UNIVERSITY OF CALGARY

$\Sigma\Pi$-Polycategories, Additive Linear Logic, and Process Semantics

by

Craig Antonio Pastro

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

MARCH, 2004

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "ΣΠ-Polycategories, Additive Linear Logic, and Process Semantics" submitted by Craig Antonio Pastro in partial fulfillment of the requirements for the degree of Master of Science.

_____

Supervisor, Dr. James Robin B. Cockett
Department of Computer Science

_____

Dr. Robert J. Walker
Department of Computer Science

_____

Dr. Richard Zach
Department of Philosophy

_____

Date

Abstract

We present a process semantics for the purely additive fragment of linear logic in which formulas denote protocols and (equivalence classes of) proofs denote multi-channel concurrent processes. The polycategorical model induced by this process semantics is shown to be equivalent to the free polycategory based on the syntax (i.e., it is full and faithfully complete). This establishes that the additive fragment of linear logic provides a semantics of concurrent processes. Another property of this semantics is that it gives a canonical representation of proofs in additive linear logic.

Acknowledgements

Without the support, inspiration, and guidance of innumerable friends and colleagues this work would never have been accomplished. It gives me great pleasure to thank them all here.

First of all I would like to thank my supervisor, Professor Robin Cockett. His guidance, support, and encouragement has been invaluable for this thesis, as well as for myself. I consider myself extremely fortunate to have worked under his supervision.

A thank you (in alphabetical order) to Robin Cockett, Rob Walker, and Richard Zach for serving on my thesis committee.

To the wonderful staff in the Department of Computer Science at the University of Calgary. I am also grateful to the Department of Computer Science at the University of Calgary for financial support.

To my many mentors (academic or otherwise) who have helped me to find myself (or, indeed, to lose myself when that was necessary) — for their guidance and encouragement I will always be indebted.

My friends... what can I say really? You all mean more to me than I could write on a few lines here. I love you all. Cheers!

To both of my parents whose love and support (emotionally and financially) for me is absolute. To both of them my deepest and most heartfelt thanks.

This thesis was typeset using LaTeX with diagrams constructed using the XY-pic package of K. Rose and R. Moore and with inferences constructed using the `proof.sty` package of M. Tatsuya.

In memory of my Mother, Lydia Javier Gladman
1945 - 2000

Table of Contents

List of Tables

Chapter 0

Introduction

Since the introduction of linear logic by Girard [Gir87], people have believed that it should somehow provide a logic of concurrent communication. This intuition, however, has turned out to be rather difficult to substantiate in practice. In a seminal paper, Abramsky and Jagadeesan [AJ94] describe a game model for the multiplicative fragment of linear logic (MLL) with the MIX rule: formulas denote games and proofs denote winning strategies. As they point out, a game can be seen as a process; in fact, they provide a "dictionary" translating between the terminology of the two areas. The basic idea behind game semantics is to interpret a formula as a two-player game between "player" and "opponent", and proofs of formulas as "winning strategies" for the player. Using this idea, a proof can then be seen as an interaction between player and opponent. In terms of processes, one thinks of the player as the "system", the opponent as the "environment", and winning strategies for the player as "deadlock free processes". In this view, therefore, a proof can be seen as a process or system interacting with its environment.

Game semantics has turned out to be a remarkably effective tool for providing fully abstract semantics for programming languages. Its initial success was in providing the first syntax-free, fully abstract model for Scott's language PCF [Sco93], given independently by Abramsky, Jagadeesan, and Malacaria [AJM00], Hyland and Ong [HO00], and Nickau [Nic96]. Since then game semantics has provided fully abstract models for other programming languages with various other features, such

as richer type structures, different evaluation strategies, non-determinism, etc. (see, e.g., [AM98a, DH00, HM99, Lai98, McC96]).

As pointed out by Abramsky and Melliès [AM99], these types of game models do not, however, provide a model of concurrent communication. A play in these "sequential" games is a fixed interleaving of player and opponent moves. In [AM99], Abramsky and Melliès generalize games to "concurrent games", in which they abandon this interleaving of moves, and construct games in which both the player and opponent act in a distributed, asynchronous fashion. They announce in [AM99] that the concurrent games model is a "good" (i.e., fully complete; see below) model of multiplicative-additive linear logic.

In [AJ94], Abramsky and Jagadeesan introduce the notion of a fully complete model of a logic. Given a logic $\mathcal{L}$ and a categorical model of that logic $\mathcal{M}$, full completeness says that for any formulas A and B of the logic, any morphism $f : [\![A]\!] \to [\![B]\!]$ in the $\mathcal{M}$-interpretation of $\mathcal{L}$ is the interpretation of a proof of $A \vdash B$. One may also ask for a stronger notion: that any map in the model is the denotation of a unique proof. The term "full" is derived from category theory; that a model be fully complete is equivalent to requiring that the functor from the free category based on the syntax to the model is full. That any map in the model is the denotation of a unique proof is equivalent to requiring that this functor also be faithful. The construction of fully complete and full and faithfully complete models is of interest in its own right as it provides insights into the proofs of a logic. For a recent survey see [BHS03].

In this thesis we present a process semantics for the additive fragment of linear logic. As indicated above, this semantics could alternatively have been described as a game theoretic semantics, and indeed, many of the ideas and terminology are derived from this view. The games (processes) which we introduce are quite different from the Abramsky-Jagadeesan or Hyland-Ong style of games, and are perhaps more in the spirit of the money games of Joyal [Joy95c] and the games of Luigi Santocanale [San02b].

That the additive fragment of linear logic is particularly relevant to the perspective of proofs as processes began to be suspected when Joyal started his investiga-

tion of free bicompletions of categories [Joy95a, Joy95b] and their relationship to linear logic. Cockett and Seely, in their efforts to understand Joyal's work, then investigated the logic of finite sums and products [CS01] and realized that this was precisely the logic of communication along a single (two-way) channel. Their logic is a fragment of additive linear logic, restricted to exactly one formula on each side of the turnstile. By removing this restriction and allowing an arbitrary number of formulas on each side of the turnstile we get the logic used in this thesis, which we (also) call $\Sigma\Pi$. $\Sigma\Pi$ is a slightly untraditional presentation of the additive fragment of linear logic in that the sums and products are indexed by arbitrary finite sets, which is equivalent to binary sums and products with the nullary cases.

In this thesis we develop four equivalent views of concurrent processes: a process semantic view, a proof theoretic view, a term representation view, and a categorical view. The proof theoretic view is our starting point. We begin with the $\Sigma\Pi$ logic and consider proofs as processes. The process semantics is used to show that proofs in $\Sigma\Pi$ do, in fact, correspond to concurrent processes. This, of course, is what validates the claim that our other views are of concurrent processes. The term representation provides terms as processes. Since it is known that the terms correspond to concurrent processes, they can now be viewed as a programming language for these processes. In the categorical view, concurrent processes are represented by morphisms in a polycategory. The proof that all four views of concurrent processes are equivalent also uses categorical ideas.

A more detailed account is as follows. We begin with the logic $\Sigma\Pi$. Cut-elimination rewrites are presented for this logic, but in order for the cut-elimination rewrite system to be Church-Rosser, it is also necessary to give some equivalence rewrites for proofs which allows the interchange of rules: the so-called "permuting conversions".

Motivated by the interpretation of proofs as processes, two term representations for proofs in $\Sigma\Pi$ are developed: a term calculus representation and a "programming language" representation. The first step to view processes as terms is to view formulas as "protocols" assigned to a channel. For example, to view the formula $X+Y$ as a protocol it is assigned a channel, say α, and each subformula is assigned an

"event" resulting in $\alpha : (x : X + y : Y)$. Proofs may now be viewed as "multi-channel processes". If f is a proof the sequent $\alpha : X \vdash \beta : Z$ viewed as a process and g a proof of $\alpha : Y \vdash \beta : Z$ viewed as a process, then a proof of $\alpha : (x : X + y : Y) \vdash \beta : Z$ may be viewed as the process

$$\alpha \left\{ \begin{array}{l} x \mapsto f \\ y \mapsto g \end{array} \right\}$$

This process can be read as, "wait for an input event to occur on the channel α. If it is x then do the process f, and if it is y then it will do the process g."

The cut-elimination rewrites and permuting conversions may be represented using these terms. The notation "$f ;_\gamma g$" will be used to indicate cutting the proofs f and g together on the channel γ. If we interpret cut as communication, cut-elimination then is the dynamics of communication. As an example consider the terms

$$\overrightarrow{\gamma}[a](f) \qquad \text{and} \qquad \gamma \left\{ \begin{array}{l} a \mapsto f \\ b \mapsto g \end{array} \right\}$$

The first term is interpreted as, "output a on γ and then do the process f." The second term as, "wait for an input event on γ. If it is the event a then do the process g and if it is the event b then do the process h." If we tell these two processes to communicate on γ it should be obvious that after one step of communication they will evolve to the process f communicating with the process g on γ, i.e.,

$$\overrightarrow{\gamma}[a](f) \;;_\gamma \gamma \left\{ \begin{array}{l} a \mapsto f \\ b \mapsto g \end{array} \right\} \quad \Longrightarrow \quad f \;;_\gamma g$$

These processes look very sequential in nature, but the permuting conversions are hiding the fact that these are concurrent processes!

As terms are much easier to manipulate than proofs, they are used to show that the rewriting system induced by the cut-elimination rewrites is Church-Rosser and terminating modulo the equivalences. A decision procedure is presented which allows us to determine if two (cut-free) terms of the same type (derivations of the same sequent) are equivalent, i.e., that they are related by the equivalence rewrites.

A polycategory of processes may be formed by considering protocols as objects,

multi-channel processes (the terms) as morphisms, and communication as composition. It is shown that this is the free polycategory with finite sums and finite products. Following Joyal's and Cockett and Seely's lead, we prove a "Whitman theorem" which gives a characterization of the free polycategory with finite sums and products over a base polycategory in terms of characterizations of the hom sets.

Finally, a process semantics for $\Sigma\Pi$-terms is presented. This semantics illustrates precisely how the terms correspond to channel-based concurrent processes. The rough idea is as follows. We define the notion of a behaviour, which is an explicit representation of the history of a process. These are typically represented using tables, e.g.,

			b
d	e		$\overline{c}$
$\overline{c}$	a	$\overline{d}$	a
α	β	γ	δ

This behaviour represents outputting c and then receiving d on channel α, receiving a and then e on β, and so on. From behaviours there is no way to tell the order that the events occurred, however, this is necessary. To see this consider the process:

$$\alpha \left\{ \begin{array}{l} a \mapsto \overrightarrow{\beta}\,[c](f) \\ b \mapsto \overrightarrow{\beta}\,[d](g) \end{array} \right\}$$

It is easily seen that this process is unable to produce an output on β until it has first received an event on α. That is, there is a hierarchy on events. This motivates the use of "entailments". An entailment is a behaviour with a distinguished output event. If all the events in the behaviour have occurred then the output event may be performed. The following set of entailments

$$\left\{ \begin{array}{|c||c|}\hline a & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[c], \quad \begin{array}{|c||c|}\hline & b \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[d], \quad \begin{array}{|c||c|}\hline a & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash f, \quad \begin{array}{|c||c|}\hline b & \overline{d} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash g \right\}$$

encodes the hierarchy on events of the process above. Sets of entailments (satisfying the seven rules defined in Chapter 5) are what we call "extensional processes". These

are what gives us our interpretation of proofs as concurrent processes.

A polycategory of extensional processes is constructed and shown to be equivalent to the polycategory of processes. This proves that this model satisfies the property that every process is the denotation of a unique cut-free proof, i.e., it is a full and faithfully complete model.

Some of the previous results are essentially an extension from the (ordinary) categorical case to the polycategorical case of the results of Cockett and Seely [CS01]. On occasion I have used the exposition found therein when it is much clearer than I could hope to achieve. Errors in these sections, as in the rest of the text, are of course solely my responsibility.

Outline of this thesis This thesis is organized as follows: Chapter 1 introduces some standard concepts of category theory and the notion of a polycategory. The concepts here will be needed throughout the thesis. Chapter 2 presents the $\Sigma\Pi$-logic. In Chapter 3 we develop two term representations for derivations in this logic: a term calculus and a "programming language" representation. A rewrite system for cut-elimination is presented and shown to be Church-Rosser and terminating. The proof of decidability for these terms is also presented in this chapter. Chapter 4 is devoted to proving that $\Sigma\Pi_{\mathbf{A}}$ is the free polycategory built over an arbitrary polycategory $\mathbf{A}$. In Chapter 5 we present a process semantics for our logic. This semantics is shown to provide a full and faithfully complete polycategorical model for our logic.

Contributions of this thesis In Chapter 2 the description of additive linear logic is a new presentation of this fragment. Chapter 3 contains two term representations which are essentially new to this thesis, as is the presentation of cut-elimination and the decision procedure for terms. Chapter 4 contains a new polycategorical presentation of additive linear logic and the development of the Whitman theorem for this setting. Chapter 5 contains the description of extensional processes, which is the main novel aspect of this thesis.

Chapter 1

Categorical Preliminaries

This chapter is meant to give a brief introduction to category theory and also to help accustom the reader with the notation that will be used throughout this thesis. For a more complete introduction to category theory see, e.g., [ML98] or [BW99].

1.1 Categories

A **category C** consists of a class of objects, $\mathrm{ob}(\mathbf{C})$ (or $\mathbf{C}_0$), for each pair of objects A and B, a class of morphisms (or arrows), $\mathbf{C}(A, B)$, for each object A, an identity morphism $\mathrm{id}_A \in \mathbf{C}(A, A)$, and for each triple of objects, A, B, and C, a composition law

$$\mathbf{C}(A, B) \times \mathbf{C}(B, C) \xrightarrow{;} \mathbf{C}(A, C)$$

satisfying:

- if $f \in \mathbf{C}(A, B)$ then the domain (or source) of f is $\mathrm{dom}(f) = A$ and the codomain (or target) of f is $\mathrm{cod}(f) = B$.
- if $f : A \to B$ then $\mathrm{id}_A; f = f; \mathrm{id}_B = f$.
- $f; (g; h) = (f; g); h$ whenever either side is defined, i.e., composition is associative.

When it is clear from the context we will omit the label $\mathrm{ob}(\mathbf{C})$ and simply refer to an object $A \in \mathrm{ob}(\mathbf{C})$ as $A \in \mathbf{C}$. We will sometimes write $\mathrm{Hom}(A, B)$ or just (A, B) to mean $\mathbf{C}(A, B)$ when this will not lead to confusion. A morphism $f \in \mathbf{C}(A, B)$ may also be written as $f : A \to B$ or $A \xrightarrow{f} B$. For any composible pair of arrows $f : A \to B$ and $g : B \to C$ we will sometimes denote their composition in the usual manner with a $\circ$ as $g \circ f$ (as opposed to the diagrammatic order $f; g$ that we typically use). We often omit the $\circ$ and simply write gf but we will never omit the ; .

Note that when we defined the notion of a category we did not say that the collection of objects constitute a set. Indeed, in the most famous of categories, **Set**, where the objects are sets and the morphisms are functions between sets, the collection of all sets is not itself a set. This motivates the following definitions: a category is called **small** if its objects constitute a set, and **large** otherwise. If $\mathbf{C}(A, B)$ is a set for all objects $A, B \in \mathbf{C}$ then $\mathbf{C}$ is called **locally small**.

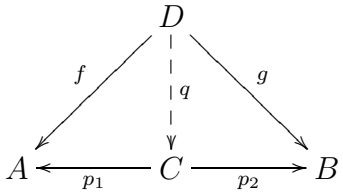
If $\mathbf{C}$ is a category, then its **dual** $\mathbf{C}^{\mathrm{op}}$, is defined by $\mathrm{ob}(\mathbf{C}^{\mathrm{op}}) = \mathrm{ob}(\mathbf{C})$ and $\mathbf{C}^{\mathrm{op}}(A, B) = \mathbf{C}(B, A)$. That is, the dual of a category is the category with all its arrows reversed: if $f : A \to B$ is an arrow in $\mathbf{C}$ then $f : B \to A$ is an arrow in $\mathbf{C}^{\mathrm{op}}$. It is clear that the dual of a category is also a category.

1.2 Properties of morphisms

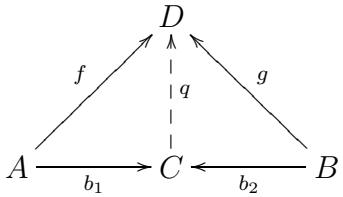
An arrow $f : A \to B$ in a category $\mathbf{C}$ is called **monic** (or a **monomorphism**) if for any object $C \in \mathbf{C}$ and arrows $x, y : C \to A$ such that $x; f = y; f$ then $x = y$. The dual of a monomorphism is an **epic** (or **epimorphism**), i.e., an arrow $f : A \to B$ is epic if for any object C and arrows $x, y : B \to C$ such that $f; x = f; y$ then $x = y$. An arrow $f : A \to B$ is called an **isomorphism** if there is an arrow $g : B \to A$ such that $f; g = \mathrm{id}_A$ and $g; f = \mathrm{id}_B$. If f is an isomorphism then the arrow g is uniquely determined and is usually called the inverse of f. If such an isomorphism exists, we say that A is isomorphic to B and denote this as $A \cong B$.

1.3 Sums and products

If A and B are objects in a category $\mathbf{C}$ then by the **product** of A and B we mean an object C together with arrows $p_1 : C \to A$ and $p_2 : C \to B$ (called the first and second projections respectively) such that for any object D and arrows $f : D \to A$ and $g : D \to B$ there is a unique arrow $q : D \to C$ making the following diagram commute:



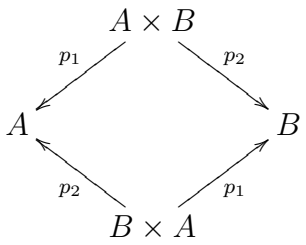
The dual of a product is a **sum** (or **coproduct**). That is, if A and B are objects in a category $\mathbf{C}$ then the coproduct of A and B is an object C together with arrows $b_1 : A \to C$ and $b_2 : B \to C$ (called the first and second injections respectively) such that for any object D and arrows $f : A \to D$ and $g : B \to D$ there is a unique arrow $q : C \to D$ making the following diagram commute:



The product of A and B is typically denoted as $A \times B$ and the unique arrow from D to $A \times B$ as $\langle f, g \rangle$. The coproduct of A and B is typically denoted as $A + B$ and the unique arrow from $A + B$ to D as $\langle f | g \rangle$.

