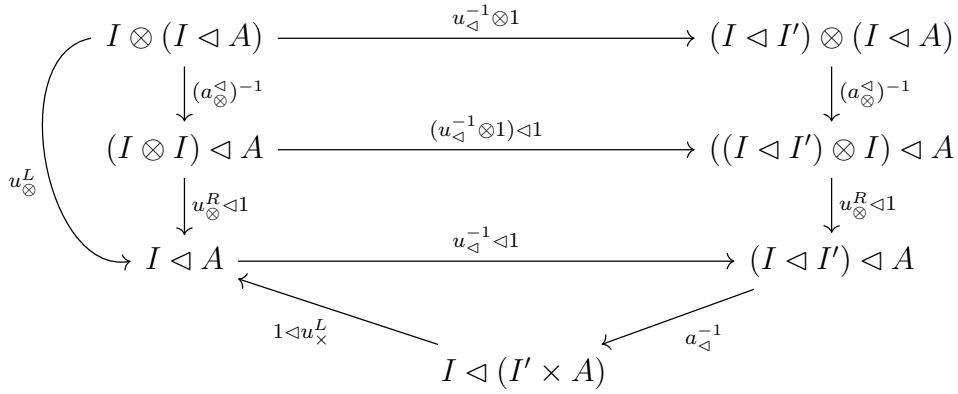
$$
\begin{aligned}
(m^F_\otimes \otimes 1)m^F_\otimes(1 \lhd a_\times) &= ((a^{\lhd}_\otimes)^{-1} \otimes 1)((u^R_\otimes \lhd 1) \otimes 1)(a^{-1}_\lhd \otimes 1)(a^{\lhd}_\otimes)^{-1}(u^R_\otimes \lhd 1)a^{-1}_\lhd(1 \lhd a_\times) \\
&= ((a^{\lhd}_\otimes)^{-1} \otimes 1)((u^R_\otimes \lhd 1) \otimes 1)(a^{\lhd}_\otimes)^{-1}((a^{-1}_\lhd \otimes 1) \lhd 1)(u^R_\otimes \lhd 1)a^{-1}_\lhd(1 \lhd a_\times) \\
&\qquad \text{(naturality of } (a^{\lhd}_\otimes)^{-1}\text{)} \\
&= ((a^{\lhd}_\otimes)^{-1} \otimes 1)(a^{\lhd}_\otimes)^{-1}(((u^R_\otimes \lhd 1) \otimes 1) \lhd 1)((a^{-1}_\lhd \otimes 1) \lhd 1)(u^R_\otimes \lhd 1)a^{-1}_\lhd(1 \lhd a_\times) \\
&\qquad \text{(naturality of } (a^{\lhd}_\otimes)^{-1}\text{)} \\
&= ((a^{\lhd}_\otimes)^{-1} \otimes 1)(a^{\lhd}_\otimes)^{-1}(((u^R_\otimes \lhd 1) \otimes 1) \lhd 1)(u^R_\otimes \lhd 1)(a^{-1}_\lhd \lhd 1)a^{-1}_\lhd(1 \lhd a_\times) \\
&\qquad \text{(naturality of } u^R_\otimes\text{)} \\
&= ((a^{\lhd}_\otimes)^{-1} \otimes 1)(a^{\lhd}_\otimes)^{-1}(((u^R_\otimes \lhd 1) \otimes 1) \lhd 1)(u^R_\otimes \lhd 1)a^{-1}_\lhd a^{-1}_\lhd \quad \text{(associativity of } a^{-1}_\lhd\text{)} \\
&= (a^{\lhd}_\otimes)^{-1}(((a^{\lhd}_\otimes)^{-1} \otimes 1) \lhd 1)(((u^R_\otimes \lhd 1) \otimes 1) \lhd 1)(u^R_\otimes \lhd 1)a^{-1}_\lhd a^{-1}_\lhd \\
&\qquad \text{(naturality of } (a^{\lhd}_\otimes)^{-1}\text{)} \\
&= (a^{\lhd}_\otimes)^{-1}(((a^{\lhd}_\otimes)^{-1} \otimes 1) \lhd 1)(u^R_\otimes \lhd 1)((u^R_\otimes \lhd 1) \lhd 1)a^{-1}_\lhd a^{-1}_\lhd \\
&\qquad \text{(naturality of } (a^{\lhd}_\otimes)^{-1}\text{)} \\
&= (a^{\lhd}_\otimes)^{-1}(((a^{\lhd}_\otimes)^{-1} \otimes 1) \lhd 1)(u^R_\otimes \lhd 1)a^{-1}_\lhd(u^R_\otimes \lhd 1)a^{-1}_\lhd \qquad \text{(naturality of } a^{-1}_\lhd\text{)} \\
&= (a^{\lhd}_\otimes)^{-1}(u^R_\otimes \lhd 1)((a^{\lhd}_\otimes)^{-1} \lhd 1)a^{-1}_\lhd(u^R_\otimes \lhd 1)a^{-1}_\lhd \qquad \text{(naturality of } u^R_\otimes\text{)} \\
&= (a^{\lhd}_\otimes)^{-1}(u^R_\otimes \lhd 1)((a^{\lhd}_\otimes)^{-1} \lhd 1)a^{-1}_\lhd(u^R_\otimes \lhd 1)a^{-1}_\lhd \qquad \text{(naturality of } u^R_\otimes\text{)} \\
&= (a^{\lhd}_\otimes)^{-1}(a_\otimes \lhd 1)((1 \otimes u^R_\otimes) \lhd 1)((a^{\lhd}_\otimes)^{-1} \lhd 1)a^{-1}_\lhd(u^R_\otimes \lhd 1)a^{-1}_\lhd \\
&\qquad (u^R_\otimes \otimes 1 = a_\otimes(1 \otimes u^R_\otimes)) \\
&= a_\otimes(1 \otimes (a^{\lhd}_\otimes)^{-1})(a^{\lhd}_\otimes)^{-1}((1 \otimes u^R_\otimes) \lhd 1)((a^{\lhd}_\otimes)^{-1} \lhd 1)a^{-1}_\lhd(u^R_\otimes \lhd 1)a^{-1}_\lhd \\
&\qquad \text{(associativity of } (a^{\lhd}_\otimes)^{-1}\text{)} \\
&= a_\otimes(1 \otimes (a^{\lhd}_\otimes)^{-1})(1 \otimes (u^R_\otimes \lhd 1))(a^{\lhd}_\otimes)^{-1}((a^{\lhd}_\otimes)^{-1} \lhd 1)a^{-1}_\lhd(u^R_\otimes \lhd 1)a^{-1}_\lhd \\
&\qquad \text{(naturality of } (a^{\lhd}_\otimes)^{-1}\text{)} \\
&= a_\otimes(1 \otimes (a^{\lhd}_\otimes)^{-1})(1 \otimes (u^R_\otimes \lhd 1))(1 \otimes a^{-1}_\lhd)(a^{\lhd}_\otimes)^{-1}(u^R_\otimes \lhd 1)a^{-1}_\lhd \qquad \text{(diagram 2.9.1)} \\
&= a_\otimes(1 \otimes m^F_\otimes)m^F_\otimes
\end{aligned}
$$

For the second one we have:



The top triangle commutes because of 2.9.1, the middle square commutes because of naturality of $u_{\lhd}$ and the bottom triangle commutes because of the coherence regarding $a_{\lhd}$ and $u_{\lhd}$ mentioned in Section 2.9.

And finally for the last one we have:



The top square commutes because of naturality of $a_{\otimes}^{\lhd}$, the bottom square commutes becuase $u_{\otimes}^{R}$ is natural, the bottom triangle commutes because of the coherence regarding $a_{\lhd}$ and $u_{\lhd}$ mentioned in Section 2.9.

This proves that $F := I \lhd -$ is a strong monoidal functor, which based on Proposition 2.7.2, we can conclude that $G := \mathrm{Hom}(I, -)$ is monoidal and the unit and counit of the adjunction are monoidal natural transformations, Therefore the adjunction is monoidal and we have a linear/non-linear adjunction.

□

This result shows that, based on Section 2.7.5, a strong monoidal $\mathbb{A}$-actegory $\mathbb{X}$ in which $\mathbb{A}$ is a cartesian category, is a linear category with $!X := I \lhd \mathrm{Hom}(I, X)$.

Chapter 4

Implementation of Higher-Order Processes in CaMPL

In the previous chapters, we explored the categorical foundations of higher-order processes in detail. In this chapter, we shift focus to the implementation of higher-order processes in the Categorical Message Passing Language (CaMPL).

To introduce the setting, we begin by presenting the syntax and semantics of CaMPL to establish a clear understanding of the language. We then explain how higher-order processes are introduced by integrating the store and use constructs, and describe the necessary modifications across the various stages of compilation.

4.1 Syntax and Semantics of the Categorical Message Passing Language (CaMPL)

Categorical Message Passing Language (CaMPL) is a functional-style concurrent programming language that implements concurrency through message passing.

The categorical semantics of CaMPL is modeled by a *linear actegory*, where a sym-

metric monoidal category $\mathbb{S}$—representing the semantics of the sequential side—acts on a symmetric linearly distributive category $\mathbb{C}$—representing the semantics of the concurrent side. This interaction is defined by two action functors:

$$\circ : \mathbb{S} \times \mathbb{C} \to \mathbb{C} \qquad \text{and} \qquad \bullet : \mathbb{S}^{\text{op}} \times \mathbb{C} \to \mathbb{C}.$$

These functors describe how sequential data, or *messages*, are transmitted through *channels*, which are modeled as concurrent types. The $\circ$ functor is the left parameterized left adjoint of $\bullet$ in the sense that

$$A \in \mathbb{A}, \quad A \circ \_ \dashv A \bullet \_ : \mathbb{C} \to \mathbb{C}$$

and this parameterized adjunction indicates that a process transmitting or receiving a message can do this equivalently on input or output polarity channels.

To provide an understanding of CaMPL programs, we first present the syntax and semantics of sequential data and constructs. We then discuss the syntax and semantics of the concurrent constructs. Finally, we explain how sequential messages are transmitted along concurrent channels by introducing the syntax and semantics of message passing.

4.1.1 Syntax and Semantics of the Sequential Side

The semantics of the sequential fragment of CaMPL is modeled by a cartesian category $\mathbb{S}$, where objects represent sequential types and morphisms represent sequential functions. The identity morphism corresponds to the identity function, and composition is given by function composition.

This category is equipped with both products and coproducts. Products are used to define `codata` by combining multiple types (e.g., records), while coproducts are used to define `data` by forming disjoint unions of types (e.g., the `Either` data type). CaMPL also

includes control-flow constructs such as `switch` and `if` expressions, which enable decision-making based on values and patterns. Further details on the syntax and semantics of sequential CaMPL can be found in [25] and [30].

4.1.2 Syntax and Semantics of the Concurrent Side

Semantics of the concurrent side of CaMPL is given by a linearly distributive category $\mathbb{C}$ with two monoidal structures $\otimes$ (tensor) and $\oplus$ (par). In this category, objects are concurrent channel types and morphisms are processes between the channels. The identity morphism is just a channel and composition of morphisms is given by plugging processes to each other so that they can communicate along a channel. In order to prevent deadlocks and livelocks, processes have to be plugged to each other along exactly one channel. When more than one channel is required for connecting two processes, the $\otimes$ and $\oplus$ functors are used to bundle the channels together and make a new channel that can be used for plugging the processes. As a result, they ensure valid arrangements of the concurrent processes to avoid any cycles.

A process with a sequential input of typ A, an input channel of type X, and an output channel of type Y can be defined in CaMPL using the syntax represented in Figure 4.1, it can be equivalently indicated as a circuit diagram, shown in 4.3 and the sequent calculus shown in Figure 4.2. In the circuit diagram in Figure 4.3, the black dot at the lower end of the input wire labeled a indicates that it is a sequential input used by the process.

```
proc p :: A | X => Y =
    a | x => y ->
        ...
```

Figure 4.1: Syntax for defining a process

$$A \mid X \Vdash Y$$

Figure 4.2: Sequent calculus

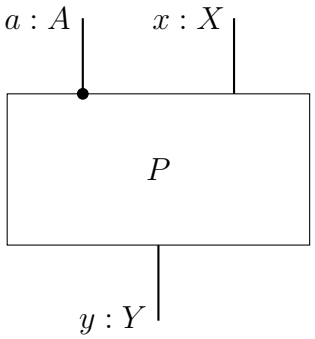


Figure 4.3: Circut diagram

Plug

Composition of two processes in the concurrent side of CaMPL is defined by the *plug* command which is the *cut* rule in linear logic. When two processes that are running in parallel are plugged to each other along a channel, they can communicate and pass messages through that channel in both directions. Two processes can only be plugged to each other along a channel, if they both have a channel of the same type but opposite polarity. One channel can be used to connect exactly two processes. Figure 4.4 shows an example of connecting two processes p and q is using programming syntax, circuit diagram and sequent calculus.

```
proc r :: A, B | X => Z =
    a, b | x => z ->
       plug
            p(a | x => y)
            q(b | y => z)
```

$$\frac{A \mid X \Vdash Y \quad B \mid Y \Vdash Z}{A, B \mid X \Vdash Z}\ \text{cut}$$

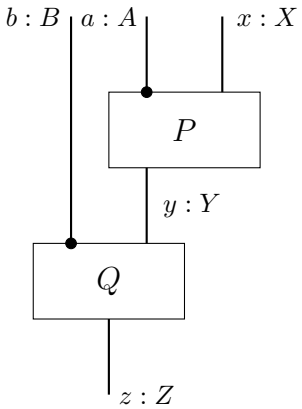


Figure 4.4: Composing two processes using `plug` command

Split and Fork

As mentioned earlier, two processes can be plugged to each other only by a single channel to prevent any cycles. However, there are some cases we want to plug processes with more than one channel. This is possible using $\otimes$ (tensor) and $\oplus$ (par) functors, that allow one to bundle more than one channels together and create a new channel that can be used for communication.