For products (and dually for sums), it can be shown that $A \times B$ is isomorphic

to $B \times A$. To see this consider the diagram



and the unique arrows from the definition that make this diagram commute.

1.4 Equalizers and coequalizers

Let $\mathbf{C}$ be a category and $f, g : A \rightrightarrows B$ be a parallel pair of arrows. An **equalizer** of f and g is an object E together with an arrow e such that

(i) $e; f = e; g$, and

(ii) for any object Q and arrow $q : Q \to A$ such that $q; f = q; g$, there is a unique arrow $h : Q \to E$ such that $h; e = q$.

Dually, a **coequalizer** of $f, g : A \rightrightarrows B$ is an object C together with an arrow $c : B \to C$ such that

(i) $f; c = g; c$, and

(ii) for any object Q and arrow $q : B \to Q$ such that $f; q = g; q$, there is a unique arrow $h : C \to Q$ such that $c; h = q$.

1.5 Functors

Let $\mathbf{C}$ and $\mathbf{D}$ be categories. A **functor** $F : \mathbf{C} \to \mathbf{D}$ between categories is a pair of maps $F_0 : \mathbf{C}_0 \to \mathbf{D}_0$ and $F_1 : \mathbf{C}(A, B) \to \mathbf{D}(F_0(A), F_0(B))$, for all objects $A, B \in \mathbf{C}$, satisfying

- $F_1(\mathrm{id}_A) = \mathrm{id}_{F_0(A)}$

- $F_1(f;g) = F_1(f);F_1(g)$

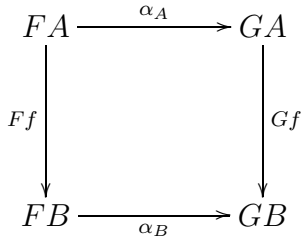
It is standard practice to omit the subscripts from the functor when the context is clear. The brackets may also be omitted when they are not required to disambiguate precedence. Thus, we will sometimes write FA and Ff to mean $F_0(A)$ and $F_1(f)$ respectively.

For any category $\mathbf{C}$ there is an identity functor, $\mathrm{id}_{\mathbf{C}} : \mathbf{C} \to \mathbf{C}$, defined in the obvious way.

Two categories $\mathbf{C}$ and $\mathbf{D}$ are said to be **isomorphic**, denoted $\mathbf{C} \cong \mathbf{D}$, if there are a pair of functors $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ such that $F;G = \mathrm{id}_{\mathbf{C}}$ and $G;F = \mathrm{id}_{\mathbf{D}}$.

1.6 Natural transformations

Given two functors $F, G : \mathbf{C} \to \mathbf{D}$, a **natural transformation** $\alpha : F \Rightarrow G$ consists of a family of morphisms $\alpha_A : FA \Rightarrow GA$, one for each object $A \in \mathbf{C}$, such that for any morphism $f : A \to B$ in $\mathbf{C}$ the following diagram commutes.



We call the α_A the **component** of the natural transformation at A.

For any functor $F : \mathbf{C} \to \mathbf{D}$ there is an identity natural transformation $\mathrm{id}_F : F \Rightarrow F$ defined by $(\mathrm{id}_F)_A = \mathrm{id}_{FA}$.

A natural transformation $\alpha : F \Rightarrow G$ is called a **natural isomorphism** if every component α_A is invertible in $\mathbf{D}$. In this case we say that F and G are isomorphic and write $F \cong G$.

Two categories $\mathbf{C}$ and $\mathbf{D}$ are said to be **equivalent** if there are functors $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ such that $F;G \cong \mathrm{id}_{\mathbf{C}}$ and $G;F \cong \mathrm{id}_{\mathbf{D}}$.

1.7 Polycategories

A **planar polycategory P** consists of the following data

- a class $\mathbf{P}_0$ of objects of $\mathbf{P}$,
- for each $m, n \in \mathbb{N}$ and $x_1, \ldots, x_m, y_1, \ldots, y_n \in \mathbf{P}_0$, a set

$$\mathbf{P}(x_1, \ldots, x_m \ ; \ y_1, \ldots, y_n)$$

  whose elements are called polymorphisms. Using Γ and Δ to represent strings of elements of $\mathbf{P}_0$, the polymorphisms in $\mathbf{P}(\Gamma \ ; \ \Delta)$ may be denoted $f : \Gamma \to \Delta$ or $\Gamma \vdash_f \Delta$ where $\mathrm{dom}(f) = \Gamma$ and $\mathrm{cod}(f) = \Delta$.

together with

- for each $x \in \mathbf{P}_0$, an identity morphism $1_x \in \mathbf{P}(x, x)$
- an operation

$$\mathbf{P}(\Gamma; \Delta_1, x, \Delta_2) \times \mathbf{P}(\Gamma_1, x, \Gamma_2; \Delta) \to \mathbf{P}(\Gamma_1, \Gamma, \Gamma_2; \Delta_1, \Delta, \Delta_2)$$

  called **cut**, restricted to the cases where either Γ_1 or Δ_1 is empty and either Γ_2 or Δ_2 is empty. (This restriction is called the **crossing** (or **planarity condition**), cf. circuit diagrams below.) Explicitly, this gives four cut rules:

  - $\mathbf{P}(\Gamma_1; \Delta_1, x) \times \mathbf{P}(x, \Gamma_2; \Delta_2) \xrightarrow{\mathrm{cut}_r} \mathbf{P}(\Gamma_1, \Gamma_2; \Delta_1, \Delta_2)$
  - $\mathbf{P}(\Gamma_1; x, \Delta_1) \times \mathbf{P}(\Gamma_2, x; \Delta_2) \xrightarrow{\mathrm{cut}_l} \mathbf{P}(\Gamma_2, \Gamma_1; \Delta_2, \Delta_1)$
  - $\mathbf{P}(\Gamma; x) \times \mathbf{P}(\Gamma_1, x, \Gamma_2; \Delta) \xrightarrow{\mathrm{mcut}} \mathbf{P}(\Gamma_1, \Gamma, \Gamma_2; \Delta)$
  - $\mathbf{P}(\Gamma; \Delta_1, x, \Delta_2) \times \mathbf{P}(x; \Delta) \xrightarrow{\mathrm{mcut}^{\mathrm{op}}} \mathbf{P}(\Gamma; \Delta_1, \Delta, \Delta_2)$

These data are subject to three axioms:

- cut has identities, i.e., the following diagrams commute

$$\begin{array}{ccc} \mathbf{P}(\Gamma_1, x, \Gamma_2; \Delta) & \xrightarrow{1_x \times -} & \mathbf{P}(x,x) \times \mathbf{P}(\Gamma_1, x, \Gamma_2; \Delta) \\ & \searrow & \downarrow \mathrm{cut} \\ & & \mathbf{P}(\Gamma_1, x, \Gamma_2; \Delta) \end{array}$$

$$\begin{array}{ccc} \mathbf{P}(\Gamma; \Delta_1, y, \Delta_2) & \xrightarrow{- \times 1_y} & \mathbf{P}(\Gamma; \Delta_1, y, \Delta_2) \times \mathbf{P}(y,y) \\ & \searrow & \downarrow \mathrm{cut} \\ & & \mathbf{P}(\Gamma; \Delta_1, y, \Delta_2) \end{array}$$

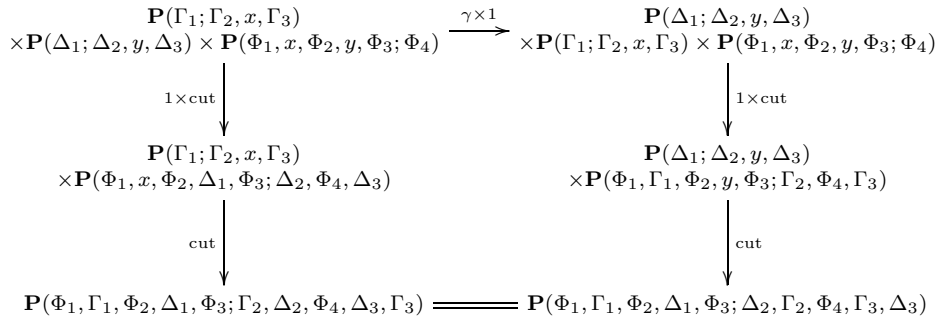
- cut is associative, i.e., the following diagram commutes

$$\begin{array}{ccc} \begin{array}{c}\mathbf{P}(\Gamma_1; \Gamma_2, x, \Gamma_3) \\ \times \mathbf{P}(\Delta_1, x, \Delta_2; \Delta_3, y, \Delta_4) \\ \times \mathbf{P}(\Phi_1, y, \Phi_2; \Phi_3)\end{array} & \xrightarrow{1 \times \mathrm{cut}} & \begin{array}{c}\mathbf{P}(\Gamma_1; \Gamma_2, x, \Gamma_3) \\ \times \mathbf{P}(\Phi_1, \Delta_1, x, \Delta_2, \Phi_2; \Delta_3, \Phi_3, \Delta_4)\end{array} \\ \downarrow \mathrm{cut} \times 1 & & \downarrow \mathrm{cut} \\ \begin{array}{c}\mathbf{P}(\Delta_1, \Gamma_1, \Delta_2; \Gamma_2, \Delta_3, x, \Delta_4, \Gamma_3) \\ \times \mathbf{P}(\Phi_1, y, \Phi_2; \Phi_3)\end{array} & \xrightarrow{\mathrm{cut}} & \mathbf{P}(\Phi_1, \Delta_1, \Gamma_1, \Delta_2, \Phi_2; \Gamma_2, \Delta_3, \Phi_3, \Delta_4, \Gamma_3) \end{array}$$

Recall that cut is subject to the crossing condition; writing this restriction explicitly would result in nine separate commutative diagrams.

- cut satisfies the interchange law (originally referred to as "commutativity" by Lambek [Lam69]), i.e., the following diagrams commute

$$\begin{array}{ccc} \begin{array}{c}\mathbf{P}(\Gamma_1; \Gamma_2, x, \Gamma_3, y, \Gamma_4) \times \mathbf{P}(\Delta_1, x, \Delta_2; \Delta_3) \\ \times \mathbf{P}(\Phi_1, y, \Phi_2; \Phi_3)\end{array} & \xrightarrow{1 \times \gamma} & \begin{array}{c}\mathbf{P}(\Gamma_1; \Gamma_2, x, \Gamma_3, y, \Gamma_4) \times \mathbf{P}(\Phi_1, y, \Phi_2; \Phi_3) \\ \times \mathbf{P}(\Delta_1, x, \Delta_2; \Delta_3)\end{array} \\ \downarrow \mathrm{cut} \times 1 & & \downarrow \mathrm{cut} \times 1 \\ \begin{array}{c}\mathbf{P}(\Delta_1, \Gamma_1, \Delta_2; \Gamma_2, \Delta_3, \Gamma_3, y, \Gamma_4) \\ \times \mathbf{P}(\Phi_1, y, \Phi_2; \Phi_3)\end{array} & & \begin{array}{c}\mathbf{P}(\Phi_1, \Gamma_1, \Phi_2; \Gamma_2, x, \Gamma_3, \Phi_3, \Gamma_4) \\ \times \mathbf{P}(\Delta_1, x, \Delta_2; \Delta_3)\end{array} \\ \downarrow \mathrm{cut} & & \downarrow \mathrm{cut} \\ \mathbf{P}(\Phi_1, \Delta_1, \Gamma_1, \Delta_2, \Phi_2; \Gamma_2, \Delta_3, \Gamma_3, \Phi_3, \Gamma_4) & = & \mathbf{P}(\Delta_1, \Phi_1, \Gamma_1, \Phi_2, \Delta_2; \Gamma_2, \Delta_3, \Gamma_3, \Phi_3, \Gamma_4) \end{array}$$



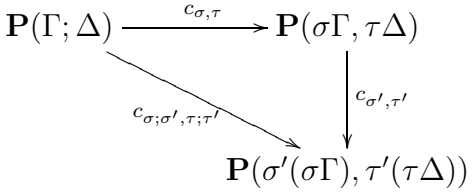
where $\gamma : A \times B \xrightarrow{\sim} B \times A$ is the isomorphism for products. Similarly here writing out the crossing condition explicitly would give four separate diagrams for each of the above diagrams.

A **symmetric polycategory P** is a polycategory equipped with a symmetric action, i.e., for permutations $\sigma \in \mathbf{S}_m,\ \tau \in \mathbf{S}_n$ (where $\mathbf{S}_k$ is the group of permutations on k objects), a map

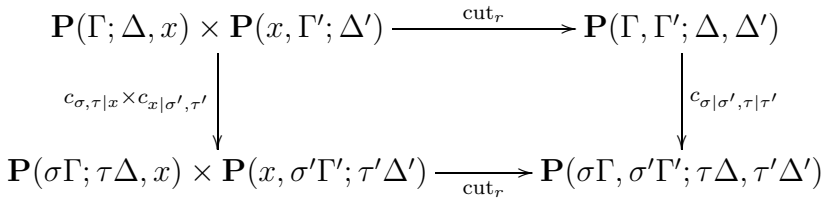
$$\mathbf{P}(\Gamma; \Delta) \xrightarrow{c_{\sigma,\tau}} \mathbf{P}(\sigma\Gamma; \tau\Delta)$$

where $\sigma\Gamma = \sigma(x_1, \ldots, x_n) = (x_{\sigma(1)}, \ldots x_{\sigma(n)})$, satisfying the following coherence conditions.

- The symmetric actions may be composed.



- That the next four diagrams commute assert that cutting and then permuting the objects is equivalent to first permuting the objects and then cutting.



$$\begin{array}{ccc}
\mathbf{P}(\Gamma; x, \Delta) \times \mathbf{P}(\Gamma', x; \Delta) & \xrightarrow{\text{cut}_l} & \mathbf{P}(\Gamma', \Gamma; \Delta', \Delta) \\
\downarrow{\scriptstyle c_{\sigma,x|\tau} \times c_{\sigma'|x,\tau'}} & & \downarrow{\scriptstyle c_{\sigma'|\sigma,\tau'|\tau}} \\
\mathbf{P}(\sigma\Gamma; x, \tau\Delta) \times \mathbf{P}(\sigma'\Gamma', x; \tau'\Delta') & \xrightarrow[\text{cut}_l]{} & \mathbf{P}(\sigma'\Gamma', \sigma\Gamma; \tau'\Delta', \tau\Delta)
\end{array}$$

$$\begin{array}{ccc}
\mathbf{P}(\Gamma; x) \times \mathbf{P}(\Gamma_1, x, \Gamma_2; \Delta) & \xrightarrow{\text{mcut}} & \mathbf{P}(\Gamma_1, \Gamma, \Gamma_2; \Delta) \\
\downarrow{\scriptstyle c} & & \downarrow{\scriptstyle c} \\
\mathbf{P}(\sigma\Gamma; x) \times \mathbf{P}(\Gamma'_1, x, \Gamma'_2; \tau\Delta) & \xrightarrow[\text{mcut}]{} & \mathbf{P}(\Gamma'_1, \sigma\Gamma, \Gamma'_2; \tau\Delta)
\end{array}$$

where $\Gamma'_1, -, \Gamma'_2 = \sigma(\Gamma_1, -, \Gamma_2)$.

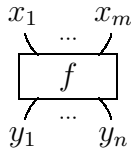
$$\begin{array}{ccc}
\mathbf{P}(\Gamma; \Delta_1, x, \Delta_2) \times \mathbf{P}(x; \Delta) & \xrightarrow{\text{mcut}^{\text{op}}} & \mathbf{P}(\Gamma; \Delta_1, \Delta, \Delta_2) \\
\downarrow{\scriptstyle c} & & \downarrow{\scriptstyle c} \\
\mathbf{P}(\sigma\Gamma; \Delta'_1, x, \Delta'_2) \times \mathbf{P}(x; \tau'\Delta) & \xrightarrow[\text{mcut}^{\text{op}}]{} & \mathbf{P}(\sigma\Gamma; \Delta'_1, \tau'\Delta, \Delta'_2)
\end{array}$$

where $\Delta'_1, -, \Delta_2 = \tau(\Delta_1, -, \Delta_2)$

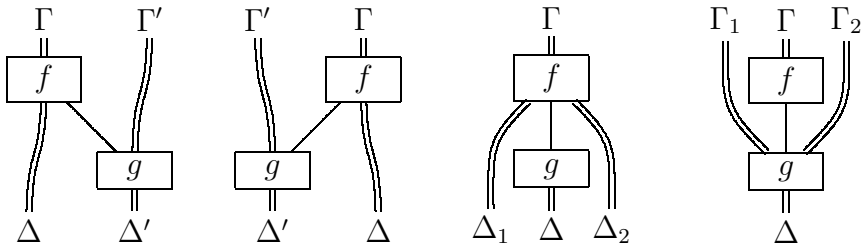
We will only be concerned with symmetric polycategories in this thesis and so will refer to the symmetric version of polycategories simply as polycategories.

1.7.1 Circuit diagrams for polycategories

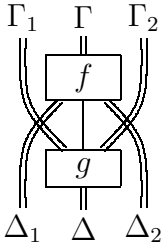
Circuit diagrams may be used to represent the constituents of a polycategory. Objects represent themselves and morphisms are labeled nodes with the domain objects connected to the top of the node by "wires" and the codomain objects connected to the bottom of the node. For example, the morphism $x_1, \ldots, x_m \xrightarrow{f} y_1, \ldots, y_n$ may be represented as:



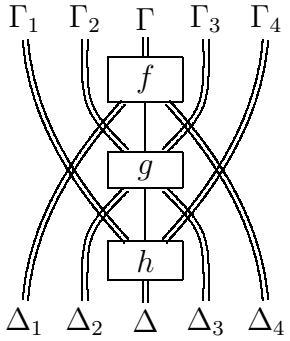
If we use double lines to represent strings of formulas, the (planar) cut rules are concerned with the following four shapes:



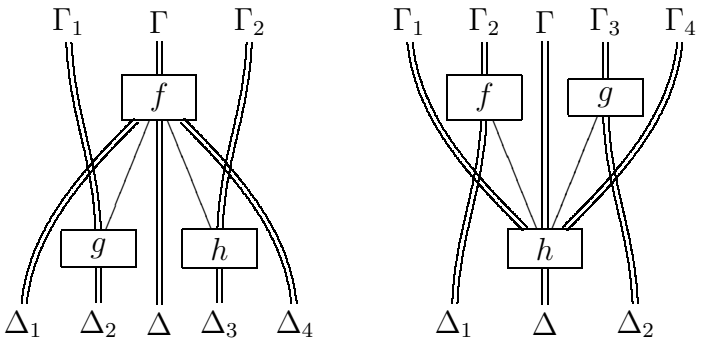
These four planar cut rules are equivalent to the following diagram with the condition that no wires are allowed to cross. This is called the **crossing** (or **planarity condition**).