If a process has an output polarity channel of type $A \otimes B$ then, it can *fork* into two parallel processes with two distinct output channels of type A and B. Dually, the same thing happens if a process has an input channel of type $A \oplus B$. Examples of fork on output and input channels are indicated in Figure 4.5 and 4.6 respectively.

```
proc p :: A, B | X1, X2 => Y1 (*) Y2 =
    a, b | x1, x2 => y  ->
       fork y as
            y1 -> p(a | x1 => y1)
            y2 -> q(b | x2 => y2)
```

$$\frac{A \mid X1 \Vdash Y1 \quad B \mid X2 \Vdash Y2}{A, B \mid X1, X2 \Vdash Y1 \otimes Y2} \otimes_R$$

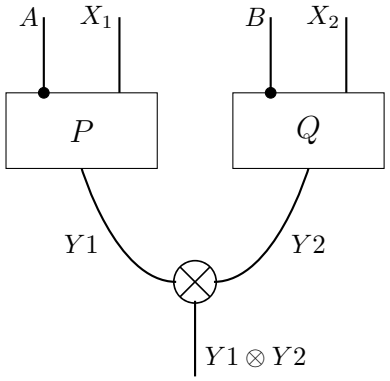


Figure 4.5: Forking an output channel

```
proc p :: A, B | X1 (+) X2 => Y1, Y2 =
    a, b | x => y1, y2  ->
       fork x as
            x1 -> p1(a | x1 => y1)
            x2 -> p2(b | x2 => y2)
```

$$\frac{A \mid X1 \Vdash Y1 \quad B \mid X2 \Vdash Y2}{A, B \mid X1 \oplus X2 \Vdash Y1, Y2} \oplus_L$$

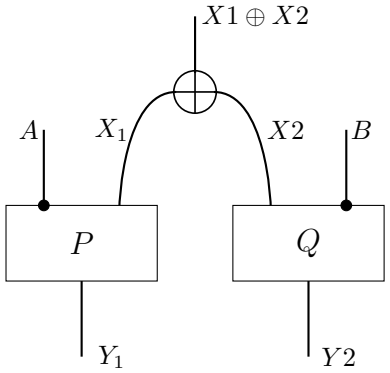


Figure 4.6: Forking an input channel

If a process has an input polarity channel of type $A \otimes B$, it can split this channel into to new channels of type A and B that can be used for further computations by the process. Dually, the same thing happens when a channel has an output channel of type $A \oplus B$. Examples of splitting input and output polarity channels are indicated in Figure 4.7 and Figure 4.8 respectively.

```
proc q :: A | X1 (*) X2 => Y =
    a | x => y -> do
       split x into x1, x2
       p(a | x1, x2 => y)
```

$$\frac{A \mid X1, X2 \Vdash Y}{A \mid X1 \otimes X2 \Vdash Y} \otimes_L$$

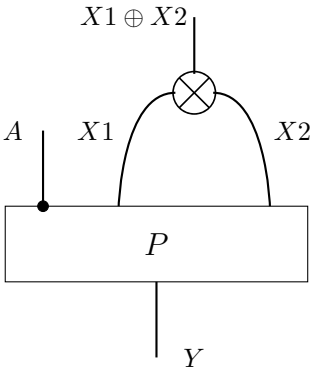


Figure 4.7: Splitting an input channel

```
proc q :: A | X => Y1 (+) Y2 =
    a | x => y -> do
       split y into y1, y2
       p(a | x => y1, y2)
```

$$\frac{A \mid X \Vdash Y1, Y2}{A \mid X \Vdash Y1 \oplus Y2} \oplus_R$$

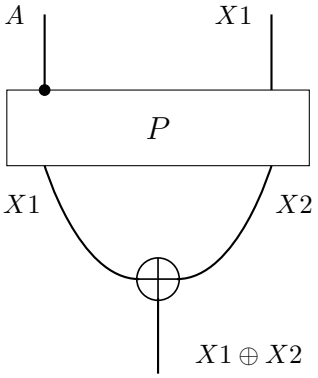


Figure 4.8: Splitting an output channel

4.1.3 Syntax and Semantics of Message Passing

The semantics of message passing in CaMPL is given by a linear actegory structure, in which the sequential category acts on the concurrent category. The functor $\circ$ corresponds to the `Put` operation, and the functor $\bullet$ corresponds to the `Get` operation in CaMPL. For

example, a channel of type $A \circ X$ indicates that a sequential message of type A can be placed on a channel of type X, which results in the protocol `Put(A | X)` in the language.

A key point in CaMPL is that channels are *polarized*, they can have either input or output polarity. However, this polarity does not restrict the direction of communication: a process can both send and receive data on channels of either polarity. This flexibility arises from the fact that the $\circ$ and $\bullet$ functors form an adjunction.

When two processes are connected via a channel, that channel must have input polarity for one process and output polarity for the other. Additionally, the type of the channel must match on both ends. For instance, if one process has an output polarity channel of type `Put(A | X)` and another process has an input polarity channel of the same type, they can be connected and communicate through that channel. Crucially, in this case, the process with the output-polarity side can send a value of type A, and the process with the input-polarity side can receive it, even though the channel type is `Put(A|X)`.

Examples illustrating data transmission along `Put` and `Get` channels are shown in Figures 4.9 through 4.12.

```
proc p' :: A, B | X => Put(A' | Y) =
  a, b | x => y -> do
    put f(a) on y
    p( b | x => y)
```

$$\frac{A \vdash A' \quad B \mid X \Vdash Y}{A, B \mid X \Vdash A' \circ Y} \circ_R$$

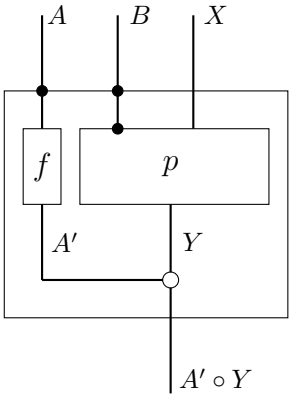


Figure 4.9: Sending on output channel

```
proc p' :: B | X => Get(A | Y) =
   b | x => y -> do
        get a on y
        p( a, b | x => y)
```

$$\frac{A, B \mid X \Vdash Y}{B \mid X \Vdash A \bullet Y} \bullet_R$$

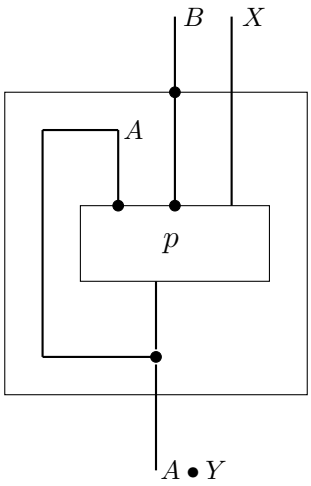


Figure 4.10: Receiving on output channel

```
proc p' :: A | Put(B | X) => Y =
  a | x => y -> do
    get b on x
    p(a, b | x => y)
```

$$\frac{A, B \mid X \Vdash Y}{A \mid B \circ X \Vdash Y} \circ_L$$

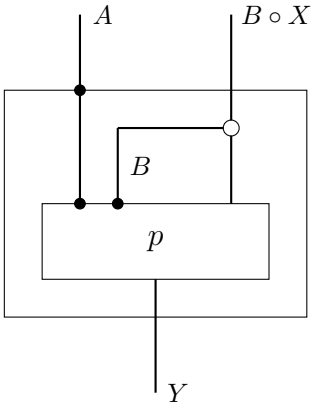


Figure 4.11: Receiving on input channel

```
proc p' :: A, B | Get(B' | X) => Y =
  a, b | x => y -> do
    put f(b) on x
    p(a | x => y)
```

$$\frac{B \vdash B' \quad A \mid X \Vdash Y}{A, B \mid B' \bullet X \Vdash Y} \bullet_R$$

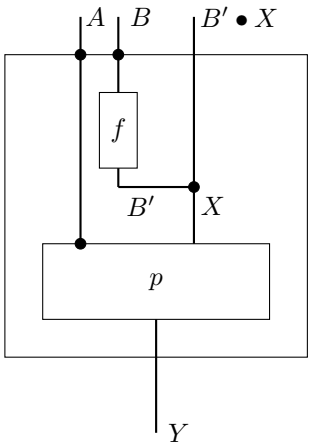


Figure 4.12: Sending on input channel.

4.2 Implementing Store and Use

The interpretation of a CaMPL program proceeds through several stages: lexing, parsing, type inference/type checking, compilation of pattern-matching, lambda lifting, and translation to abstract machine instructions. The resulting instructions are then executed on the CaMPL's abstract machine. These stages are illustrated in Figure 4.13.

In this section, we outline the stages of compilation and execution of a CaMPL program. For each stage, we provide a brief overview and describe the modifications required to support higher-order processes through the introduction of the store and use commands.

4.2.1 Parsing and α-Renaming

The grammar of CaMPL is extended to support a new expression, `store`, a new process command, `use`, and a new sequential type, `Store`:

- `store` function stores a process as sequential data. This function accepts either a process name or an inline (anonymous) process definition.
- `use` command unpacks a process so it can be run. The use command accepts any expression that evaluates to a stored process.
- `Store` type encapsulates a process type, which consists of a sequential type (for the sequential context) and two lists of concurrent types representing input and output channel types.

In a program, name clashes, arising from both local variables and local function definitions, can lead to significant semantic issues, particularly when definitions are lifted to the global scope. The initial step in program normalization is therefore to perform **α-renaming**, ensuring that all variable and function identifiers are globally unique. During

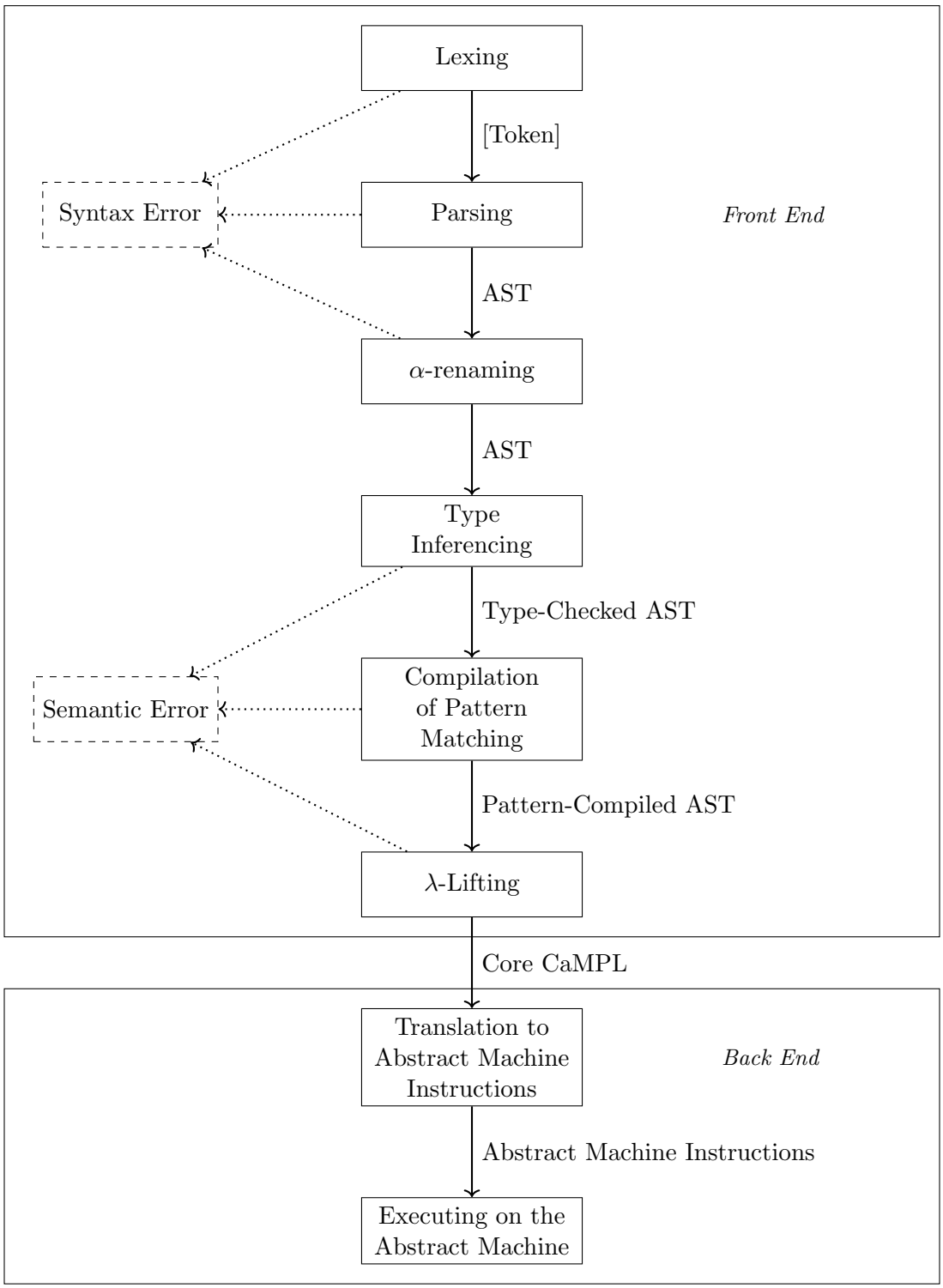


Figure 4.13: Interpretation Stages of a CaMPL Program

this phase, additional static checks should be performed: the presence of any free variables in the program should be reported as an error, and function definitions must be validated to ensure they are consistently applied with the correct arity. After a CaMPL program is parsed, it undergoes α-renaming. The introduction of store and use constructs does not alter this process; it only requires that α-renaming be correctly applied to the terms encapsulated within these commands.

4.2.2 Type Checking and Type Inference

Following α-renaming, the program undergoes type checking and type inference to detect potential type errors. Type checking and type inference is done using a collection of rules that form a type system. Typing rules are described using sequents of the form:

$$\Phi \vdash t :: T$$

Such a sequent relates a variable context Φ with a term t and its corresponding type T. If this relation can be derived using the rules of the type system, it constitutes a type judgment, indicating that there exists a proof assigning type T to term t under context Φ. The context Φ is a finite list of distinct bound variables along with their associated types. For example, $\Phi = x :: X, y :: Y$ denotes a context where variable x has type X and y has type Y. CaMPL's type system provides typing rules for both sequential and concurrent constructs. Sequential typing follows the form above, while concurrent typing judgments are expressed as

$$\Phi \mid \Gamma \Vdash \Delta$$

Here, Φ is the sequential context, Γ is a list of input channels, and Δ is a list of output channels annotated with their types. Complete type system of CaMPL can be found in [25].

$$\frac{p : \Phi, \Psi \mid \Gamma \Vdash \Delta}{\Phi \vdash \text{store}(p) : \text{Store}(\Psi \mid \Gamma \Vdash \Delta)}\ \text{store}$$

$$\frac{\Phi \vdash p : \text{Store}(\Psi \mid \Gamma \Vdash \Delta)}{\text{use}(p) : \Phi, \Psi \mid \Gamma \Vdash \Delta}\ \text{use}$$

Figure 4.14: Type Checking Rules for Store and Use

The typing rules introduced in CaMPL to support the store and use constructs are presented in Figure 4.14. The first rule states that if a process p has sequential contexts Φ and Ψ, input channels Γ, and output channels Δ, then the term `store(p)` has type $Store(\Psi|\Gamma \Vdash \Delta)$. Note that the choice of Φ and Ψ in this case is arbitrary. When p is defined inline, some parts of the sequential context can be treated as input arguments to p, while the rest can remain in the context of the parent process.

The second rule allows a stored process of type $Store(\Psi \mid \Gamma \Vdash \Delta)$ to be invoked using the `use` construct, yielding a process with type $\Phi, \Psi|\Gamma \Vdash \Delta$. This reflects the fact that the context Ψ required by the stored process is supplied at the point of use and combined with the current context Φ.