Associativity is concerned with the diagrams of the form (up to the crossing condition):



and the two interchange laws with diagrams of the form (again, up to the crossing condition)



1.7.2 Morphisms of polycategories

Let $\mathbf{P}$ and $\mathbf{Q}$ be polycategories. A **morphism** $\mathbf{P} \xrightarrow{F} \mathbf{Q}$ between polycategories is a pair of maps $F_0 : \mathbf{P}_0 \to \mathbf{Q}_0$ and $F_1 : \mathbf{P}(\Gamma; \Delta) \to \mathbf{Q}(F^*(\Gamma); F^*(\Delta))$, where $F^*(X_1, \ldots, X_n) = F_0(X_1), \ldots, F_0(X_n)$, such that the functor preserves identities,

$$F_1(\mathrm{id}_A) = \mathrm{id}_{F_0(A)}$$

preserves composition,

$$\begin{array}{ccc}
\mathbf{P}(\Gamma_1; \Gamma_2, x, \Gamma_3) \times \mathbf{P}(\Delta_1, x, \Delta_2; \Delta_3) & \xrightarrow{\text{cut}} & \mathbf{P}(\Delta_1, \Gamma_1, \Delta_2; \Gamma_2, \Delta_1, \Gamma_3) \\
\downarrow F & & \downarrow F \\
\begin{array}{l}\mathbf{Q}(F^*(\Gamma_1); F^*(\Gamma_2), F_0(x), F^*(\Gamma_3)) \\ \times \mathbf{Q}(F^*(\Delta_1), F_0(x), F^*(\Delta_2); F^*(\Delta_3))\end{array} & \xrightarrow{\text{cut}} & \mathbf{Q}(F^*(\Delta_1), F^*(\Gamma_1), F^*(\Delta_2); F^*(\Gamma_2), F^*(\Delta_3), F^*(\Gamma_3))
\end{array}$$

and for symmetric polycategories, preserves the symmetric action,

$$\mathbf{Q}(F^*(\sigma\Gamma); F^*(\tau\Delta)) = \mathbf{Q}(\sigma(F^*(\Gamma)); \tau(F^*(\Delta)))$$

Again here, we usually drop the subscripts and write F for both F_0 and F_1.

Notice that a morphism of polycategories is the polycategorical notion of a functor between regular categories. There is a more elaborate notion of a polyfunctor introduced in [CKS03] which will not be discussed here.

As with functors, for any polycategory $\mathbf{P}$, there is an obvious notion of an identity morphism of polycategories $\mathrm{id}_{\mathbf{P}} : \mathbf{P} \to \mathbf{P}$ defined in the obvious way.

Two polycategories $\mathbf{P}$ and $\mathbf{Q}$ are said to be **isomorphic**, denoted $\mathbf{P} \cong \mathbf{Q}$, if there are a pair of morphisms of polycategories $F : \mathbf{P} \to \mathbf{Q}$ and $G : \mathbf{Q} \to \mathbf{P}$ such that $F; G = \mathrm{id}_{\mathbf{P}}$ and $G; F = \mathrm{id}_{\mathbf{Q}}$.

For a much more detailed exposition on polycategories and poly-bicategories see, e.g., [CKS03, CS97].

Remark 1.7.1 (Notation for polycategories) Until now we have avoided using any notation to represent the cut rule. Typically, one represents the cut rule using a "positional" notation, i.e., the object to cut on is specified by an index. For example, given

$$\Gamma_1 \xrightarrow{f} \Gamma_2 \qquad \text{and} \qquad \Delta_1 \xrightarrow{g} \Delta_2$$

the notation $f_i;_j g$ indicates that we are cutting the i-th component of Γ_2 with the j-th component of Δ_1, provided the requirements for cut are satisfied.

In this thesis we take a somewhat different approach. Instead of using a "positional" notation as above, we will use a "referential" notation. For each morphism in a polycategory we label the "wires" (or "channels") and use these labels to indicate which object we are cutting on. To prevent ambiguity, we require that each wire of a polymorphism receives a unique name, and make the restriction that cuts may only occur between polymorphisms which have no wire name in common (so that after the cut each wire still has a unique name). (This can be accomplished via a renaming procedure for wire names; we will not describe the details here.) For example, let

$$\alpha_1 : X, \alpha_2 : Y \xrightarrow{f} \alpha_3 : Z, \alpha : W \qquad \text{and} \qquad \beta : W, \beta_1 : S \xrightarrow{g} \beta_2 : T$$

be polymorphisms. We may cut f on $\alpha : W$ with g on $\beta : W$, which we will write $f_\alpha;_\beta g$.

Example 1.7.2 (Polycategories)

1. Any category is a polycategory with one input and one output.

2. Any (symmetric) multicategory (see, e.g., [Che02]) is a (symmetric) polycategory with one output.

3. Modules and multilinear maps form a multicategory (and hence a polycategory).

4. Any (symmetric) linearly distributive category[1] [CS97] is an example of a (symmetric) polycategory. As symmetric linearly distributive categories with negation are the same as $*$-autonomous categories (see [CS97] for the details), this implies that any $*$-autonomous category is an example of a polycategory.

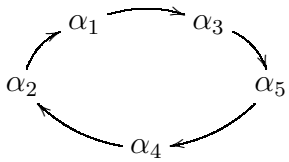
5. The primary example that this thesis is concerned with is: A Gentzen style sequent calculus (with multiple formulas on either side of the turnstile "$\vdash$") with formulas as objects and (equivalence classes of) derivations as the polymorphisms forms a polycategory.

Below is a concrete example which is worked out in more detail. This example provides the polycategorical version of an operad, and so is of some independent interest.

Example 1.7.3 (A one-object polycategory)

We may describe morphisms in a one-object polycategory (or **polyad**) using pairs consisting of labelled cyclic graphs and their related signatures, up to renaming of nodes. The nodes of the cyclic graphs are labeled with "channel" names. The signature of a graph indicates (from left-to-right) the ordering of its input and output wires. We will write these pairs simply as $G : S$, where G is a cyclic graph, and S is the signature, e.g.,

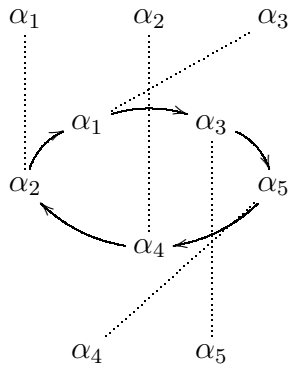
($\bigstar$)



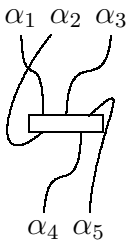
$: \langle \alpha_1, \alpha_2, \alpha_3 \rangle \to \langle \alpha_4, \alpha_5 \rangle$

[1] Originally referred to as weakly distributive categories.

The above polymorphism may be thought of graphically as



or, in circuit notation, as



Note that we are labeling the channels where one typically labels the objects, however, this is natural as, in this example, there is only one object.

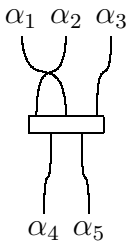
This polymorphism may also be denoted using a cyclic permutation presentation as follows

$$(\alpha_1, \alpha_3, \alpha_5, \alpha_4, \alpha_2) : \langle \alpha_1, \alpha_2, \alpha_3 \rangle \rightarrow \langle \alpha_4, \alpha_5 \rangle$$

Any cyclic permutation of a polymorphism represents the same polymorphism. Thus, an equivalent presentation of the above polymorphism is

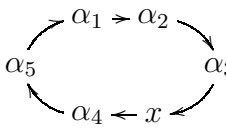
$$(\alpha_2, \alpha_1, \alpha_3, \alpha_5, \alpha_4) : \langle \alpha_1, \alpha_2, \alpha_3 \rangle \rightarrow \langle \alpha_4, \alpha_5 \rangle$$

which can be presented in circuit notation as



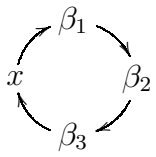
The identity polymorphism in this setting is given by $(\alpha, \beta) : \langle \alpha \rangle \to \langle \beta \rangle$.

Composition is given by "gluing" two composible cyclic graphs together (to make larger cycles). For example, the composite of



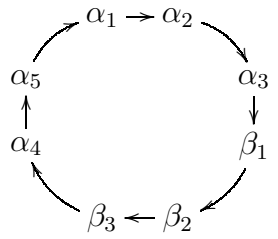
$$: \langle \alpha_1, \alpha_2, \alpha_3 \rangle \to \langle \alpha_4, \alpha_5, x \rangle$$

and



$$: \langle x, \beta_1 \rangle \to \langle \beta_2, \beta_3 \rangle$$

at the node x is



$$: \langle \alpha_1, \alpha_2, \alpha_3, \beta_1 \rangle \to \langle \alpha_4, \alpha_5, \beta_2, \beta_3 \rangle$$

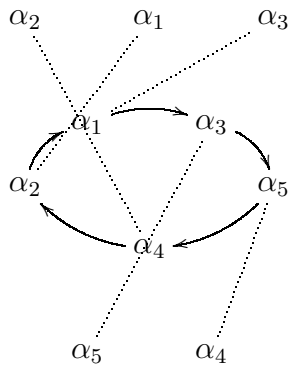
This data can now be seen to satisfy the requirements of a symmetric polycategory. The actions of the symmetric group on these polymorphisms simply permute the domain and codomain of the nodes in the signature. That is

$$c_{\sigma,\tau}(G : \Gamma \to \Delta) = G : \sigma\Gamma \to \tau\Delta$$

For example, if $\sigma(\langle \alpha_1, \alpha_2, \alpha_3 \rangle) = \langle \alpha_2, \alpha_1, \alpha_3 \rangle$ and $\tau(\langle \alpha_4, \alpha_5 \rangle) = \langle \alpha_5, \alpha_4 \rangle$ then

$$\begin{aligned} c_{\sigma,\tau}((\alpha_1, \alpha_3, \alpha_5, \alpha_4, \alpha_2) : \langle \alpha_1, \alpha_2, \alpha_3 \rangle \to \langle \alpha_4, \alpha_5 \rangle) \\ = (\alpha_1, \alpha_3, \alpha_5, \alpha_4, \alpha_2) : \langle \alpha_2, \alpha_1, \alpha_3 \rangle \to \langle \alpha_5, \alpha_4 \rangle \end{aligned}$$

which may be presented graphically as



This polycategory is, in fact, the polyad of a non-commutative cyclic monoid (in any linearly distributive category).

Chapter 2

$\Sigma\Pi$-Poly Logic

In this chapter we introduce $\Sigma\Pi$-poly logic; this logic is an extension of the logic $\Sigma\Pi$ of Cockett and Seely [CS01] which allows an arbitrary number of formulas on each side of the turnstile instead of exactly one. In what follows we will refer to this logic (the $\Sigma\Pi$-poly logic) as $\Sigma\Pi$. If there is a need to differentiate between the two logics it will be explicitly mentioned which logic, the $\Sigma\Pi$-poly logic or the Cockett-Seely $\Sigma\Pi$-logic, is under consideration.

In Section 2.1 the sequent calculus for $\Sigma\Pi$ is introduced. In Section 2.2, the cut-elimination rewrites for $\Sigma\Pi$-derivations are presented. The proof that these rewrites terminate is left until the next chapter. In $\Sigma\Pi$ the identity axiom applies only in the atomic case; in Section 2.3, it is shown that the identity holds for arbitrary formulas. Finally in Section 2.4, we give some equivalence schema, the so-called "permuting conversions", which are necessary in order for our system to have the Church-Rosser property.

2.1 The sequent calculus

The logic is presented in a Gentzen sequent style: a sequent takes the form $\Gamma \vdash \Delta$, where the antecedent Γ, and the succedent Δ, are comma separated strings of formulas. For convenience, we take the strings of formulas to be unordered which

then eliminates the need for the exchange rules:

$$\frac{\Gamma, X, Y, \Gamma' \vdash \Delta}{\Gamma, Y, X, \Gamma' \vdash \Delta} \qquad \text{and} \qquad \frac{\Gamma \vdash \Delta, X, Y, \Delta'}{\Gamma \vdash \Delta, Y, X, \Delta'}$$

The propositions are either atoms (which we write as $A, B, \ldots$) or compound formulas (which we write as $X, Y, \ldots$). A compound formula is either an I-ary **sum**, where I is a finite set, denoted $\sum_{i \in I} X_i$, or a **product**, denoted $\prod_{i \in I} X_i$. The index set I may be empty which gives the empty sum and product, denoted $\sum_{\emptyset} = 0$ and $\prod_{\emptyset} = 1$ respectively. For sufficiently small index sets we may write out the sum or product explicitly, e.g., $X + Y$ or $X \times Y$.

A typical rule in the Gentzen sequent style looks like

$$\frac{\Gamma_1 \vdash \Delta_1 \quad \Gamma_2 \vdash \Delta_2 \quad \cdots \quad \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta}$$

where we read this as an inference from top-to-bottom, i.e., if all the sequents on the top (the $\Gamma_i \vdash \Delta_i$'s) can be derived, then the sequent on the bottom ($\Gamma \vdash \Delta$) may be inferred. The special case is when there are no sequents on top

$$\frac{}{\Gamma \vdash \Delta}$$

in which case we may simply infer the bottom with no assumptions. A proof (or deduction) of a sequent is a finite tree with the given sequent at the root, axioms at the leaves and internal nodes corresponding to inference rules. A Gentzen style presentation together with its inference rules is called a **sequent calculus**. The rules of inference for $\Sigma\Pi$ are as follows:

$$\frac{}{A \vdash A}\ \text{(identity)}$$

$$\frac{\{\Gamma, X_i \vdash \Delta\}_{i\in I}}{\Gamma, \sum_{i\in I} X_i \vdash \Delta}\ \text{(cotuple)} \qquad \frac{\{\Gamma \vdash Y_i, \Delta\}_{i\in I}}{\Gamma \vdash \prod_{i\in I} Y_i, \Delta}\ \text{(tuple)}$$

$$\frac{\Gamma, X_k \vdash \Delta}{\Gamma, \prod_{i\in I} X_i \vdash \Delta}\ \text{(projection)} \qquad \frac{\Gamma \vdash Y_k, \Delta}{\Gamma \vdash \sum_{i\in I} Y_i, \Delta}\ \text{(injection)}$$

where $k \in I, I \neq \emptyset$

$$\frac{\Gamma \vdash \Delta, Z \quad Z, \Gamma' \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{(cut)}$$

Notice that in the cotuple and tuple rules the index set I may be empty, though not in the injection and projection rules.

Observe that the inference system for $\Sigma\Pi$ is self-dual, that is, it has an obvious sum-product symmetry. Explicitly, we may swap the direction of the sequents while turning sums into products and products into sums to obtain the same system. This means that each proof has a dual interpretation and can be "reused" to prove a dual theorem.

We shall consider various augmentations of this basic logic:

- The "initial logic" is the logic with no atoms. Notice that this is still a non-trivial logic because of the symbols $\Sigma_\emptyset$ and $\Pi_\emptyset$ which we may use to construct more complex formulas. We shall denote this logic as $\Sigma\Pi_\emptyset$.

- The "pure logic" is the logic with an arbitrary set of atoms A: we shall denote this logic as $\Sigma\Pi_A$.

- The "free logic" is the logic with an arbitrary set of atoms and an arbitrary set of non-logical axioms relating lists of atoms. If f is a non-logical axiom from A, B to C, D, this may be denoted as $f : A, B \to C, D$ or as an inference $\overline{A, B \vdash_f C, D}$. The atoms will be regarded as objects in a polycategory and the axioms as maps in that polycategory (with the "essential cuts" being provided

by composition). If the polycategory is denoted $\mathbf{A}$, the resulting logic will be denoted as $\Sigma\Pi_{\mathbf{A}}$.

If we think of the atoms of a pure logic as forming a discrete category, the free logic on this discrete category is then just the "pure" logic. Each variant above therefore includes the previous variant, and as it is more general, we shall tend to consider only this last variant.

Example 2.1.1 ($\Sigma\Pi$-derivations) Some typical proofs in this logic.

1.
$$\dfrac{\dfrac{\dfrac{\overline{A \vdash A}}{A \vdash A + B} \quad \dfrac{\overline{A \vdash A}}{A \vdash A + C}}{A \vdash (A + B) \times (A + C)} \quad \dfrac{\dfrac{\dfrac{\overline{B \vdash B}}{B \vdash A + B} \quad \dfrac{\overline{C \vdash C}}{C \vdash A + C}}{B \times C \vdash A + B} \quad \dfrac{\overline{C \vdash C}}{B \times C \vdash A + C}}{B \times C \vdash (A + B) \times (A + C)}}{A + (B \times C) \vdash (A + B) \times (A + C)}$$

This is one direction of the proof that sums distribute over products. The other direction cannot be proved in our system.

2.
$$\dfrac{\dfrac{\dfrac{\overline{A \vdash A}}{A \times B \vdash A} \quad \dfrac{\overline{B \vdash B}}{A \times B \vdash B}}{A \times B \vdash A \times B}}{\dfrac{A \times B \vdash (A \times B) + C}{(A \times B) + C \vdash (A \times B) + C} \quad \dfrac{\overline{C \vdash C}}{C \vdash (A \times B) + C}}$$

This proves the identity inference.