4.2.3 Compilation of Pattern Matching and Lambda Lifting

Once a CaMPL program has been type checked, the next step in its compilation is the conversion of its AST to Core CaMPL. This process occurs in two main stages: **Compilation of Pattern Matching** and **Lambda Lifting**. Compilation of pattern matching stage translates the pattern-matching syntax into CaMPL code without patterns. It marks the first step in converting the AST of CaMPL programs into Core CaMPL. In a CaMPL program, one can write local function definitions, whereas Core CaMPL does

not allow local definitions. The lambda lifting transformation eliminates local function definitions by lifting them to the global scope.

When the `store` and `use` constructs are introduced in CaMPL, processes can be stored using the `store` command. The process being stored can either be a globally declared process (referred to by its identifier) or an inline-defined process specified directly within the `store` command. In the latter case, the body of the inline-defined process may reference the sequential context available in the surrounding scope of the definition.

As part of the lambda lifting transformation, any locally defined process is lifted to the global scope. Since such processes may depend on local data, their required sequential context must be explicitly passed as arguments. This is handled during the lambda lifting phase.

Any process that invokes a stored process via the `use` command must have access to the sequential data required by that stored process in order to invoke it properly. However, a stored process—received as an input argument or message—has no information about what sequential context is needed. To address this, we replace every stored process with a tuple containing the required context and the name of the stored process. As a result, any process that receives a stored process knows it is receiving a tuple, where the first component of the tuple is the context and the second component is the process name. This allows the calling process to invoke the stored process with the appropriate context. Consequently, during λ-lifting, the code for handling the context must be generated to support stored processes.

If the stored process is a global process with no context requirements, we still ensure consistency by duplicating the process and adding an empty context tuple to it. The reason for duplication is that the original global process may still be called directly elsewhere in the program, and such direct calls must remain valid without requiring the additional context tuple.

The code generation process can be summarized as follows: First, we collect all global processes in the program that are referenced in a `store` command. These are added to a symbol table (implemented as a `Map` in Haskell), using the process name as the key.

We then traverse the function and process definitions again, performing the following transformations:

- When encountering a `store` command:
  - If the process is defined inline, we lift it to the global scope, assign it a fresh name, and extend its parameter list with the context from the enclosing scope.
  - If the process is referred to by name, we look it up in the symbol table. If found, we duplicate it, assign it a fresh name, and extend it with an empty context tuple.

  In both cases, we replace the argument of the `store` command with a tuple: the first component is the context, and the second is the unique name of the stored process.

- When encountering a `use` command:
  - We treat the expression passed to `use` as a tuple. We extract the process name using π_1 (the second projection) and use it as the process identifier to call.
  - We pass π_0 (the first projection), which contains the required context, as the argument to the process invocation.

An example of code generation is shown in Figure 4.15 and 4.16.

```
proc q =
    p | => ch -> use(p)( | => ch)

proc r =
    a1, a2 | => ch ->
        q( store ( | => ch -> do
                        put a1 on ch
                        put a2 on ch
                        halt ch
                ) | => ch
         )
```

Figure 4.15: Example of a program that written with store and use in CaMPL

```
proc q =
    p | => ch -> use(proj_1(p))(proj_0(p) | => ch)

proc r =
    a1, a2 | => ch ->
        q((context(a1, a2), store(stored_proc_r)) | => ch)

fun context =
    a, b -> (a, b)

proc stored_proc_r =
    ctx | => ch -> do
            put proj_0(ctx) on ch
            put proj_1(ctx) on ch
            halt ch
```

Figure 4.16: Translation of Figure 4.15

4.2.4 Abstract Machine

Abstract machines restructure program evaluation as a series of simple, low-level transitions—each designed to execute in nearly constant time. Evaluation begins by initializ-

ing the machine in a state that represents the input program, and proceeds step-by-step through these transitions until a final state is reached, from which the result is obtained. This systematic approach not only clarifies the operational semantics of the language but also enables the compilation of high-level constructs into sequences of machine instructions.

CaMPL's abstract machine, called AMPL, consists of two distinct components: one that executes sequential instructions and another that runs concurrent actions. This conceptual separation gives AMPL a modular design, allowing each component to be modified independently without affecting the other. In the following sections we go through these two components and explain how they work in detail.

Sequential AMPL

Sequential AMPL is based on the modern SECD machine [26], with several key extensions to support the compilation of both data and codata types—features not explicitly handled in the original SECD design. To enable this, the AMPL introduces specialized commands: `constructor` and `case` for compiling data types, and `record` and `destructor` for compiling codata types.

The core of CaMPL compilation involves translating high-level CaMPL programs into an intermediate representation called Core CaMPL. The back end then compiles this into a sequence of abstract machine instructions. Once the program is compiled to a list of instructions, it is evaluated using two additional structures: the **Stack** and the **Environment**. Both are Last-In-First-Out (LIFO) structures (implemented as Haskell list in AMPL). So the sequential abstract machine state is a triple

$$(S,\ E,\ C)$$

which consists of:

- S **(Stack)**: Stores intermediate values and ultimately the final result of the computation.
- E **(Environment)**: Maintains the current variable bindings and reflects the scope at each point in the program. It is a transient structure that evolves during evaluation.
- C **(Code)**: A list of instructions generated from the compiled CaMPL program.

Table 4.1 describes sequential instructions of AMPL and Table 4.2 describes the transition table for these sequential instructions. In the transition table, $\mathsf{clos}(c, e)$ denotes closure of code c with environment e, $e(n)$ is the n^{th}-element of the environment and $\mathsf{proc}(p)$ denotes the input and output channels along with the list of instructions of the process p.

Instruction	Explanation
Store	pushes the top stack element into the environment
Access(n)	put n^{th} value in the environment onto the stack.
Ret	return the top stack value and jump to the continuation below
Call code	jump to the code
StoreProc(p)	stores the process p, including its input and output channels and body's code (as a list of instructions), onto the environment (this instruction was added by the author)
Built in instructions:	
$\text{Const}_T(k)$	push the constant k of basic type T on the stack
Add	Pop two arguments from the top of the stack and add them
Mul	Pop two arguments from the top of the stack and add them
Leq	Pop two arguments from the top of the stack and compare them etc.
Data instructions:	
$\text{Cons}(i, n)$	push the i^{th} constructor onto the stack with arguments the top n elements of the stack, $\text{Cons}(i, s_1, ..., s_n)$.
$\text{Case}[c_1, ..., c_n]$	when $\text{Cons}(i, t_1, ..., t_n)$ is on the stack remove it and push $t_1, ..., t_n$ into the environment and evaluate c_i.
Codata instructions:	
$\text{Rec}[c_1, ..., c_n]$	create a record on the stack with current environment, $\text{rec}([c_1, ..., c_n], e)$
$\text{Dest}(i, n)$	destruct a record: choose the i^{th} function closure (c_i, e) and run c_i in environment e supplemented with the first n values on the stack.

Table 4.1: AMPL's sequential instructions

Before			After		
Code	Env	Stack	Code	Env	Stack
$\mathsf{Store}; c$	e	$v : s$	c	$v : e$	s
$\mathsf{Access}(n); c$	e	s	c	e	$e(n) : s$
$\mathsf{Call}(c) : c'$	e	s	c	e	$\mathsf{clos}(c', e) : s$
$\mathsf{Ret} : c$	e	$v : \mathsf{clos}(c', e') : s$	c'	e'	$v : s$
$\mathsf{StoreProc}(p) : c$	e	s	c	e	$\mathsf{proc}(p) : s$
$\mathsf{Cons}(i, n) : c$	e	$v_1 : ..., v_n : s$	c	e	$\mathsf{cons}(i, [v_1, .., v_n]) : s$
$\mathsf{Case}(c_1, ..., c_n) : c$	e	$\mathsf{Cons}(i, [v_1, ..., v_n]) : s$	c_i	$v_1 : .. : v_n : e$	$\mathsf{clo}(c, e) : s$
$\mathsf{Rec}(c_1, .., c_n) : c$	e	s	c	e	$\mathsf{rec}([c_1, ..., c_n], e) : s$
$\mathsf{Dest}(i, n) : c$	e	$\mathsf{rec}([c_1, .., c_n], e') : v_n : .. : v_1 : s$	c_i	$v_1 : .. : v_n : e'$	$\mathsf{clo}(c, e) : s$
$\mathsf{Const}_T(k) : c$	e	s	c	e	$\mathsf{const}_T(k) : s$
$\mathsf{Add} : c$	e	$n : m : s$	c	e	$(n + m) : s$
$\mathsf{Mul} : c$	e	$n : m : s$	c	e	$(n * m) : s$
$\mathsf{Leq} : c$	e	$n : m : s$	c	e	$(n \leq m) : s$
...					

Table 4.2: Transition table for sequential AMPL

Concurrent AMPL

The concurrent component of CaMPL's abstract machine extends the sequential model by incorporating a **translation** component which serves as a mapping from local channel names to global channel names. A channel connecting two processes may be referred to differently within each process, and the translation is used to resolve how these local channel names correspond to their shared global channels.

Concurrent AMPL consists of two main components: the **channel manager** $\mathcal{C}$ and the **process manager** $\mathcal{P}$, both of which are treated abstractly as sets (i.e., unordered collections).

The channel manager maintains a set of pairs consisting of a channel name and an associated communication queue. We write this as $\mathcal{C}\{(\alpha, q' \mid q)\}$ to indicate that the currently focused channel α has the communication queue $q' \mid q$, where q' is the *output* queue and q is the *input* queue. Each channel associated with a process has either input or output polarity. A process writes to the input queue of a channel if it is an input-polarity

channel, and to the output queue if it is an output-polarity channel. We denote the addition of a communication item x to the back of the input queue using the notation $q : x$, and we also use this pattern to indicate that x is the last item in the queue. Conversely, to add x to the front of the output queue, we write $x : q$. An empty queue is denoted by ϵ.

The process manager $\mathcal{P}$ is a set of processes. We write $\mathcal{P}\{(S, t, E, C)\}$ to indicate the currently selected process that is to take an execution step. In theory, the Process Manager selects a process to execute non-deterministically. In practice, however, it may be more efficient to allow the selected process to run for multiple steps, or continue execution until it reaches a concurrent action before interrupting it.

The channel manager not only holds the state of the communication queues but also manages execution effects arising from communication actions. For example, when a process attempts to `get` a value from a channel, it is suspended until another process performs a corresponding `put`. The order of these operations does not matter. Once a `put` has occurred (or if a value is already present), the suspended process can resume: the communication step "revives" the process and returns it to the process manager with the value it was waiting for.

Likewise, if a process needs to `fork` a channel, it requires a matching `split` action by the communicating process. The forking process is suspended and placed in the appropriate queue. Once a corresponding split action occurs, the communication step generates two new processes, each communicating on a new channel, and adds them to the process manager.

All communication actions occur in pairs—`put`/`get`, `split`/`fork`, `hput`/`hcase`, `close`/`halt`, etc.—reflecting the duality of protocol components in concurrent typing. Each action must occur in a specific polarity: either input or output. A protocol is *negative* if it results in suspension (i.e., waiting for a communication), and *positive* if

it initiates communication (i.e., deposits a datum). This polarity distinction becomes especially relevant in the context of races: only channels with negative types (i.e., those expecting input) may participate in races.

Concurrent Instructions

To understand how CAMPL operates, we begin by examining its commands. Table 4.3 presents the concurrent commands along with brief descriptions. The execution of CAMPL commands is governed by the Process Manager, which operates in coordination with the Channel Manager. The actions of the process manager and the channel manager are detailed in the following sections respectively.

get α; C	get a value on channel α (in the π-calculus $\alpha?[\_]$)
put α; C	put a value on channel α (in the π-calculus $\alpha![\_]$)
split α into $(\alpha_1, \alpha_2); C$	split channel α into two (new) channels
fork α as	
α_1 with $\Gamma_1 \to C_1$	forking on a channel into two distinct processes
α_2 with $\Gamma_2 \to C_2$	
plug $[\alpha_1, ..., \alpha_n]$	
$\Gamma_1 \to C_1$	two processes to communicate on n channels
$\Gamma_2 \to C_2$	
hput α $n; C$	put a "handle" on channel α
hcase$\{C_1, .., C_n\}$	the cases on receiving a "handle"
run t `process`	runs a process with local channel to caller channel translation t
close $\alpha; C$	closing a channel
halt α	halting process attached to a single channel
id $\alpha = \beta$	identifying channels
race	
$\alpha_1 \to C_1$	racing channels α and β
$\alpha_2 \to C_2$	
use t `process`	uses a stored process and runs it with local channel to caller channel translation

Table 4.3: Basic concurrent commands

Process Manager Actions

The actions managed by the process manager are described as follows:

Get/Put: These are the fundamental communication operations. The command `put` sends a value along a channel, while `get` receives a value from a channel. Executing a `put` places a value onto the input queue of the specified channel. Conversely, executing a `get` causes the process to suspend while waiting for a value to appear on the channel. Once a value is available, the channel manager transfers the value and resumes the suspended process. Typically, a `get` α is immediately followed by a `load` x to bind the received value in the sequential environment. Similarly, a `put` α is usually preceded by sequential code that computes the value to be sent.

Split: The `split` instruction directs the machine to divide a channel α into two new channels, α_1 and α_2. This is achieved by appending a notification to the communication queue indicating the two newly generated global channels, say β_1 and β_2, that replace α_1 and α_2. The local-to-global name mapping is updated with the translation $t[\beta_1/\alpha_1, \beta_2/\alpha_2]$, after which execution continues normally.

Fork: The dual of `split`, the `fork` instruction creates two concurrent processes that are expected to communicate via the channels generated by a prior `split`. However, since the `split` may not yet have occurred, the `fork` suspends the current process and attaches it to the relevant channel awaiting splitting. Once the corresponding `split` action is completed by the channel manager, the necessary translations are applied, and the two new processes are added to the process manager.

Plug: The `plug` command enables two processes to be connected via specified channels. The invoking process contains channels designated for external communication. These channels are divided between the two processes being connected. New global channels are then assigned to facilitate communication. After this setup, both processes are added

to the process manager.

Handle: The `hput` command sends a *handle*—a protocol constructor—which is matched by an `hcase` instruction in the receiving process. This mechanism resembles a `put`/`get` pair, but instead of simply passing a value, the receiver chooses the continuation based on the handle received.

Call: The `call` instruction invokes a predefined process. The key step here is setting up the mapping from the predefined process's local channels to global channel names before transferring control to its code.

Use: It expects a stored process (containing its input/output channels and instruction body) to be present on the top of the stack. It retrieves the stored process, creates mappings between its input/output channels and those expected by the calling context, establishes the necessary channel translations, and then jumps to the stored code (this instruction was added by the author).