3. The above examples have only one proposition on each side of the turnstile. Here is an example in which each side has more than one proposition. In this case we will need some non-logical axioms.

$$\begin{array}{ll} i : A, E \to G, I & k : C, F \to H, I \\ j : B, E \to G, J & l : D, F \to H, K \end{array}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A,\ E \vdash_i G,\ I}\quad\dfrac{\dfrac{}{B,\ E \vdash_j G,\ J}}{B,\ E \vdash G,\ J+K}}{A \times B, E \vdash G, I \quad A \times B, E \vdash G, J+K}}{A \times B,\ E \vdash G,\ I \times (J+K)}\quad\dfrac{\dfrac{\dfrac{\dfrac{}{C,\ F \vdash_k H,\ I}\quad\dfrac{\dfrac{}{D,\ F \vdash_l H,\ K}}{D,\ F \vdash H,\ J+K}}{C \times D, F \vdash H, I \quad C \times D, F \vdash H, J+K}}{C \times D,\ F \vdash H,\ I \times (J+K)}}{A \times B,\ E \vdash G+H,\ I \times (J+K)}\quad\dfrac{}{C \times D,\ F \vdash G+H,\ I \times (J+K)}}{A \times B,\ E \times F \vdash G+H,\ I \times (J+K) \quad C \times D,\ E \times F \vdash G+H,\ I \times (J+K)}}{(A \times B) + (C \times D),\ E \times F \vdash G+H,\ I \times (J+K)}$$

2.2 Cut-elimination

In this section we show that cut-elimination holds for the free logic $\Sigma\Pi_{\mathbf{A}}$, i.e., any proof can be rewritten so that it does not contain any applications of the cut rule. Of course, this process will get stuck on the introduced atomic polymorphisms. A cut between atomic axioms is called an **essential cut**:

$$\dfrac{\Gamma_1 \vdash_f \Gamma_2, A \quad A, \Delta_1 \vdash_g \Delta_2}{\Gamma_1, \Delta_1 \vdash_{f;g} \Gamma_2, \Delta_2}$$

Proposition 2.2.1 (Cut-elimination) *Any proof in the free logic $\Sigma\Pi_{\mathbf{A}}$ can be transformed to a proof in which the only cuts are essential.*

We shall provide a family of rewrites for $\Sigma\Pi$-derivations and show that they terminate. As is typical in cut-elimination proofs, the rewrites will either replace a cut by cuts involving simpler formulas, or "push up" a cut into the surrounding proof. A proof that cannot be further rewritten using this set of rewrites will be a "cut-eliminated" proof in the sense of having no inessential cuts.

The rewrites are as follows. It will always be assumed that $i \in I$ and $j \in J$ for index sets I and J. Duality will be used to reduce the number of rewrites presented.

- Sequent-identity (identity-sequent): This rewrite removes the cut below an

identity axiom on the right.

$$\frac{\dfrac{\pi}{\Gamma \vdash \Delta, A} \quad \dfrac{\iota}{A \vdash A}}{\Gamma \vdash \Delta, A} \quad \Longrightarrow \quad \frac{\pi}{\Gamma \vdash \Delta, A}$$

The dual of this rewrite removes the cut below an identity axiom on the left.

- Cotuple-sequent (sequent-tuple): This rewrite moves a cut which is below a cotupling and an arbitrary sequent above the cotupling.

$$\frac{\left\{ \dfrac{\pi_i}{\Gamma, X_i \vdash \Delta, Z} \right\}_i \quad \dfrac{\pi}{Z, \Gamma' \vdash \Delta'}}{\Gamma, \sum X_i \vdash \Delta, Z \quad Z, \Gamma' \vdash \Delta'} \quad \Longrightarrow \quad \frac{\left\{ \dfrac{\dfrac{\pi_i}{\Gamma, X_i \vdash \Delta, Z} \quad \dfrac{\pi}{Z, \Gamma' \vdash \Delta'}}{\Gamma, X_i, \Gamma' \vdash \Delta, \Delta'} \right\}_i}{\Gamma, \sum X_i, \Gamma' \vdash \Delta, \Delta'}$$

The dual of this rewrite moves the cut above a tupling on the right.

- Injection-sequent (sequent-projection): This rewrite moves a cut which is below an injection and an arbitrary sequent above the injection. There are two cases to consider: the cut is on the injection, or it is on an arbitrary formula. The rewrite for the former case is below (the injection-cotuple rewrite); the rewrite for the latter case is as follows.

$$\frac{\dfrac{\dfrac{\pi}{\Gamma \vdash X_k, \Delta, Z}}{\Gamma \vdash \sum X_i, \Delta, Z} \quad \dfrac{\pi'}{Z, \Gamma' \vdash \Delta'}}{\Gamma, \Gamma' \vdash \sum X_i, \Delta, \Delta'} \quad \Longrightarrow \quad \frac{\dfrac{\dfrac{\pi}{\Gamma \vdash X_k, \Delta, Z} \quad \dfrac{\pi'}{Z, \Gamma' \vdash \Delta'}}{\Gamma, \Gamma' \vdash X_k, \Delta, \Delta'}}{\Gamma, \Gamma' \vdash \sum X_i, \Delta, \Delta'}$$

The dual of this rewrite moves the cut above a projection on the right.

- Projection-sequent (sequent-injection): This rewrite moves a cut which is below a projection and an arbitrary sequent above the projection.

$$\frac{\dfrac{\dfrac{\pi}{\Gamma, X_k \vdash \Delta, Z}}{\Gamma, \prod X_i \vdash \Delta, Z} \quad \dfrac{\pi'}{Z, \Gamma' \vdash \Delta'}}{\Gamma, \prod X_i, \Gamma' \vdash \Delta, \Delta'} \quad \Longrightarrow \quad \frac{\dfrac{\dfrac{\pi}{\Gamma, X_k \vdash \Delta, Z} \quad \dfrac{\pi'}{Z, \Gamma' \vdash \Delta'}}{\Gamma, X_k, \Gamma' \vdash \Delta, \Delta'}}{\Gamma, \prod X_i, \Gamma' \vdash \Delta, \Delta'}$$

The dual of this rewrite moves the cut above an injection.

- Tuple-sequent (sequent-cotuple): This rewrite moves a cut which is below a tuple and an arbitrary sequent above the tuple. There are two cases to consider: the cut is on the tupling, or the cut is on an arbitrary formula. The rewrite for the former case is dual to the injection-cotuple rewrite below; the rewrite for the latter case is as follows.

$$\dfrac{\left\{\dfrac{\pi_i}{\Gamma \vdash X_i, \Delta, Z}\right\}_i \quad \dfrac{\pi}{Z, \Gamma' \vdash \Delta'}}{\Gamma, \Gamma' \vdash \prod X_i, \Delta, \Delta'} \quad \Longrightarrow \quad \dfrac{\left\{\dfrac{\pi_i \quad \pi}{\Gamma, \Gamma' \vdash X_i, \Delta, \Delta'}\right\}_i}{\Gamma, \Gamma' \vdash \prod X_i, \Delta, \Delta'}$$

The dual of this rewrite moves the cut above a cotupling on the right.

- Injection-cotuple (tuple-projection): This rewrite moves the cut above an injection and cotupling.

$$\dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, X_k}}{\Gamma \vdash \Delta, \sum X_i} \quad \dfrac{\left\{\dfrac{\pi_i}{X_i, \Gamma' \vdash \Delta'}\right\}_i}{\sum X_i, \Gamma' \vdash \Delta'} \quad \Longrightarrow \quad \dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, X_k} \quad \dfrac{\pi_k}{X_k, \Gamma' \vdash \Delta'}}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

The dual of this rewrite moves the cut above a tupling and projection.

This accounts for all the ways in which compound formulas are introduced either on the left or on the right above a cut, and we have shown how to move the cut above these rules. Thus, a proof which cannot be rewritten further must have an axiom above the cut on each side; this is an essential cut.

It remains to show that this rewriting procedure terminates. For this we will need to define a measure on proofs which is reduced by each rewrite. The technical details and proof that the rewritings terminate will be presented in Chapter 3.

2.3 Identity derivations

Our goal is to view this proof system as a polycategory where cut is the composition. The cut-elimination process therefore provides part of the dynamics of composition: the activity which takes place when two proofs are plugged together.

Part of proving that cut acts as a composition is showing that there are identity derivations which behave in the correct manner. The identity derivations are defined inductively as follows.

- The identity atomic sequent:

$$\frac{}{A \vdash A}\ (\text{identity})$$

- The identity derivation on sums is given by

$$\frac{\left\{ \dfrac{\iota_{X_i}}{\dfrac{X_i \vdash X_i}{X_i \vdash \sum X_i}} \right\}_i}{\sum X_i \vdash \sum X_i}$$

  where the identity derivation ι_{X_i} of $X_i \vdash X_i$ is given by induction on the structure of X_i.

- The identity on products is given by the dual of the proof above. Explicitly,

$$\frac{\left\{ \dfrac{\iota_{X_i}}{\dfrac{X_i \vdash X_i}{\prod X_i \vdash X_i}} \right\}_i}{\prod X_i \vdash \prod X_i}$$

  where the identity derivation ι_{X_i} of $X_i \vdash X_i$ is given by induction on the structure of X_i.

The following lemma now proves that cutting (composing) any sequent derivation together with the identity derivation (and vise versa) results in the same sequent derivation.

Lemma 2.3.1 *The sequent-identity and identity-sequent cut-elimination reductions are derivable reductions for the general identity derivations as defined above. That is,*

$$\dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, X} \quad \dfrac{\iota_X}{X \vdash X}}{\Gamma \vdash \Delta, X} \quad \Longrightarrow \quad \dfrac{\pi}{\Gamma \vdash \Delta, X}$$

and similarly for the dual rule.

Proof. We shall suppose that the identity derivation is on the right; duality covers the other case. The proof is by structural induction on the derivation π.

- The base case is a cut with an atomic sequent: here the cut-elimination step removes the atomic identity and the result is immediate.

- Next suppose the identity is on a sum type:

$$\dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, \sum X_i} \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma \vdash \Delta, \sum X_i}$$

  There are four possibilities for the root inference of π.

  (1) If the root inference is a cotupling the cut-elimination step moves the cut onto smaller proofs. We may now apply the inductive hypothesis to each of these smaller proofs which yields the required result.

$$\dfrac{\left\{\dfrac{\pi_j}{\Gamma, Y_j \vdash \Delta, \sum X_i}\right\}_j \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma, \sum Y_j \vdash \Delta, \sum X_i} \Longrightarrow \dfrac{\left\{\dfrac{\dfrac{\pi_j}{\Gamma, Y_j \vdash \Delta, \sum X_i} \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma, Y_j \vdash \Delta, \sum X_i}\right\}_j}{\Gamma, \sum Y_j \vdash \Delta, \sum X_i}$$

  (2) If the root inference is an injection there are two possibilities: the cut is on $\sum X_i$ or it is not. In the first case we use the injection-cotuple rewrite and the second case the injection-sequent rewrite. The injection-cotuple rewrite is

as follows.

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, X_k}}{\Gamma \vdash \Delta, \sum X_i} \quad \dfrac{\left\{ \dfrac{\dfrac{\iota_i}{X_i \vdash X_i}}{X_i \vdash \sum X_i} \right\}_i}{\sum X_i \vdash \sum X_i}}{\Gamma \vdash \Delta, \sum X_i} \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash \Delta, X_k} \quad \dfrac{\iota_k}{X_k \vdash X_k}}{\Gamma \vdash \Delta, X_k}}{\Gamma \vdash \Delta, \sum X_i}$$

which moves the cut onto a smaller proof and so the inductive hypothesis applies and we are done. The injection-sequent rewrite is

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash Y_k, \Delta, \sum X_i}}{\Gamma \vdash \sum Y_j, \Delta, \sum X_i} \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma \vdash \sum Y_j, \Delta, \sum X_i} \Longrightarrow \dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash Y_k, \Delta, \sum X_i} \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma \vdash Y_k, \Delta, \sum X_i}}{\Gamma \vdash \sum Y_j, \Delta, \sum X_i}$$

which again moves the cut onto a smaller proof

(3) If the root inference is a projection the cut-elimination step moves the cut onto a smaller proof and so we are done.

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, Y_k \vdash \Delta, \sum X_i}}{\Gamma, \prod Y_j \vdash \Delta, \sum X_i} \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma, \prod Y_j \vdash \Delta, \sum X_i} \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\pi}{\Gamma, Y_k \vdash \Delta, \sum X_i} \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma, Y_k \vdash \Delta, \sum X_i}}{\Gamma, \prod Y_j \vdash \Delta, \sum X_i}$$

(4) If the root inference is a tupling the cut-elimination step moves the cut onto a smaller proof and so we are done.

$$\dfrac{\dfrac{\left\{ \dfrac{\pi_j}{\Gamma \vdash Y_j, \Delta, \sum X_i} \right\}_j}{\Gamma \vdash \prod Y_j, \Delta, \sum X_i} \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma \vdash \prod Y_j, \Delta, \sum X_i} \Longrightarrow \dfrac{\left\{ \dfrac{\dfrac{\pi_j}{\Gamma \vdash Y_j, \Delta, \sum X_i} \quad \dfrac{\iota}{\sum X_i \vdash \sum X_i}}{\Gamma \vdash Y_j, \Delta, \sum X_i} \right\}_j}{\Gamma \vdash \prod Y_j, \Delta, \sum X_i}$$

- Finally, suppose the identity is on a product type:

$$\frac{\dfrac{\pi}{\Gamma \vdash \Delta, \prod X_i} \quad \dfrac{\iota}{\prod X_i \vdash \prod X_i}}{\Gamma \vdash \Delta, \prod X_i}$$

This case is very similar to the one above. In all the cases for the root inference of π, other than tupling, the appropriate cut-elimination rewrite (e.g., when the root inference is a cotupling use the cotuple-sequent rewrite) will move the cut onto a smaller proof (or smaller proofs).

For the case where the root inference is a tupling there are two subcases to consider: the cut is on $\prod X_i$ or it is not. In the latter case the tuple-sequent cut-elimination rewrite suffices to move the cut onto smaller proofs. The former case is unique in that we must apply two rewrites: a sequent-tuple followed by a tuple-projection.

$$\frac{\left\{\dfrac{\pi_i}{\Gamma \vdash \Delta, X_i}\right\} \quad \left\{\dfrac{\dfrac{\iota_i}{X_i \vdash X_i}}{\prod X_i \vdash X_i}\right\}_i}{\dfrac{\Gamma \vdash \Delta, \prod X_i \quad \prod X_i \vdash \prod X_i}{\Gamma \vdash \Delta, \prod X_i}} \quad \Longrightarrow \quad \frac{\left\{\dfrac{\dfrac{\left\{\dfrac{\pi_i}{\Gamma \vdash \Delta, X_i}\right\}_i}{\Gamma \vdash \Delta, \prod X_i} \quad \dfrac{\dfrac{\iota_i}{X_i \vdash X_i}}{\prod X_i \vdash X_i}}{\Gamma \vdash \Delta, X_i}\right\}_i}{\Gamma \vdash \Delta, \prod X_i}$$

$$\Longrightarrow \quad \frac{\left\{\dfrac{\dfrac{\pi_i}{\Gamma \vdash \Delta, X_i} \quad \dfrac{\iota_i}{X_i \vdash X_i}}{\Gamma \vdash \Delta, X_i}\right\}_i}{\Gamma \vdash \Delta, \prod X_i}$$

■

2.4 Permuting conversions

In order to obtain a normal form for sequent derivations, we would like to show that the cut-elimination rewrites are Church-Rosser. Currently this is not the case; for

example, consider a derivation with a cotupling and tupling immediately above the cut:

$$\dfrac{\left\{\dfrac{\pi_i}{\Gamma, X_i \vdash \Delta, Z}\right\}_i \quad \left\{\dfrac{\pi'_j}{Z, \Gamma' \vdash Y_j, \Delta'}\right\}_j}{\dfrac{\Gamma, \sum_i X_i \vdash \Delta, Z \qquad Z, \Gamma' \vdash \prod_j Y_j, \Delta'}{\Gamma, \sum_i X_i, \Gamma' \vdash \Delta, \prod_j Y_j, \Delta'}}$$

In this case one may apply the cotuple-sequent rewrite or the sequent-tuple rewrite to reduce the derivation, but there seems to be no way in which to resolve these rewrites. This motivates the use of additional rewrites which will allow us to interchange these two rules. Similar considerations for the nine other possible critical pairs (cotuple vs. cotuple, cotuple vs. injection, etc.) leads us to the following ten conversions (which we denote by $\models\!\!=\!\!\models$). Again, duality is used to reduce the number of conversions presented.

- Cotuple-cotuple (tuple-tuple) interchange:

$$\dfrac{\left\{\dfrac{\left\{\dfrac{\pi_{ij}}{X_i, \Gamma, Y_j \vdash \Delta}\right\}_j}{X_i, \Gamma, \sum Y_j \vdash \Delta}\right\}_i}{\sum X_i, \Gamma, \sum Y_j \vdash \Delta} \quad \models\!\!=\!\!\models \quad \dfrac{\left\{\dfrac{\left\{\dfrac{\pi_{ij}}{X_i, \Gamma, Y_j \vdash \Delta}\right\}_i}{\sum X_i, \Gamma, Y_j \vdash \Delta}\right\}_j}{\sum X_i, \Gamma, \sum Y_j \vdash \Delta}$$

- Cotuple-injection (projection-tuple) interchange:

$$\dfrac{\left\{\dfrac{\dfrac{\pi_i}{\Gamma, X_i \vdash Y_k, \Delta}}{\Gamma, X_i \vdash \sum Y_j, \Delta}\right\}_i}{\Gamma, \sum X_i \vdash \sum Y_j, \Delta} \quad \models\!\!=\!\!\models \quad \dfrac{\dfrac{\left\{\dfrac{\pi_i}{\Gamma, X_i \vdash Y_k, \Delta}\right\}_i}{\Gamma, \sum X_i \vdash Y_k, \Delta}}{\Gamma, \sum X_i \vdash \sum Y_j, \Delta}$$

- Cotuple-projection (injection-tuple) interchange:

$$\dfrac{\left\{\dfrac{\dfrac{\pi_i}{X_i, \Gamma, Y_k \vdash \Delta}}{X_i, \Gamma, \prod Y_j \vdash \Delta}\right\}_i}{\sum X_i, \Gamma, \prod Y_j \vdash \Delta} \quad \models\!\!=\!\!\models \quad \dfrac{\left\{\dfrac{\pi_i}{X_i, \Gamma, Y_k \vdash \Delta}\right\}_i}{\dfrac{\sum X_i, \Gamma, Y_k \vdash \Delta}{\sum X_i, \Gamma, \prod Y_j \vdash \Delta}}$$

- Cotuple-tuple interchange:

$$\dfrac{\left\{\dfrac{\left\{\dfrac{\pi_{ij}}{\Gamma, X_i \vdash Y_j, \Delta}\right\}_j}{\Gamma, X_i \vdash \prod Y_j, \Delta}\right\}_i}{\Gamma, \sum X_i \vdash \prod Y_j, \Delta} \quad \Longmapsto \quad \dfrac{\left\{\dfrac{\left\{\dfrac{\pi_{ij}}{\Gamma, X_i \vdash Y_j, \Delta}\right\}_i}{\Gamma, \sum X_i \vdash Y_j, \Delta}\right\}_j}{\Gamma, \sum X_i \vdash \prod Y_j, \Delta}$$

- Injection-injection (projection-projection) interchange:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash X_k, \Delta, Y_{k'}}}{\Gamma \vdash X_k, \Delta, \sum Y_j}}{\Gamma \vdash \sum X_i, \Delta, \sum Y_j} \quad \Longmapsto \quad \dfrac{\dfrac{\dfrac{\pi}{\Gamma \vdash X_k, \Delta, Y_{k'}}}{\Gamma \vdash \sum X_i, \Delta, Y_{k'}}}{\Gamma \vdash \sum X_i, \Delta, \sum Y_j}$$

- Projection-injection interchange:

$$\dfrac{\dfrac{\dfrac{\pi}{\Gamma, X_k \vdash Y_{k'}, \Delta}}{\Gamma, X_k \vdash \sum Y_j, \Delta}}{\Gamma, \prod X_i \vdash \sum Y_j, \Delta} \quad \Longmapsto \quad \dfrac{\dfrac{\dfrac{\pi}{\Gamma, X_k \vdash Y_{k'}, \Delta}}{\Gamma, \prod X_i \vdash Y_{k'}, \Delta}}{\Gamma, \prod X_i \vdash \sum Y_j, \Delta}$$

It can now be shown that the cut-elimination procedure for ΣΠ-derivations is confluent modulo these permuting conversions. It is, however, more convenient to do so after providing a term representation for the ΣΠ-derivations. As this is the purpose of the next chapter, the proof of confluence will also be presented there in Section 3.5.

Chapter 3

Term Logic for $\Sigma\Pi$-Poly Maps

In this chapter we prove that the cut-elimination rewrites are Church-Rosser and terminating modulo the permuting conversions. It will, therefore, be convenient to have a more compact notation for sequent derivations; this leads us to introduce a system of terms typed by sequents.

The terms for the $\Sigma\Pi$-logic introduced in this chapter will reflect the view of the formulas as protocols and the proofs as processes. That is, that a proof of a sequent $\Gamma \vdash \Delta$ may be regarded as a process between the protocols (the formulas) in Γ and Δ. As we shall see in the next chapter, these terms provide a categorical semantics for the logic: the free polycategory with sums and products.

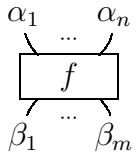
In order to motivate the term logic we shall start by introducing the view of this system as protocols and processes. In Section 3.3, we then introduce two term representations for $\Sigma\Pi$-derivations: a compact term calculus representation and a "programming language" representation. A rewriting system for cut-elimination using these representations is developed. In Section 3.4, it is made explicit how the term calculus handles the additive units (the objects $\Sigma_\emptyset$ and $\Pi_\emptyset$). Then, with our term calculus representation in hand, in Sections 3.5 and 3.6, we prove that our cut-elimination rewriting system for the $\Sigma\Pi$-derivations has the Church-Rosser property and is terminating. Finally, in Section 3.7, we give a decision procedure for $\Sigma\Pi$-derivations.

3.1 Protocols and processes

If we adopt the system/environment perspective it is natural to view polymorphisms as processes and the objects as protocols: the system is connected to the environment via channels along which the environment may input to the system and the system may output to the environment, where the channels here allow two-way communication, i.e., on any given channel one may receive input and/or produce output.

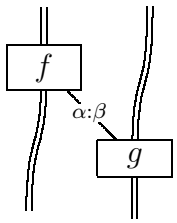
Protocols are assigned to channels along which processes can communicate (in both input and output mode). A protocol on a channel may be thought of as a convention between the system and the environment that dictates who may place the next event on that channel. We will adopt the convention of using Greek letters for channel names. Let S represent a system and E the environment such that they are connected via a channel α. Suppose P is a protocol which is only going to allow the system to output either the event a or the event b to the environment. Placing P on α between S and E, $S \overset{\alpha:P}{\longleftrightarrow} E$, allows S to output either the event a or the event b, and moreover, prevents E from inputing anything to S.

A process should be regarded as an agent which acts on any number of channels by either accepting input events or producing output events in accordance with the protocols associated with each channel. Thus, it is a system communicating with the environment along a number of channels. For reasons that will become apparent, the channels a process interacts along are divided into (unordered) domain (or "input") channels and (unordered) codomain (or "output") channels. A process f with domain channels $\alpha_1, \ldots, \alpha_n$ and codomain channels $\beta_1, \ldots, \beta_m$ may be represented graphically as



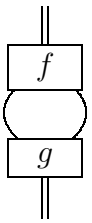
Our convention will be to have the domain channels (the α_i's) on the top and the codomain channels (the β_j's) on the bottom. Recall that we are considering these channels unordered. If a codomain channel α of a process f, and a domain

channel β of a process g, share a common protocol then f and g may be "plugged" (or composed) together on α and β to form a new process. The domain channels of this composed process are the domain channels of f and the domain channels of g (minus β) and the codomain channels are the codomain channels of f (minus α) and the codomain channels of g. This may be pictured graphically as



where the double lines represent strings of channels. (This plugging together of processes is exactly the cut rule. This is explained in detail in Section 3.3.2).

Note that any two processes may be plugged together on at most one channel. That is, situations like the following are not allowed:



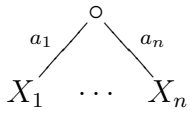
Physically, one could think of the channels as "wires", and the processes as black boxes which send events along these wires, where the protocols determine which events are allowed to be passed though the wires at any given moment.

3.2 Formulas as protocols

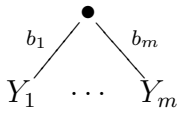
"Protocols" in our system will simply be ΣΠ-terms annotated with channel names and "events". Formally, a **protocol** may be either:

- an **atomic** protocol: $A, B, C, \ldots$ (the objects of a polycategory)

- a **coproduct** protocol: $\sum_{i \in I} a_i : X_i = \{a_i : X_i \mid i \in I\}$ where each X_i is a protocol. If we suppose $I = \{1, \ldots, n\}$, we could represent this by the following tree:



- a **product** protocol: $\prod_{j \in J} b_j : Y_j = (b_j : Y_j \mid j \in J)$ where each Y_j is a protocol. Again, we could represent this by the following tree:



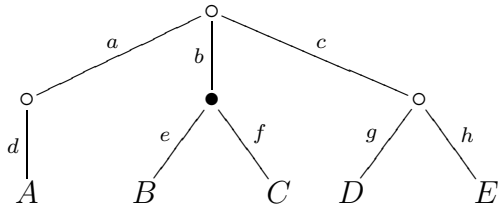
The a_i's and b_j's are thought of as the events. An event, as introduced here, is not a formal object; for our purposes they may be thought of as names (or tags, constructors, etc.).

Note that in specifying protocols we allow the index sets I and J to be empty; this gives two atomic protocols

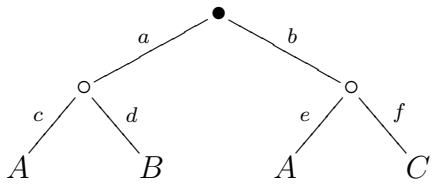
$$\sum\nolimits_{\emptyset} = \{\ \} = \circ \qquad \text{and} \qquad \prod\nolimits_{\emptyset} = (\ ) = \bullet$$

Example 3.2.1 (Protocols)

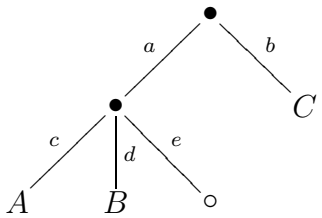
1. $X = \{a : \{d : A\}, b : (e : B, f : C), c : \{g : D, h : E\}\}$



2. $Y = (a : \{c : A, d : B\}, b : \{e : A, f : C\})$



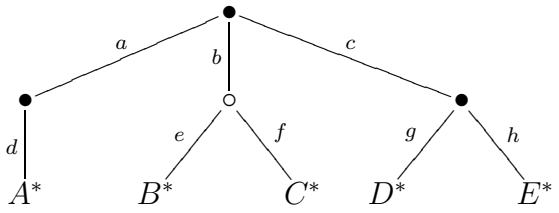
3. $Z = (a : (c : A, d : B, e : \{\}), b : C)$



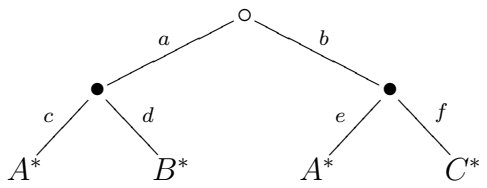
where A, B, C, D, and E are protocols.

Given a protocol X, we may form its **dual** protocol X^* by flipping the product and coproduct structure and leaving the channel names and events alone. For example, the duals of our protocols above are:

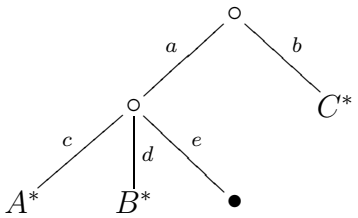
1. $X^* = (a : (d : A^*), b : \{e : B^*, f : C^*\}, c : (g : D^*, h : E^*))$



2. $Y^* = \{a : (c : A^*, d : B^*), b : (e : A^*, f : C^*)\}$



3. $Z^* = \{a : \{c : A^*, d : B^*, e : ()\}, b : C^*\}$



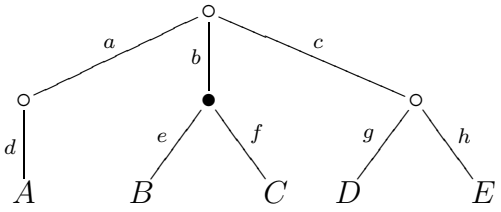
Notice that taking the dual of some protocol implies that the atoms in that protocol must have a dual in the underlying polycategory. In the "initial" $\Sigma\Pi$-logic (the logic with no atoms) taking the dual of a dual gets us back to the original formula, i.e., $(X^*)^* = X$. This is true in the "pure" and "free" logic if and only if it is true for the atoms.

What we now want to do is to assign protocols to channels. Channels, as mentioned earlier, will be denoted with Greek letters. The notation $\alpha : X$ will denote the assignment of the protocol X to channel α.

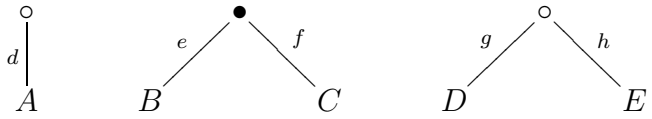
3.2.1 Protocol transitions

A protocol may make a transition to one of its subprotocols (subformulas) by following one of the edges from the "root" of the protocol. For example, given the

protocol



the transitions a, b, and c (respectively) lead to the following three subprotocols:



In our system, in order for a protocol to make a transition it needs be triggered by an event, of which there are two kinds: input events and output events. Given a channel α and an event a we use $\alpha[a]$ to denote that the event a has been input on channel α and $\overline{\alpha}[a]$ to denote that the event a has been output on channel α (following, for example, the convention of the π-calculus). Output events are thought of as being generated by the system and input events as being generated by the environment. There is an obvious analogy here with the game theoretic view of player and opponent moves.

A process has two types of channels: domain channels and codomain channels. A protocol will have a different (actually dual) behaviour depending on which type of channel we associate it with. When a process is using a protocol X on an domain channel α, it will use the **domain protocol** of X, denoted $\mathrm{dom}(X)$. When a process is using X on a codomain channel β, it will use the **codomain protocol** of X, denoted $\mathrm{cod}(X)$.

The protocols $\mathrm{dom}(X)$ and $\mathrm{cod}(X)$ have as states all the subprotocols of X. We are then able to associate with each protocol X and channel α, two finite labeled trees (or labeled transition systems): one for $\mathrm{dom}(X)$ and one for $\mathrm{cod}(X)$. In a domain transition system a coproduct protocol may only respond to input events while a product protocol may only respond to output events. Dually, in a codomain transition system a coproduct protocol may only respond to output events while a

X	dom(X) expects:	cod(X) expects:
$\{a_i : X_i\}_i$	input	output
$(a_i : X_i)_i$	output	input

Table 3.1: Expected transitions for a protocol X on a channel α

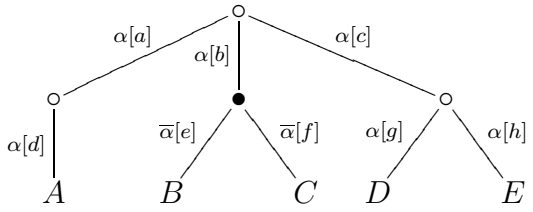
product protocol may only respond to input events. We summarize these expected transitions in Table 3.1.

Example 3.2.2 (Protocol transitions)

1. The transitions of

$$\mathrm{dom}(\alpha : X) = \alpha : \{a : \{d : A\}, b : (e : B, f : C), c : \{g : D, h : E\}\}$$

may be displayed as

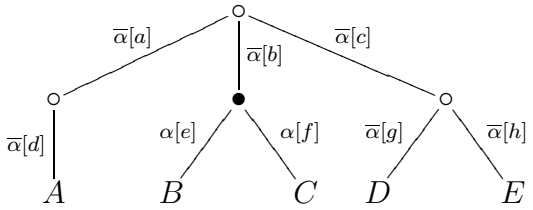


This protocol says that on α in the domain the system must wait for one of the events a, b, or c to be input from the environment. If a is received the system must to wait for the event d to be input, after which it will continue on with the protocol A. Alternatively, if a b is received the system enters an output state at which point it may output either e or f, and then continue with protocol B or C respectively. Lastly, if c is received it must then wait for more input, either g or h after which it will continue with D or E respectively.

2. The transitions of

$$\mathrm{cod}(\alpha : X) = \alpha : \{a : \{d : A\}, b : (e : B, f : C), c : \{g : D, h : E\}\}$$

may be displayed as

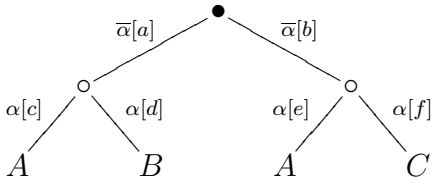


Notice that we could turn input events into output events and output events into input events to obtain the same transition system as (1) above.

3. The transitions of

$$\mathrm{dom}(\alpha : Y) = \alpha : (a : \{c : A, d : B\}, b : \{e : A, f : C\})$$

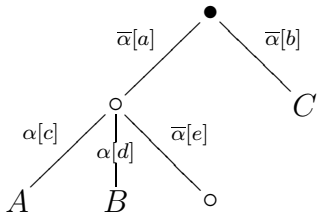
may be displayed as



4. The transitions of

$$\mathrm{dom}(\alpha : Z) = \alpha : (a : (c : A, d : B, e : \{\}), b : C)$$

may be displayed as



3.3 Proofs as processes

A **process** in our system is a representation of a ΣΠ-derivation. In the previous section we saw that on a domain channel, a coproduct protocol allows the environment to input certain events to a process, and a product protocol allows a process to output certain events to the environment. Protocols on a codomain channel behave dually, i.e., on an codomain channel, a coproduct protocol allows a process to output certain events to the environment, and a product protocol allows the environment to input certain events to a process. Additionally, there are atomic protocols between which there can be an "atomic process" (i.e., an atomic map).

This means that the kind of events a process may perform depend on the "type" of the process. The **type** of a process f is defined to be two strings of channel names (the domain channels and the codomain channels), and an assignment of protocols to those channels. This will be denoted

$$\alpha_1 : X_1, \ldots, \alpha_n : X_n \underset{f}{\longrightarrow} \beta_1 : Y_1, \ldots, \beta_m : Y_m$$

The channels to the left of the arrow are the domain channels, and the channels to the right of the arrow are codomain channels. In the following, Γ and Δ will be used to represent comma separated strings of channels and their assigned protocols, e.g., $\Gamma = \alpha_1 : X_1, \ldots, \alpha_n : X_n$.

In this section we introduce two term representations for ΣΠ-derivations (processes). The first is a compact term calculus which will allow for easy manipulations of the derivations (i.e., cut reduction steps and permuting conversions). This representation will be used for proving the Church-Rosser property and cut-elimination. The second term representation uses a "programming language like" syntax which will be more representative of the "proofs as processes".

3.3.1 A term calculus representation

The formation rules for this representation are given in Table 3.2. The notation $\alpha\{\}$ will be used for the map from the empty sum on a domain channel α and $\beta()$ for the map to the empty product on a codomain channel β.

$$\frac{}{\alpha : A \vdash_{1_A} \beta : A}\ \text{(identity)}$$

$$\frac{\{\Gamma, \alpha : X_i \vdash_{f_i} \Delta\}_{i \in I}}{\Gamma, \alpha : \sum_{i \in I} a_i : X_i \vdash_{\alpha\{a_i \mapsto f_i\}_{i \in I}} \Delta} \qquad \frac{\{\Gamma \vdash_{f_i} \alpha : X_i, \Delta\}_{i \in I}}{\Gamma \vdash_{\alpha(a_i \mapsto f_i)_{i \in I}} \alpha : \prod_{i \in I} a_i : X_i, \Delta}$$

$$\frac{\Gamma, \alpha : X_k \vdash_f \Delta}{\Gamma, \alpha : \prod_{i \in I} a_i : X_i \vdash_{\overleftarrow{\alpha}[a_k](f)} \Delta} \qquad \frac{\Gamma \vdash_f \alpha : X_k, \Gamma}{\Gamma \vdash_{\overrightarrow{\alpha}[a_k](f)} \alpha : \sum_{i \in I} a_i : X_i, \Gamma}$$

where $k \in I, I \neq \emptyset$

$$\frac{\Gamma \vdash_f \Delta, \gamma : Z \quad \gamma : Z, \Gamma' \vdash_g \Delta'}{\Gamma, \Gamma' \vdash_{f;_\gamma g} \Delta, \Delta'}$$

Table 3.2: $\Sigma\Pi$ term calculus formation rules

If the domain and the codomain (i.e., both sides of the turnstile) consist of the same single atom A (e.g., $A \vdash A$), the identity rule asserts that the process may output the atomic identity map $1_A : A \to A$ (from the underlying polycategory) and then end. The axiom rule asserts that if a process is sitting between atoms Γ in the domain and Δ in the codomain, and $f : \Gamma \to \Delta$ is an atomic map, then the process may output f and then end.