Close/Halt: Executing `close` α removes the channel α from the system. The corresponding `halt` command terminates a process, but only after all other channels associated with the process have been closed. Thus, a `halt` α is valid only when α is the last open channel.

Id $\alpha = \beta$: This command sets up an identity mapping between channels α and β by informing the channel manager to treat their global translations as identical. This is commonly used for implementing wiring transformations, such as simulating negation in linear logic or defining identity processes.

Race: This command initiates a race condition between two or more channels. The system waits for input on any of these channels and triggers the corresponding continuation for the first one to receive data. The continuations for the losing channels are discarded.

$\mathcal{C}\{(t(\alpha), q' \mid q)\}$	$\mathcal{P}\{(s, t, e, \mathsf{get}\ \alpha; c)\}$	$\mathcal{C}\{(t(\alpha), q' \mid q{:}\mathsf{g}(s, t, e, c))\}$	$\mathcal{P}\{\}$
$\mathcal{C}\{(t(\alpha), q' \mid q)\}$	$\mathcal{P}\{(v{:}s, t, e, \mathsf{put}\ \alpha; c)\}$	$\mathcal{C}\{(t(\alpha), q' \mid q{:}v)\}$	$\mathcal{P}\{(s, t, e, c)\}$
$\mathcal{C}\{(t(\alpha), q' \mid q)\}$	$\mathcal{P}\{([], t, e, \begin{matrix}\mathsf{split}\ \alpha\ \mathsf{into} \\ (\alpha_1, \alpha_2); c\end{matrix})\}$	$\mathcal{C}\left\{\begin{pmatrix} t(\alpha), \\ q' \mid q{:}\beta_1, \beta_2 \end{pmatrix}\right\}$ (β_1, ε) (β_2, ε)	$\mathcal{P}\{([], t\begin{bmatrix}\beta_1/\alpha_1 \\ \beta_2/\alpha_2\end{bmatrix}, e, c)\}$
$\mathcal{C}\{(t(\alpha), q' \mid q)\}$	$\mathcal{P}\{(s, t, e, \mathsf{close}\ \alpha; c)\}$	$\mathcal{C}\{t(\alpha), q' \mid q{:}\mathsf{close})\}$	$\mathcal{P}\{(s, t\backslash\alpha, e)\}$
$\mathcal{C}\{(t(\alpha), q' \mid q)\}$	$\begin{matrix}\mathsf{fork}\ \alpha\ \mathsf{as} \\ \mathcal{P}\{([], t, e, \alpha_1\ \mathsf{with}\ \Gamma_1.c_1\ , [])\} \\ \alpha_2\ \mathsf{with}\ \Gamma_2.c_2\end{matrix}$	$\mathcal{C}\{\begin{pmatrix} t(\alpha), \\ q' \mid q{:}\begin{bmatrix} t, e, & \alpha_1/\Gamma_1.c_1 \\ & \alpha_2/\Gamma_2.c_2 \end{bmatrix}\end{pmatrix}\}$	$\mathcal{P}\{\}$
$\mathcal{C}\{(t(\alpha), q' \mid q)\}$	$\mathcal{P}\{([], t, e, \mathsf{id}\ \alpha = \gamma, [])\}$	$\mathcal{C}\{(t(\alpha), q' \mid q{:}\mathsf{Id}\ t(\alpha) = t(\beta))\}$	$\mathcal{P}\ \{\}$
$\mathcal{C}\{\}$	$\begin{matrix}\mathsf{plug}[\alpha_1, ..., \alpha_n] \\ \mathcal{P}\{([], t, e, \quad \Gamma_1 \to c_1 \quad\quad , [])\} \\ \Gamma_2 \to c_2\end{matrix}$	$\mathcal{C}\{(\gamma_i, \varepsilon \mid \varepsilon)_{i=1...n}\}$	$\mathcal{P}\left\{\begin{matrix}([], t[\gamma_i/\alpha_i]_{\Gamma_1}, e, c_1), \\ ([], t[\gamma_i/\beta_i]_{\Gamma_2}, e, c_2)\end{matrix}\right\}$
$\mathcal{C}\{(t(\alpha), q' \mid q)\}$	$\mathcal{P}\{([], t, e, \mathsf{halt}\ \alpha)\}$	$\mathcal{C}\{t(\alpha), q' \mid q{:}\mathsf{halt})\}$	$\mathcal{P}\{\}$
$\mathcal{C}\{\}$	$\mathcal{P}\{(s, t, e, \mathsf{run}\ t'\ c)\}$	$\mathcal{C}\{\}$	$\mathcal{P}\{(s, t; t', e', c)\}$
$\mathcal{C}\{(t(\alpha), q' \mid q)\}$	$\mathcal{P}\{(s, t, e, \mathsf{hput}\ \alpha\ n; c)\}$	$\mathcal{C}\{(t(\alpha), q' \mid q{:}\mathsf{h}(n))\}$	$\mathcal{P}\{(s, t, e, c)\}$
$\mathcal{C}\{(t(\beta), q' \mid q)\}$	$\mathcal{P}\{(s, t, e, \mathsf{hcase}\ \beta\ \{c_i\})\}$	$\mathcal{C}\{(t(\beta), q' \mid q{:}(s, t, e, \mathsf{hc}\{c_i\}))\}$	$\mathcal{P}\{\}$
$\mathcal{C}\left\{\begin{matrix} t(\alpha), q' \mid q), \\ t(\beta), r' \mid r)\end{matrix}\right\}$	$\mathcal{P}\left\{(s, t, e, \mathsf{race}\ \begin{matrix}\alpha \to c_1 \\ \beta \to c_2\end{matrix}\right\})$	$\mathcal{C}\left\{\begin{matrix}(t(\alpha), q' \mid \mathsf{r}([\beta], s, t, e, c_1){:}q), \\ (t(\beta), r' \mid \mathsf{r}([\alpha], s, t, e, c_2){:}r)\end{matrix}\right\}$	P{}
$\mathcal{C}\{\}$	$\mathcal{P}\{((\mathrm{proc(p)} : s, t, e, \mathsf{use}\ t')\}$	$\mathcal{C}\{\}$	$\mathcal{P}\{s, t; t', e', p)\}$

Table 4.4: Process execution steps (α with input polarity)

Channel Manager Actions

The channel manager is responsible for ensuring that communications are actually executed. When a process performs a communication action (such as sending a value), that action is first stored in the corresponding channel's queue, managed by the channel manager. Communication is completed only when a complementary action appears on the same channel. In each such pair of complementary actions, one typically causes the initiating process to suspend until the matching action becomes available (or resume immediately if it is already present).

An exception to this pattern is the `Id` action, which serves as the final instruction of a process. Rather than involving suspension, it merges two channels so that subsequent communication occurs directly on the unified channel.

Communication of Values or Handles: When a value or handle is waiting in a channel's queue, and a complementary process is suspended awaiting it, the value is transmitted and the waiting process is reactivated.

Split/Fork Communication: If a suspended process is waiting to perform a `fork`

and a corresponding `split` has already occurred on the same channel, then the newly introduced channels are assigned global names. The forked processes are then enabled, and their channel translations are updated accordingly.

Close/Halt Communication: A channel can be fully removed from the channel manager only when a `close` command on one side is matched with a `halt` command on the other.

Id $\alpha = \beta$: When the channel manager encounters an `Id` command at the head of a queue (which must always be the final command of the process), it merges channel α into channel β (or vice versa). This operation can proceed only if the complementary queue of the target channel is empty. To support this mechanism, the channel manager maintains a secondary translation structure to track global channel identifications and ensure they are applied correctly.