Given a process that is interacting on a channel α in an input state (the protocol assigned to α is in a state in which input is allowed: a coproduct protocol in the domain or a product protocol in the codomain), the cotuple and tuple rules assert that the process must be prepared to accept any one of the possible input events that the protocol will allow on α, in this case any of the a_i's, for $i \in I$. After receiving one of the input events a_i the process will then continue on with its subprocess f_i (where, in the cotuple case, the type of f_i is $\Gamma, \alpha : X_i \to \Delta$, and in the tuple case, $\Gamma \to \alpha : X_i, \Delta$).

Given a process that is interacting on a channel α in an output state, the projection and injection rules assert that the process may output one of the specified events, in this case, one of the a_i's, for $i \in I$, and then continue on with its subprocess

f of type $\Gamma, \alpha : X_i \to \Delta$ or $\Gamma \to \alpha : X_i, \Delta$ respectively.

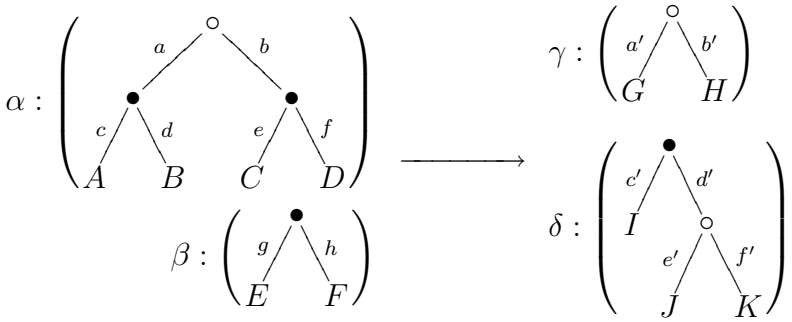
Example 3.3.1 (Term calculus representation)

1. Suppose that the following are atomic maps

$$i : A, E \to G, I \qquad j : B, E \to G, J$$

$$k : C, F \to H, I \qquad l : D, F \to H, K$$

and consider the assignment of protocols to channels:



A process between α and β in the domain and γ and δ in the codomain is:

$$\alpha \left\{ \begin{array}{l} a \mapsto \overleftarrow{\beta}[g](\overrightarrow{\gamma}[a'](\delta \left(\begin{array}{l} c' \mapsto \overleftarrow{\alpha}[c](i) \\ d' \mapsto \overleftarrow{\alpha}[d](\overrightarrow{\delta}[e'](j)) \end{array} \right))) \\ b \mapsto \overleftarrow{\beta}[h](\overrightarrow{\gamma}[b'](\delta \left(\begin{array}{l} c' \mapsto \overleftarrow{\alpha}[e](k) \\ d' \mapsto \overleftarrow{\alpha}[f](\overrightarrow{\delta}[f'](l)) \end{array} \right))) \end{array} \right\}$$

This process must first wait for the environment to input either an the event a or the event b on α; say it receives the event a. The process will then proceed to output first g on β, and then a' on γ, after which it will wait for either a c' or a d' to be input on δ. Suppose c' is input, then it will output c on α and finish by outputting the atomic map i.

This, of course, is just describing one possible interaction that could happen. The environment could have first input b on α, in which case a different interaction would have occurred.

This term represents the derivation of the sequent in the third example of Example 2.1.1 annotated in the following way:

$$\alpha{:}\{a : (c : A, d : B), b : (e : C, f : D)\},\ \beta : (g : E, h : F) \vdash \gamma : \{a' : G, b' : H\},\ \delta : (c' : I, d' : \{e' : J, f' : K\})$$

2. Consider the following annotated version Example 2.1.1(2):

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{\alpha : A \vdash \beta : A}}{\alpha : (e : A, f : B) \vdash \beta : A} \quad \dfrac{\overline{\alpha : B \vdash \beta : B}}{\alpha : (e : A, f : B) \vdash \beta : B}}{\alpha : (e : A, f : B) \vdash \beta : (g : A, h : B)}}{\alpha : (e : A, f : B) \vdash \beta : \{c : (g : A, h : B), d : C\}} \quad \dfrac{\overline{\alpha : C \vdash \beta : C}}{\alpha : C \vdash \beta : \{c : (A, B), d : C\}}}{\alpha : \{a : (e : A, f : B), b : C\} \vdash \beta : \{c : (g : A, h : B), d : C\}}$$

The term corresponding to this sequent derivation is

$$\alpha \left\{ \begin{array}{l} a \mapsto \overrightarrow{\beta}[c](\beta \left(\begin{array}{l} g \mapsto \overleftarrow{\alpha}[e](1_A) \\ h \mapsto \overleftarrow{\alpha}[f](1_B) \end{array} \right)) \\ b \mapsto \overrightarrow{\beta}[d](1_C) \end{array} \right\}$$

3.3.2 Cut-elimination rewrites

If a domain channel of a process f, and a codomain channel of a process g, share a common protocol then these processes may be "plugged" (or composed) together on these channels to form a new process. For example, the two processes

$$\alpha : X \underset{f}{\longrightarrow} \gamma : Z \qquad \text{and} \qquad \gamma : Z \underset{g}{\longrightarrow} \beta : Y$$

share the protocol Z in common where Z is in the codomain of f and in the domain of g, and thus, f and g may be "plugged" together on γ to form a new process

$$\alpha : X \underset{f ;_\gamma g}{\longrightarrow} \beta : Y$$

In this new process, any output event on γ from f becomes an input event on

γ for g, and vise versa. This means that the environment may no longer input to either γ as it is hidden (in the same sense as in the π-calculus).

In general, processes have many domain and codomain channels, e.g.,

$$\Gamma \underset{f}{\longrightarrow} \Delta, \gamma : Z \qquad \text{and} \qquad \gamma : Z, \Gamma' \underset{g}{\longrightarrow} \Delta'$$

Plugging these processes together on γ results in

$$\Gamma, \Gamma' \underset{f ;_\gamma g}{\longrightarrow} \Delta, \Delta'$$

where Γ and Γ' are the domain channels and Δ and Δ' are the codomain channels for this new process.

Notice that plugging processes together can only occur when they have exactly one channel name in common: after this plugging process all the channel names must be distinct. This means that channels may have to be renamed in order to compose. We shall use (simultaneous) channel name substitution to indicate this renaming process

$$f[\alpha'_1/\alpha_1, \ldots, \alpha'_n/\alpha_n]$$

where $\alpha_1, \ldots, \alpha_n$ must be distinct channel names, as must $\alpha'_1, \ldots, \alpha'_n$, and

$$(\text{channels}(f) \backslash \{\alpha_1, \ldots, \alpha_n\}) \cap \{\alpha'_1, \ldots, \alpha'_n\} = \emptyset$$

so that the new channel names are distinct from the old channel names which are not replaced.

Example 3.3.2 (Renaming channels)

1. In order to compose

$$\alpha : W, \beta : X \vdash_f \gamma : Y, \delta : Z \qquad \text{and} \qquad \delta : Z, \alpha : W' \vdash_g \beta : X', \gamma : Y'$$

on δ we must first rename $\alpha,\ \beta$, and γ in f or g. Suppose we rename in g:

$$g[\epsilon/\alpha,\ \eta/\beta,\ \theta/\gamma] = \delta : Z,\ \epsilon : W' \vdash_g \eta : X',\ \theta : Y'$$

The result of composing f and the renamed g on δ is then

$$\alpha : W,\ \beta : X,\ \epsilon : W' \vdash_{f;_\delta g} \gamma : Y,\ \eta : X',\ \theta : Y'$$

2. Suppose we wish to compose the following two processes

$$\alpha : X \vdash_f \beta : Z \qquad \text{and} \qquad \gamma : Z \vdash_g \delta : Y$$

using $\beta : Z$ from f and $\gamma : Z$ from g. In this case, we may substitute either β for γ, γ for β, or a fresh channel name (a channel name which does not occur in either f or g) for both β and γ. Suppose we substitute γ for β in f which yields $\alpha : X \vdash_f \gamma : Z$. Now f and g may be composed on γ:

$$\alpha : X \vdash_{f;_\gamma g} \delta : Y$$

In general we may end up renaming the channels on which we want to compose to have the same "name" and the rest of the channels to be distinct from one another.

To simplify the exposition, in the following we will simply assume that the channel names of processes are distinct unless otherwise specified. For example, given the following two processes

$$\Gamma \underset{f}{\longrightarrow} \Delta, \gamma : X \qquad \text{and} \qquad \gamma : X, \Gamma' \underset{g}{\longrightarrow} \Delta'$$

it will be assumed that the only channel name f and g have in common is γ.

It should now be obvious that plugging together two processes is exactly an application of the cut rule. The dynamics of cut-elimination then can be seen as the way in which two processes communicate.

The cut-elimination reductions and the permuting conversions are summarized in Table 3.3. (Typing information has been omitted as it can be inferred from the terms, and in any case these have been displayed as sequent derivations in the previous chapter.) Note that apart from (19) and (22), these come in dual pairs, so there are six rewrites, eight conversions and their duals, and two other conversions

(which are self-dual): essentially 16 rewrites.

Recall that we allow the index sets I and J to be empty, except where the projection and injection rules are involved (the rules (5), (7), (11), (15), (17), (20), (22) and their duals); in these cases, since reference is made to an element k or l, it does not make sense for the corresponding index set to be empty. In (15) and (17) (and their duals) the index set J for the named element k must not be empty, but the other index set I may be. In (13), (14), and (19) either (or both or neither) index set may be empty. An explicit treatment of these nullary cases may be found in Section 3.4.

Example 3.3.3 (Process communication (composition)) Suppose that

- $X = \alpha : \{a : (c : A, d : B), b : (e : A, f : C)\}$
- $Y = \beta : \{a : B, b : C\}$
- $Z = \gamma : (a : A, b : \{c : B, d : C\})$

are protocols and consider the following situation:

$$\alpha : X \longrightarrow \gamma : Z \qquad \gamma : Z \longrightarrow \beta : Y$$

$$\alpha \left\{ \begin{array}{l} a \mapsto \gamma \left(\begin{array}{l} a \mapsto \overleftarrow{\alpha}[c](1_A) \\ b \mapsto \overrightarrow{\gamma}[c](\overleftarrow{\alpha}[d](1_B)) \end{array} \right) \\ b \mapsto \gamma \left(\begin{array}{l} a \mapsto \overleftarrow{\alpha}[e](1_B) \\ b \mapsto \overrightarrow{\gamma}[d](\overleftarrow{\alpha}[f](1_C)) \end{array} \right) \end{array} \right\} \qquad \overleftarrow{\gamma}[b](\gamma \left(\begin{array}{l} c \mapsto \overrightarrow{\beta}[a](1_B) \\ d \mapsto \overrightarrow{\beta}[b](1_C) \end{array} \right)$$

Composing these two processes on $\gamma : Z$ gives

$$\alpha \left\{ \begin{array}{l} a \mapsto \gamma \left(\begin{array}{l} a \mapsto \overleftarrow{\alpha}[c](1_A) \\ b \mapsto \overrightarrow{\gamma}[c](\overleftarrow{\alpha}[d](1_B)) \end{array} \right) \\ b \mapsto \gamma \left(\begin{array}{l} a \mapsto \overleftarrow{\alpha}[e](1_B) \\ b \mapsto \overrightarrow{\gamma}[d](\overleftarrow{\alpha}[f](1_C)) \end{array} \right) \end{array} \right\} ;_\gamma \overleftarrow{\gamma}[b](\gamma \left(\begin{array}{l} c \mapsto \overrightarrow{\beta}[a](1_B) \\ d \mapsto \overrightarrow{\beta}[b](1_C) \end{array} \right)$$

$$\Longrightarrow \alpha \left\{ \begin{array}{l} a \mapsto \gamma \left(\begin{array}{l} a \mapsto \overleftarrow{\alpha}[c](1_A) \\ b \mapsto \overrightarrow{\gamma}[c](\overleftarrow{\alpha}[d](1_B)) \end{array} \right) ;_\gamma \overleftarrow{\gamma}[b](\gamma \left(\begin{array}{l} c \mapsto \overrightarrow{\beta}[a](1_B) \\ d \mapsto \overrightarrow{\beta}[b](1_C) \end{array} \right) \\ b \mapsto \gamma \left(\begin{array}{l} a \mapsto \overleftarrow{\alpha}[e](1_B) \\ b \mapsto \overrightarrow{\gamma}[d](\overleftarrow{\alpha}[f](1_C)) \end{array} \right) ;_\gamma \overleftarrow{\gamma}[b](\gamma \left(\begin{array}{l} c \mapsto \overrightarrow{\beta}[a](1_B) \\ d \mapsto \overrightarrow{\beta}[b](1_C) \end{array} \right) \end{array} \right\} \quad \text{(by (3))}$$

$$\Longrightarrow \alpha \left\{ \begin{array}{l} a \mapsto \overrightarrow{\gamma}[c](\overleftarrow{\alpha}[d](1_B));_\gamma \gamma \left(\begin{array}{l} c \mapsto \overrightarrow{\beta}[a](1_B) \\ d \mapsto \overrightarrow{\beta}[b](1_C) \end{array} \right) \\ b \mapsto \overrightarrow{\gamma}[d](\overleftarrow{\alpha}[f](1_C));_\gamma \gamma \left(\begin{array}{l} c \mapsto \overrightarrow{\beta}[a](1_B) \\ d \mapsto \overrightarrow{\beta}[b](1_C) \end{array} \right) \end{array} \right\} \quad \text{(by (12))}$$

$$
\begin{array}{lrcl}
(1) & f;_\gamma 1 & \Longrightarrow & f \\
(2) & 1;_\gamma f & \Longrightarrow & f \\
(3) & \alpha\{a_i \mapsto f_i\}_i;_\gamma g & \Longrightarrow & \alpha\{a_i \mapsto f_i;_\gamma g\}_i \\
(4) & f;_\gamma \beta(b_i \mapsto g_i)_i & \Longrightarrow & \beta(b_i \mapsto f;_\gamma g_i)_i \\
(5) & \overrightarrow{\alpha}[a_k](f);_\gamma g & \Longrightarrow & \overrightarrow{\alpha}[a_k](f;_\gamma g) \\
(6) & f;_\gamma \overleftarrow{\beta}[b_k](g) & \Longrightarrow & \overleftarrow{\beta}[b_k](f;_\gamma g) \\
(7) & \overleftarrow{\alpha}[a_k](f);_\gamma g & \Longrightarrow & \overleftarrow{\alpha}[a_k](f;_\gamma g) \\
(8) & f;_\gamma \overrightarrow{\beta}[b_k](g) & \Longrightarrow & \overrightarrow{\beta}[b_k](f;_\gamma g) \\
(9) & \alpha(a_i \mapsto f_i)_i;_\gamma g & \Longrightarrow & \alpha(a_i \mapsto f_i;_\gamma g)_i \\
(10) & f;_\gamma \beta\{b_i \mapsto g_i\}_i & \Longrightarrow & \beta\{b_i \mapsto f;_\gamma g_i\}_i \\
(11) & \overrightarrow{\gamma}[a_k](f);_\gamma \gamma\{a_i \mapsto g_i\}_i & \Longrightarrow & f;_\gamma g_k \\
(12) & \gamma(a_i \mapsto f_i)_i;_\gamma \overleftarrow{\gamma}[a_k](g) & \Longrightarrow & f_k;_\gamma g \\
(13) & \alpha\{a_i \mapsto \beta\{b_j \mapsto f_{ij}\}_j\}_i & \Longleftrightarrow & \beta\{b_j \mapsto \alpha\{a_i \mapsto f_{ij}\}_i\}_j \\
(14) & \alpha(a_i \mapsto \beta(b_j \mapsto f_{ij})_j)_i & \Longleftrightarrow & \beta(b_j \mapsto \alpha(a_i \mapsto f_{ij})_i)_j \\
(15) & \alpha\{a_i \mapsto \overrightarrow{\beta}[b_k](f_i)\}_i & \Longleftrightarrow & \overrightarrow{\beta}[b_k](\alpha\{a_i \mapsto f_i\}_i) \\
(16) & \overleftarrow{\alpha}[a_k](\beta(b_i \mapsto g_i)_i) & \Longleftrightarrow & \beta(b_i \mapsto \overleftarrow{\alpha}[a_k](g_i))_i \\
(17) & \alpha\{a_i \mapsto \overleftarrow{\beta}[b_k](f_i)\}_i & \Longleftrightarrow & \overleftarrow{\beta}[b_k](\alpha\{a_i \mapsto f_i\}_i) \\
(18) & \overrightarrow{\alpha}[a_k](\beta(b_i \mapsto g_i)_i) & \Longleftrightarrow & \beta(b_i \mapsto \overrightarrow{\alpha}[a_k](g_i))_i \\
(19) & \alpha\{a_i \mapsto \beta(b_j \mapsto f_{ij})_j\}_i & \Longleftrightarrow & \beta(b_j \mapsto \alpha\{a_i \mapsto f_{ij}\}_i)_j \\
(20) & \overrightarrow{\alpha}[a_k](\overrightarrow{\beta}[b_l](f)) & \Longleftrightarrow & \overrightarrow{\beta}[b_l](\overrightarrow{\alpha}[a_k](f)) \\
(21) & \overleftarrow{\alpha}[a_k](\overleftarrow{\beta}[b_l](f)) & \Longleftrightarrow & \overleftarrow{\beta}[b_l](\overleftarrow{\alpha}[a_k](f)) \\
(22) & \overrightarrow{\alpha}[a_k](\overleftarrow{\beta}[b_l](f)) & \Longleftrightarrow & \overleftarrow{\beta}[b_l](\overrightarrow{\alpha}[a_k](f))
\end{array}
$$

Table 3.3: $\Sigma\Pi$ conversion rules

$$\Longrightarrow \alpha \left\{ \begin{array}{l} a \mapsto \overleftarrow{\alpha}[d](1_B);_\gamma \gamma(\overrightarrow{\beta}[a](1_B)) \\ b \mapsto \overleftarrow{\alpha}[f](1_C);_\gamma \gamma(\overrightarrow{\beta}[b](1_C)) \end{array} \right\} \quad \text{(by (11))}$$

$$\Longrightarrow \alpha \left\{ \begin{array}{l} a \mapsto \overleftarrow{\alpha}[d](1_B;_\gamma \gamma(\overrightarrow{\beta}[a](1_B))) \\ b \mapsto \overleftarrow{\alpha}[f](1_C;_\gamma \gamma(\overrightarrow{\beta}[b](1_C))) \end{array} \right\} \quad \text{(by (7))}$$

$$\Longrightarrow \alpha \left\{ \begin{array}{l} a \mapsto \overleftarrow{\alpha}[d](\overrightarrow{\beta}[a](1_B)) \\ b \mapsto \overleftarrow{\alpha}[f](\overrightarrow{\beta}[b](1_C)) \end{array} \right\} \quad \text{(by (2))}$$

It is easy to see the resulting process is indeed a process between $\alpha : X$ and $\beta : Y$.

3.3.3 A "programming language" representation

In this section we introduce the second of our term representations for derivations in $\Sigma\Pi$: the "programming language" representation. This representation was suggested to Robin Cockett and the author by Robert Seely.

The term formation rules for this representation are given in Table 3.4. Notice that the cut rule in this representation does not require that the channel names be the same, however, channels may still have to be renamed (substituted) after cutting two processes together to keep the channel names distinct. It is also worth noting that this representation does not differentiate between input/output on a domain channel and input/output on a codomain channel; the typing disambiguates between the two.

Example 3.3.4 (Programming language representation)

1. The following is the programming language representation of the first example from Example 3.3.1.

```
input on α of
   | a ↦ output g on β then
      output a′ on γ then
         input on δ of
```

$$\frac{}{\alpha : A \vdash_{1_A} \beta : A}$$

$$\frac{\{\Gamma, \alpha : X_i \vdash_{f_i} \Delta\}_{i \in I}}{\Gamma, \alpha : \sum_{i \in I} a_i : X_i \vdash_{\texttt{input on } \alpha \texttt{ of } \mid_{i \in I} a_i \mapsto f_i} \Delta} \qquad \frac{\{\Gamma \vdash_{f_i} \alpha : X_i, \Delta\}_{i \in I}}{\Gamma \vdash_{\texttt{input on } \alpha \texttt{ of } \mid_{i \in I} a_i \mapsto f_i} \alpha : \prod_{i \in I} a_i : X_i, \Delta}$$

$$\frac{\Gamma, \alpha : X_k \vdash_f \Delta}{\Gamma, \alpha : \prod_{i \in I} a_i : X_i \vdash_{\texttt{output } a_k \texttt{ on } \alpha \texttt{ then } f} \Delta} \qquad \frac{\Gamma \vdash_f \alpha : X_k, \Gamma}{\Gamma \vdash_{\texttt{output } a_k \texttt{ on } \alpha \texttt{ then } f} \alpha : \sum_{i \in I} a_i : X_i, \Gamma}$$

where $k \in I, I \neq \emptyset$

$$\frac{\Gamma \vdash_f \Delta, \alpha : Z \quad \beta : Z, \Gamma' \vdash_g \Delta'}{\Gamma, \Gamma' \vdash_{\texttt{plug } \alpha \texttt{ in } f \texttt{ to } \beta \texttt{ in } g} \Delta, \Delta}$$

Table 3.4: ΣΠ programming language term formation rules

```
                | c' ↦ output c on α then i
                | d' ↦ output d on α then j
                          output e' on δ then j
    | b ↦ output h on β then
      output b' on γ then
        input on δ of
          | c' ↦ output e on α then k
          | d' ↦ output f on α then l
                    output f' on δ then l
```

2. This is the second example from Example 3.3.1.

```
input on α of
  | a ↦ output c on β then
      input on β of
        | g ↦ output e on α then 1_A
        | h ↦ output f on α then 1_B
  | b ↦ output d on β then 1_C
```

As this representation is self-dual, many of the cut-elimination reductions and permuting conversions are identical. Therefore, only the unique rewrites are presented; the rewrites which are identical will be indicated in brackets, [_] (the dual rewrites are left out).

$$(1)\quad \texttt{plug}\ \gamma\ \texttt{in}\ f\ \texttt{to}\ \delta\ \texttt{in}\ 1 \Longrightarrow f$$

$$(3), [(9)]\quad \texttt{plug}\ \gamma\ \texttt{in}\ (\texttt{input on}\ \alpha\ \texttt{of}\ \ |_i\ a_i \mapsto f_i)\ \texttt{to}\ \delta\ \texttt{in}\ g$$
$$\Longrightarrow \texttt{input on}\ \alpha\ \texttt{of}\ \ |_i\ a_i \mapsto (\texttt{plug}\ \gamma\ \texttt{in}\ f_i\ \texttt{to}\ \delta\ \texttt{in}\ g)$$

$$(5), [(7)]\quad \texttt{plug}\ \gamma\ \texttt{in}\ (\texttt{output}\ a_k\ \texttt{on}\ \alpha\ \texttt{then}\ f)\ \texttt{to}\ \delta\ \texttt{in}\ g$$
$$\Longrightarrow \texttt{output}\ a_k\ \texttt{on}\ \alpha\ \texttt{then}\ (\texttt{plug}\ \gamma\ \texttt{in}\ f\ \texttt{to}\ \delta\ \texttt{in}\ g)$$

$$(11)\quad \texttt{plug}\ \gamma\ \texttt{in}\ (\texttt{output}\ a_k\ \texttt{on}\ \gamma\ \texttt{then}\ f)$$
$$\texttt{to}\ \gamma\ \texttt{in}\ (\texttt{input on}\ \gamma\ \texttt{of}\ \ |_i\ a_i \mapsto g_i)$$
$$\Longrightarrow \texttt{plug}\ \gamma\ \texttt{in}\ f\ \texttt{to}\ \gamma\ \texttt{in}\ g_k$$

$$(13), [(19)]\quad \texttt{input on}\ \alpha\ \texttt{of}\ \ |_i\ a_i \mapsto (\texttt{input on}\ \beta\ \texttt{of}\ \ |_j\ b_j \mapsto f_{ij})$$
$$\Longleftrightarrow \texttt{input on}\ \beta\ \texttt{of}\ \ |_j\ b_j \mapsto (\texttt{input on}\ \alpha\ \texttt{of}\ \ |_i\ a_i \mapsto f_{ij})$$

$$(15), [(17)]\quad \texttt{input on}\ \alpha\ \texttt{of}\ \ |_i\ a_i \mapsto (\texttt{output}\ b_k\ \texttt{on}\ \beta\ \texttt{then}\ f_i)$$
$$\Longleftrightarrow \texttt{output}\ b_k\ \texttt{on}\ \beta\ \texttt{then}\ (\texttt{input on}\ \alpha\ \texttt{of}\ \ |_i\ a_i \mapsto f_i)$$

$$(20), [(22)]\quad \texttt{output}\ a_k\ \texttt{on}\ \alpha\ \texttt{then}\ (\texttt{output}\ b_k\ \texttt{on}\ \beta\ \texttt{then}\ f_i)$$
$$\Longleftrightarrow \texttt{output}\ b_k\ \texttt{on}\ \beta\ \texttt{then}\ (\texttt{output}\ a_k\ \texttt{on}\ \alpha\ \texttt{then}\ f_i)$$

The following example of cut-elimination is the "programming language representation" of Example 3.3.3.

- We begin with the following process.

```
plug γ in
       input on α of
          | a ↦ input on γ of
                | a ↦ output c on α then 1_A
```

```
                | b ↦ output c on γ then
                        output d on α then 1_B
            | b input on γ of
                | a ↦ output e on α then 1_B
                | b ↦ output d on γ then
                        output f on α then 1_C
to γ in
        output b on γ then
            input on γ of
                | c ↦ output a on β then 1_B
                | d ↦ output b on β then 1_C
```

- Applying rewrite (3) yields.

```
input on α of
   | a ↦ plug γ in
            input on γ of
               | a ↦ output c on α then 1_A
               | b ↦ output c on γ then
                        output d on α then 1_B
         to γ in
            output b on γ then
               input on γ of
                  | c ↦ output a on β then 1_B
                  | d ↦ output b on β then 1_C
   | b ↦ plug γ in
            input on γ of
               | a ↦ output e on α then 1_B
               | b ↦ output d on γ then
                     output f on α then 1_C
         to γ in
            output b on γ then
```

```
            input on γ of
               | c ↦ output a on β then 1_B
               | d ↦ output b on β then 1_C
```

- To this result, applying rewrite (12) (the dual of (11)) yields.

```
input on α of
   | a ↦ plug γ in
            output c on γ then
               output d on α then 1_B
         to γ in
            input on γ of
               | c ↦ output a on β then 1_B
               | d ↦ output b on β then 1_C
   | b ↦ plug γ in
            output d on γ then
               output f on α then 1_C
         to γ in
            input on γ of
               | c ↦ output a on β then 1_B
               | d ↦ output b on β then 1_C
```

- Rewrite (11).

```
input on α of
   | a ↦ plug γ in
            output d on α then 1_B
         to γ in
            output a on β then 1_B
   | b ↦ plug γ in
            output f on α then 1_C
         to γ in
            output b on β then 1_C
```

- Rewrite (7).

```
input on α of
   | a ↦ output d on α then
      plug γ in 1_B
      to γ in
         output a on β then 1_B
   | b ↦ output f on α then
      plug γ in 1_C
      to γ in
         output b on β then 1_C
```

- And finally, by rewrite (2).

```
input on α of
   | a ↦ output d on α then
      output a on β then 1_B
   | b ↦ output f on α then
      output b on β then 1_C
```

Although this syntax gives a clear intuition into processes, the reader will agree that it is quite verbose. Thus, in what follows, we shall favor the compact term calculus introduced in Section 3.3.1.

3.4 The additive units

The way in which the cut-elimination process handles the reductions and permuting conversions when the index set $I = \emptyset$ is quite subtle. To clarify this, in this section we make these special cases explicit. In the following the abbreviations $\sum_{\emptyset} = 0$ and $\prod_{\emptyset} = 1$ will be used. The nullary versions of the cotuple and tuple rules are:

$$\frac{}{\Gamma, \alpha : 0 \vdash_{\alpha\{\}} \Delta}\ (\text{cotuple}) \qquad \frac{}{\Gamma \vdash_{\beta()} \beta : 1, \Delta}\ (\text{tuple})$$

The notation here is ambiguous as one cannot derive the context from the terms. To correct this we shall write the terms above as $\alpha\{\}_{\Gamma\vdash\Delta}$ and $\beta()_{\Gamma\vdash\Delta}$ respectively.

There are four reductions that are relevant to this setting, corresponding to the rewrites (3), (4), (9), and (10) where $I = \emptyset$. Given terms $f : \Gamma \vdash \Delta, \gamma : X$ and $g : \gamma : X, \Gamma' \vdash \Delta'$ we have the following reductions.

$$\begin{aligned}
(3) \quad & \alpha\{\}_{\Gamma\vdash\Delta,\gamma:X} \;;_\gamma g \implies \alpha\{\}_{\Gamma,\Gamma'\vdash\Delta,\Delta'} \\
(4) \quad & f \;;_\gamma \beta()_{\gamma:X,\Gamma'\vdash\Delta'} \implies \beta()_{\Gamma,\Gamma'\vdash\Delta,\Delta'} \\
(9) \quad & \alpha()_{\Gamma\vdash\Delta,\gamma:X} \;;_\gamma g \implies \alpha()_{\Gamma,\Gamma'\vdash\Delta,\Delta'} \\
(10) \quad & f \;;_\gamma \beta\{\}_{\gamma:X,\Gamma'\vdash\Delta'} \implies \beta\{\}_{\Gamma,\Gamma'\vdash\Delta,\Delta'}
\end{aligned}$$

In addition, there are 13 permuting conversions, corresponding to the cases (15) and (17) (and their duals), three variants of (13) (corresponding to the cases when only $I = \emptyset$, only $J = \emptyset$, and both $I = J = \emptyset$) (and their duals), and three variants of (19). Fortunately, they are all very similar, so we present only (15) and the three variants of (19). Here we drop the typing on the term and indicate it in the brackets.

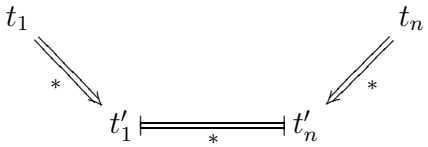
$$\begin{aligned}
(15) \quad & \alpha\{\} \;\Longrightarrow\; \overrightarrow{\beta}[b_k](\alpha\{\}) && (\Gamma, \alpha : 0 \vdash \beta : \textstyle\sum Y_i, \Delta) \\
(19) \quad & \alpha\{a_i \mapsto \beta()\}_i \;\Longrightarrow\; \beta() && (\Gamma, \alpha : \textstyle\sum X_i \vdash \beta : 1, \Delta) \\
(19) \quad & \alpha\{\} \;\Longrightarrow\; \beta(b_j \mapsto \alpha\{\})_j && (\Gamma, \alpha : 0 \vdash \beta : \textstyle\prod Y_j, \Delta) \\
(19) \quad & \alpha\{\} \;\Longrightarrow\; \beta() && (\Gamma, \alpha : 0 \vdash \beta : 1, \Delta)
\end{aligned}$$

3.5 Proof of the Church-Rosser property

In this section we present a proof of the Church-Rosser property for ΣΠ-morphisms. The proof presented here follows very closely the proof in [CS01], extended to the "poly" case.

We wish to show that given any two ΣΠ-morphisms related by a series of reductions and permuting conversions

there is an alternative way of arranging the reductions and permuting conversions so that t_1 and t_n can be reduced to terms which are related by the permuting conversions alone. That is, we wish to show that there is a convergence of the following form:



When the rewriting system terminates (in the appropriate sense) this allows the decision procedure for the equality of $\Sigma\Pi$-terms to be reduced to the decision procedure for the permuting conversions (see Section 3.7). In order to test the equality of two terms, one can rewrite both terms into a reduced form (one from which there are no further reductions), and these will be equal if and only if the two reduced forms are equivalent through the permuting conversions alone. In the current situation the reduction process is the cut-elimination procedure. In this section we will show that this is a terminating procedure.

We begin with a couple of definitions. The first is the multiset ordering of Dershowitz and Manna [DM79]. Let $(S, \succ)$ be a partially ordered set, and let $\mathcal{M}(S)$ denote the multisets (or bags) over S. For $M, N \in \mathcal{M}(S)$, $M > N$ ("$>$" is called the **multiset** (or **bag**) **ordering**), if there are multisets $X, Y \in \mathcal{M}(S)$, where $\emptyset \neq X \subseteq M$, such that

$$N = (M \backslash X) \cup Y \quad \text{and} \quad (\forall y \in Y)(\exists x \in X)\ x \succ y$$

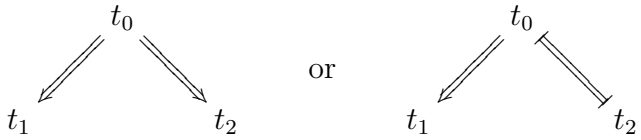
where $\cup$ here is the multiset union.

For example,

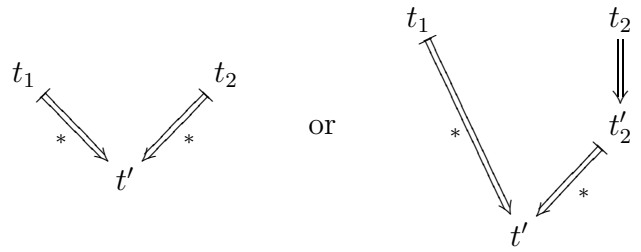
$$[3] > [2, 2, 1, 1], \quad [4, 3] > [4], \quad [3, 2] > [3, 1]$$

Recall from [DM79] that if $(S, \succ)$ is a total order (linear order) then $\mathcal{M}(S)$ is a total order. To see this consider $M, N \in \mathcal{M}(S)$. To determine whether $M > N$ sort the elements of both M and N and then compare the two sorted sequences lexicographically.

Following [CS01] we say a rewrite system is **locally confluent modulo equations** if any (one step) divergence of the following form



(where "$\Longrightarrow$" denotes a reduction and "$\models\!\!=$" an equation) has a convergence, respectively, of the form



where the new arrow "$\models\!\!\Longrightarrow$" indicates either an equality or a reduction in the indicated direction.

Proposition 3.5.1 *Suppose $(N, \mathcal{R}, \mathcal{E})$ is a rewriting system with the equations equipped with a well-ordered measure on the rewrite arrows such that the measure of the divergences is strictly greater than the measure of the convergences then the system is confluent modulo equations if and only if it is locally confluent modulo equations.*

Proof. If the system is confluent modulo equations it is certainly locally confluent modulo equations. Conversely suppose we have a chain of reductions, equations, and expansions. We may associate with it the bag of measures of the arrows of the sequence.

The idea will be to show that replacing any local divergence in this chain by a local confluence will result in a new chain whose bag measure is strictly smaller.

However, this can be seen by inspection as we are removing the arrows associated with the divergence and replacing them with the arrows associated with the convergence. The measure on the arrows associated with the divergence is strictly greater then that of the measure on the arrows associated with the convergence.

Thus, each rewriting reduces the measure and, therefore, any sequence of rewriting on such a chain must terminate. However, it can only terminate when there are no local divergences to resolve. This then implies that the end result must be a confluence modulo equations. ∎

3.5.1 Resolving critical pairs locally

The proof of the Church-Rosser property involves examining all the possible critical pairs involving reductions or reductions and conversions, and showing that they are all of the form shown above and that they may be resolved in the way shown above. It then must be shown that there is some measure on the arrows which decreases when replacing a divergence with a convergences. This will then suffice to show that our system is locally confluent modulo equations, so that by Proposition 3.5.1, it is confluent modulo equations. The rewrites (1)-(12) are the "reductions" and the permuting conversions (13)-(22) are the "equations".

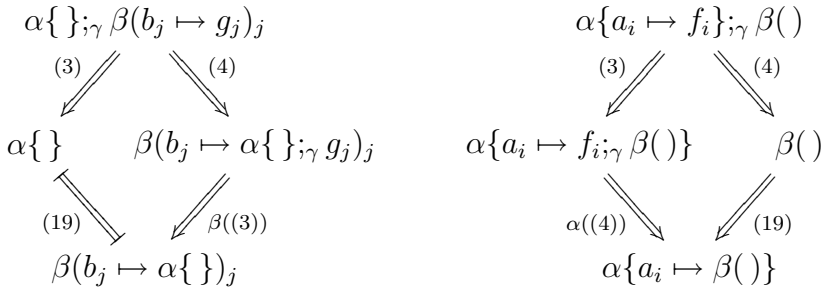
The resolutions of the critical pairs will be presented as reduction diagrams. We begin with the rewrites involving the empty cotuple and empty tuple rules.

- The resolution of the critical pair (1)-(3) (dually (2)-(4)) is indicated by the following reduction diagram.