Race: A race is resolved when one of the competing channels receives an input—this channel is the winner. At that point, the race commands are removed from all participating channels, and the continuation code associated with the winning channel is executed.

$\mathcal{C}\{(\beta, q{:}v \mid \mathsf{g}(s,t,e,c))\}$	$\mathcal{P}$	$\mathcal{C}\{(\beta, q \mid \varepsilon)\}$	$\mathcal{P}\{(v{:}s,t,e,c)\}$
$\mathcal{C}\{(t(\beta), q{:}\mathsf{h}(i) \mid (s,t,e,\mathsf{hc}\ \{c_j\}))\}$	$\mathcal{P}$	$\mathcal{C}\{(\beta, q \mid \varepsilon)\}$	$\mathcal{P}\{(s,t,e,c_i)\}$
$\mathcal{C}\{(\beta,\beta_1,\beta_2 \mid \left[t,e,\begin{smallmatrix}\alpha_1/\Gamma_1.c_1\\ \alpha_2/\Gamma_2.c_2\end{smallmatrix}\right]\}$	$\mathcal{P}$	$\mathcal{C}\{\}$	$\mathcal{P}\left\{\begin{smallmatrix}([], t_{\Gamma_1}[\beta_1/\alpha_1], e, c_1, []),\\ ([], t_{\Gamma_2}[\beta_2/\alpha_2], e, c_2, [])\end{smallmatrix}\right\}$
$\mathcal{C}\{(\beta, \mathsf{close} \mid \mathsf{halt})\}$	$\mathcal{P}$	$\mathcal{C}\{\}$	$\mathcal{P}$
$\mathcal{C}\{(\beta, q \mid \mathsf{id}\ \beta = \alpha), (\alpha, \varepsilon \mid q')\}$	$\mathcal{P}$	$\mathcal{C}\{(\alpha, q \mid q')\}$	$\mathcal{P}\{\}$
$\mathcal{C}\left\{\begin{smallmatrix}(t(\alpha), q' : a \mid \mathsf{r}([\beta],s,t,e,c) : q),\\ t(\beta), r' \mid \mathsf{r}([\alpha],s,t,e,c') : r)\end{smallmatrix}\right\}$	P{}	$\mathcal{C}\left\{\begin{smallmatrix}(t(\alpha), q' : a \mid q),\\ (t(\beta), r' \mid r)\end{smallmatrix}\right\}$	$\mathcal{P}\{(s,t,e,c\}$

Table 4.5: Channel manager actions

Chapter 5

Conclusion and Future Work

In this thesis, we explored higher-order concurrency within the Categorical Message Passing Language (CaMPL) and proposed a method for its integration. Our approach began with a theoretical investigation into the foundations of higher-order processes, followed by an in-depth discussion of their implementation.

We began by reviewing essential categorical notions such as enrichment and actegories. We then introduced a definition of copowers in a setting that is neither closed nor symmetric. A key result we established is that giving a right actegory with hom-objects is equivalent to giving a right enriched category with copowers. We subsequently examined the dual theorem, which provides a definition of powers, and proved that a category is both powered and copowered if and only if the copower functor admits a left adjoint.

On the implementation side, we presented the syntax and semantics of CaMPL and detailed the modifications necessary to incorporate the `store` and `use` constructs. These modifications spanned all major components of the compiler pipeline, including parsing, α-renaming, type checking and inference, pattern matching compilation, λ-lifting, and the abstract machine.

A key application of higher-order processes, as discussed in this thesis, is in the im-

plementation of concurrent type classes. Type classes offer a principled mechanism for ad-hoc polymorphism in functional programming languages such as Haskell [29]. To implement type classes, one typically passes the appropriate method implementation (e.g., the `==` function in the `Eq` class) as an argument to the function that uses it [18]. This naturally requires higher-order functions. Analogously, in a concurrent setting, implementing type classes would require higher-order processes, underscoring the relevance of the theory and implementation developed in this work.

Bibliography

[1] Michael Barr. *-autonomous categories, revisited. *Journal of Pure and Applied Algebra*, 111(1-3):1–20, 1996.

[2] Michael Barr. **-Autonomous categories*, volume 752. Springer, 2006.

[3] Michael Barr and Charles Wells. *Category Theory for Computing Science*, volume 1. Prentice Hall New York, 1990.

[4] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, pages 121–135, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[5] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic*, pages 121–135, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[6] RF Blute, J. Robin B. Cockett, and Robert AG Seely. ! and?–storage as tensorial strength. *Mathematical Structures in Computer Science*, 6(4):313–351, 1996.

[7] Richard F Blute, J Robin B Cockett, Robert AG Seely, and Todd H Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996.

[8] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *International Conference on Concurrency Theory*, pages 222–236. Springer, 2010.

[9] Matteo Capucci and Bruno Gavranović. Actegories for the working amthematician. 03 2022.

[10] J.R.B. Cockett. Category theory for computer science. 2009.

[11] J.R.B. Cockett and R.A.G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997.

[12] Robin Cockett and Melika Norouzbeygi. Categorical semantics of higher-order message passing. *arXiv preprint arXiv:2503.19305*, 2025.

[13] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical logic*, 28(3):181–203, 1989.

[14] Samuel Eilenberg and G Max Kelly. Closed categories. In *Proceedings of the Conference on Categorical Algebra: La Jolla 1965*, pages 421–562. Springer, 1966.

[15] Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976.

[16] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

[17] R. Gordon and A.J. Power. Enrichment through variation. *Journal of Pure and Applied Algebra*, 120(2):167–185, 1997.

[18] Cordelia Hall, Kevin Hammond, Simon Peyton Jones, and Philip Wadler. Type classes in haskell. In *European Symposium On Programming*, pages 241–256. Springer, 1994.

[19] Craig Pastro J Robin B Cockett. Logic of message passing. *Science of Computer Programming*, 74, 06 2009.

[20] G. Janelidze and G. M. Kelly. A note on actions of a monoidal category. *Theory and Applications of Categories*, 9(4), 02 2001.

[21] Simon Peyton Jones, Andrew Gordon, and Sigbjorn Finne. Concurrent haskell. In *POPL*, volume 96, pages 295–308, 1996.

[22] André Joyal and Ross Street. The geometry of tensor calculus, i. *Advances in mathematics*, 88(1):55–112, 1991.

[23] G Max Kelly. Doctrinal adjunction. In *Category Seminar: Proceedings Sydney Category Theory Seminar 1972/1973*, pages 257–280. Springer, 2006.

[24] Gregory Maxwell Kelly. *Basic concepts of enriched category theory*, volume 64. CUP Archive, 1982.

[25] P. Kumar. Implementation of message passing language. Master's thesis, University of Calgary, Calgary, Canada, 2018.

[26] Peter J Landin. The mechanical evaluation of expressions. *The computer journal*, 6(4):308–320, 1964.

[27] Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 1998.

[28] Robert Paré. Mealy morphisms of enriched categories. *Applied Categorical Structures*, 20(3):251–273, 2012.

[29] John Peterson and Mark Jones. Implementing type classes. *ACM SIGPLAN Notices*, 28(6):227–236, 1993.

[30] Jared Pon. Redesigning the abstract machine for campl, 2021.

[31] John H Reppy. Cml: A higher concurrent language. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, pages 293–305, 1991.

[32] Bernardo Toninho, Luís Caires, and Frank Pfenning. Higher-order processes, functions, and sessions: A monadic integration. In *Programming Languages and Systems: 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings 22*, pages 350–369. Springer, 2013.