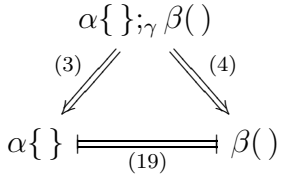
  $$\alpha\{\,\};_\gamma 1 \underset{(3)}{\overset{(1)}{\Longrightarrow}} \alpha\{\,\}$$

  The critical pair (1)-(9) is handled similarly.

- There are three cases for the resolution of the critical pair (3)-(4): only $I = \emptyset$, only $J = \emptyset$, both $I = J = \emptyset$. The first two cases correspond respectively to the following reductions diagrams:

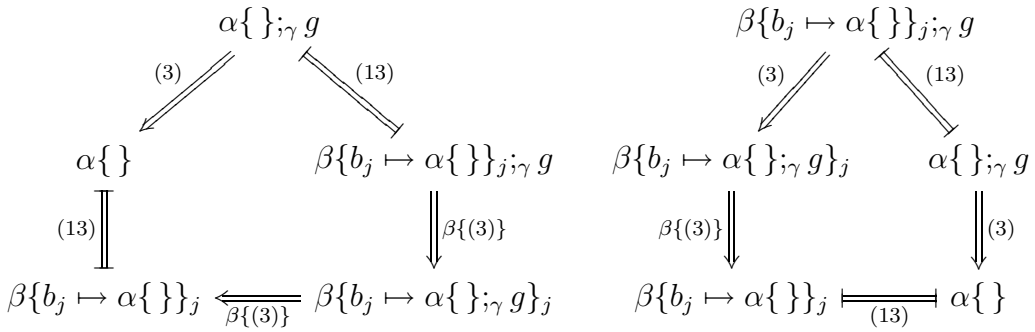


while the reduction diagram for the third case is:

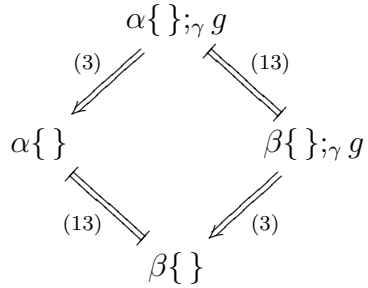


The rest of the critical pairs involving reductions from the empty cotuple and empty tuple rules will have similar reductions. So, we now look at the cases where there is a critical pair involving a reduction and a conversion.

- There are three cases for the resolution of the critical pair (3)-(13) (dually (4)-(14): only $I = \emptyset$, only $J = \emptyset$, both $I = J = \emptyset$. The first two are dual so we present one the first and third case. In the first case there are two subcases. There correspond to whether the apex (of the reduction diagram) starts with $\alpha\{\,\}$ or $\beta\{b_j \mapsto \alpha\{\,\}\}_j$. The reduction diagrams for these subcases are as follows:

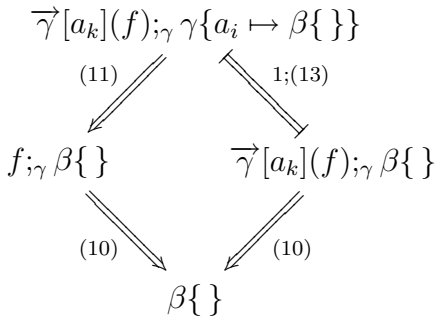


In the third case we have



The rest of the critical pairs involving reductions and critical pairs from the empty cotuple and empty tuple rules where neither operate on the cut channel will have similar reductions. The case where the terms operate on the cut channel must now be looked at.

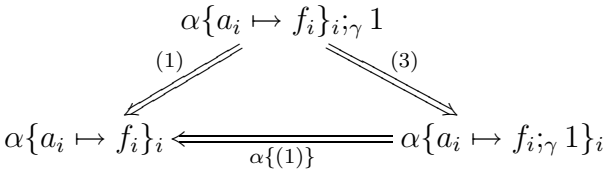
- The resolution of the critical pair (11)-(13) (dually (12)-(14)) is indicated by the following reduction diagram:



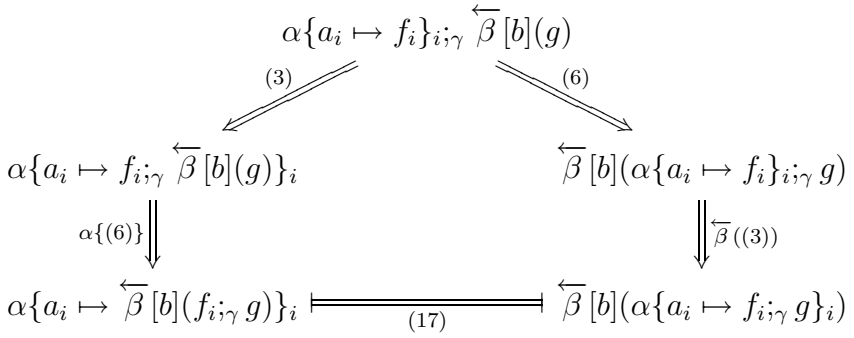
The rest of the critical pairs involving reductions and critical pairs from the empty cotuple and empty tuple rules where they operate on the cut channel will have similar reductions.

We now move on to examining the cases for non-empty index sets. In this case each of the reduction diagrams has one of five shapes. Due to the number of critical pairs, we limit ourselves to presenting one of each shape here, and the rest in Appendix A.

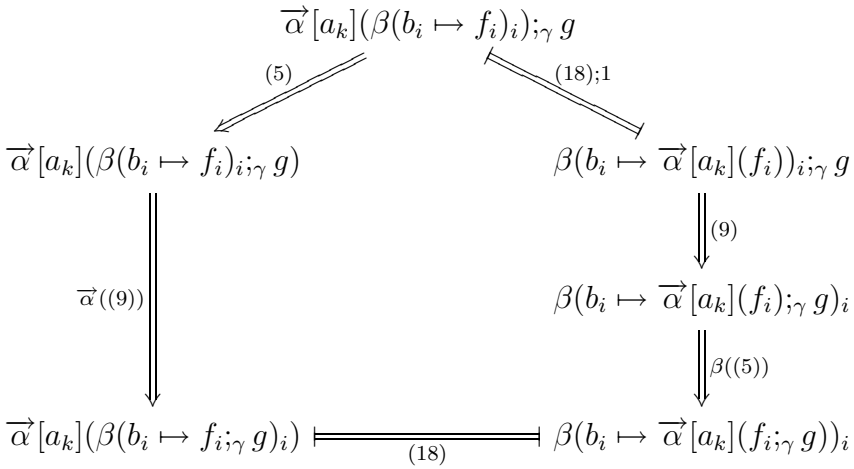
- The resolution of the critical pair (1)-(3) (dually (2)-(4)) is indicated by the following reduction diagram.



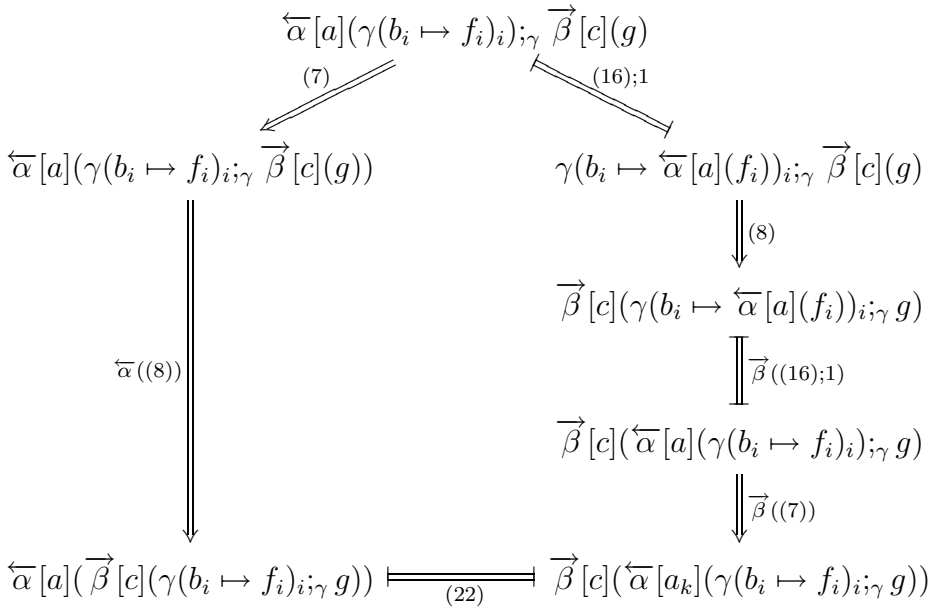
- The resolution of the critical pair (3)-(6) (dually (4)-(5)) is indicated by the following reduction diagram.



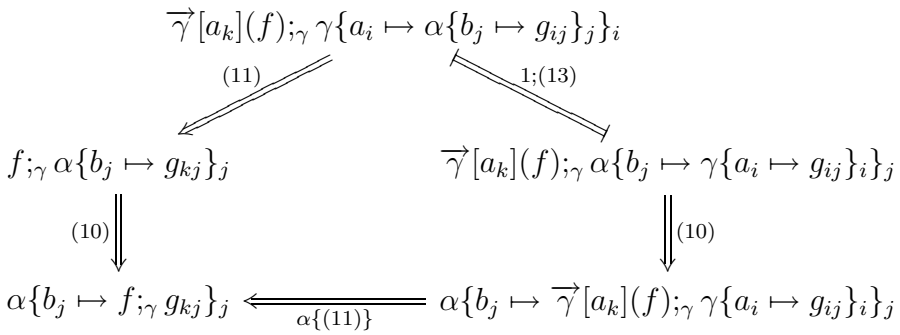
- The resolution of the critical pair (5)-(18) (dually (6)-(17)), where $\beta \neq \gamma$, is indicated by the following reduction diagram.



- The resolution of the critical pair (7)-(16) (dually (8)-(15)), where we start with the morphism $\overleftarrow{\alpha}[a](\gamma(b_i \mapsto f_i)_i);_\gamma \overrightarrow{\beta}[c](g)$, is indicated by the following reduction diagram.



- The resolution of the critical pair (11)-(13) (dually (12)-(13)) is indicated by the following reduction diagram.



Notice that each of the five shapes of reduction diagram fits the required form to show local confluence modulo equations, and hence, confluence modulo equations.

The next step in proving Church-Rosser is to show that the reduction steps terminate. This is done by associating a bag of cut costs with a sequent derivation and showing that each reduction strictly reduces the bag while each equality leaves

it stationary. This then will imply that the reductions terminate. The construction of this cost criterion is our next task.

3.6 The cut measure on $\Sigma\Pi$-morphisms

The purpose of this section is to show that the cut elimination procedure (defined in Section 2.2) terminates. To this end we define a bag of cut heights and show that the bag is strictly reduced on each of the cut elimination rewrites.

We begin by defining the **height** of a term as:

- $\mathrm{hgt}[a] = 1$ when a is an atomic map (or an identity)
- $\mathrm{hgt}[\alpha\{a_i \mapsto f_i\}_{i\in I}] = 1 + \max\{\mathrm{hgt}[f_i] \mid i \in I\}$
- $\mathrm{hgt}[\overline{\alpha}[a_k] \cdot f] = 1 + \mathrm{hgt}[f]$
- $\mathrm{hgt}[f; g] = \mathrm{hgt}[f] + \mathrm{hgt}[g]$

The **height of a cut** is defined simply as its height, e.g., $\mathrm{cuthgt}[f; g] = \mathrm{hgt}[f; g]$. Define a function $\Lambda : T \to \mathrm{bag}(\mathbb{N})$ which takes a term to its bag of cut heights.

Proposition 3.6.1

(i) *If* $t_1 \Longrightarrow t_2$ *then* $\Lambda(t_1) > \Lambda(t_2)$.

(ii) *If* $t_1 \overset{(a)}{\Longrightarrow\!\!\!\!\mid} t_2$ *and* (a) *is an interchange which does not involve the nullary cotuple or tuple then* $\Lambda(t_1) = \Lambda(t_2)$.

Proof. We begin with the proof of part (i). There are three properties that must be shown: $\mathrm{hgt}[t_1] \geq \mathrm{hgt}[t_2]$, the height of each non-principal cut does not increase, and the height of any cut produced from the principal cut is strictly less than the height of the principal cut.

A simple examination of the rewrites will confirm that if $t_1 \Longrightarrow t_2$ then $\mathrm{hgt}[t_1] \geq \mathrm{hgt}[t_2]$:

(1) (and dually (2)):

$$\mathrm{hgt}[f;1] = \mathrm{hgt}[f] + \mathrm{hgt}[1] > \mathrm{hgt}[f]$$

(3) (and similarly (4), (9), and (10)):

$$\begin{aligned}\mathrm{hgt}[\alpha\{a_i \mapsto f_i\}_{i\in I};_\gamma g] &= \mathrm{hgt}[\alpha\{a_i \mapsto f_i\}_{i\in I}] + \mathrm{hgt}[g] \\ &= 1 + \max\{\mathrm{hgt}[f_i] \mid i \in I\} + \mathrm{hgt}[g] \\ &= 1 + \max\{\mathrm{hgt}[f_i] + \mathrm{hgt}[g] \mid i \in I\} \\ &= \mathrm{hgt}[\alpha\{a_i \mapsto f_i;_\gamma g\}_{i\in I}]\end{aligned}$$

If the index set I is empty we have

$$\begin{aligned}\mathrm{hgt}[\alpha\{\};_\gamma g] &= \mathrm{hgt}[\alpha\{\}] + \mathrm{hgt}[g] \\ &= 1 + \mathrm{hgt}[g] \\ &> \mathrm{hgt}[\alpha\{\}]\end{aligned}$$

(5) (and similarly (6), (7), and (8)):

$$\begin{aligned}\mathrm{hgt}[\overrightarrow{\alpha}[a_k](f);_\gamma g] &= \mathrm{hgt}[\overrightarrow{\alpha}[a_k](f)] + \mathrm{hgt}[g] \\ &= 1 + \mathrm{hgt}[f] + \mathrm{hgt}[g] \\ &= \mathrm{hgt}[\overrightarrow{\alpha}[a_k](f;_\gamma g)]\end{aligned}$$

(11) (and dually (12)):

$$\begin{aligned}\mathrm{hgt}[\overrightarrow{\gamma}[a_k](f);_\gamma \gamma\{a_i \mapsto g_i\}_{i\in I}] &= \mathrm{hgt}[\overrightarrow{\gamma}[a_k](f)] + \mathrm{hgt}[\gamma\{a_i \mapsto g_i\}_{i\in I}] \\ &= 1 + \mathrm{hgt}[f] + 1 + \max\{\mathrm{hgt}[g_i] \mid i \in I\} \\ &> \mathrm{hgt}[f] + \max\{\mathrm{hgt}[g_i] \mid i \in I\} \\ &\geq \mathrm{hgt}[f] + \mathrm{hgt}[g_k] \\ &= \mathrm{hgt}[f;_\gamma g_k]\end{aligned}$$

Moreover, this implies that cuts below and cuts above the redex will not increase their cut height on a rewriting.

Finally, consider the principal cut of the reduction. Rewrite (1) (dually (2)) removes a cut and so strictly reduces the bag of cut heights. It is an easy observation that (5), (7), and (11) (and their duals) each replace a cut with one of lesser height, and that (3) (and its dual) replace a cut with zero or more cuts of lesser height. Thus applying any of the rewrites strictly reduces the bag.

We know prove part (ii). For the equations (13) through (19) we assume that the index sets are non-empty. This then implies that the permuting conversions are all of the form $\alpha(\beta(f))$ and thus

$$\mathrm{hgt}[\alpha(\beta(f))] = 1 + \mathrm{hgt}[\beta(f)] = 1 + 1 + \mathrm{hgt}[f] = \mathrm{hgt}[\beta(\alpha(f))]$$

which proves that the height does not change across these (non-empty tuple and cotuple) interchanges. ∎

To see that the height is not invariant across the empty cotuple (dually the tuple) rule recall one of the nullary versions of the rewrite (13):

$$\alpha\{\,\} \Longmapsto \beta\{b_j \mapsto \alpha\{\,\}\}_j$$

The height on the left-hand side is one, while on the right-hand side the height is two.

3.6.1 The measure on the rewriting arrows

We define a measure $\lambda : A \to \mathrm{bag}(\mathbb{N})$ on the rewriting arrows as follows:

- if $t_1 \overset{x}{\Longrightarrow} t_2$ then $\lambda(x) = \min\{\Lambda(t_1),\ \Lambda(t_2)\}$
- if $t_1 \overset{x}{\Longmapsto} t_2$ then $\lambda(x) = \max\{\Lambda(t_1),\ \Lambda(t_2)\}$

where $\Lambda(t)$ is the bag of cut heights of t.

A quick examination of the reduction diagrams now confirms that this measure will decrease when we replace a divergence with a convergence.

This completes the proof of the proposition:

Proposition 3.6.2 $\Sigma\Pi_{\mathbf{A}}$ *under the rewrites (1)-(12) is confluent modulo the equations (13)-(22).*

3.7 Deciding the $\Sigma\Pi$-conversions

From the above, it is clear that given any two derivations, deciding their equivalence reduces to deciding the equivalence of cut-free proofs. Of course, this means that any atomic cuts (cuts involving atomic formula) must be replaced with the atomic sequent given by the appropriate composition in the generating category **A**. Thus, the decision procedure is a relative one depending on the decision procedure for **A**.

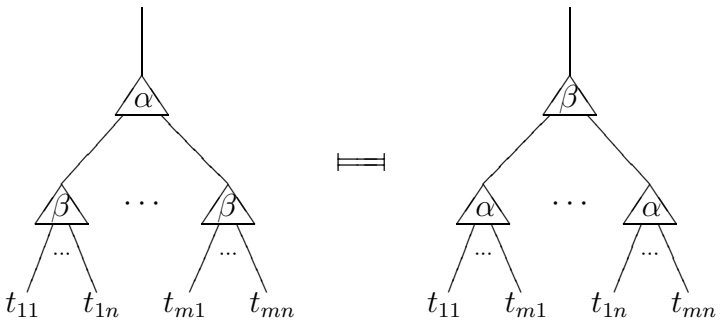
The decision procedure presented for the $\Sigma\Pi$-conversions is graphically-inspired. It operates on pairs of terms representing cut-free derivations of a given sequent. One of the terms is used as a "template" for transforming the other term into one of the same shape. The idea is to force the second term to start with the same proof rule as the template. If this is possible, then proceed inductively with the subterms, and otherwise, if it is not possible, the two terms must then not be equivalent. By using one of the terms in this manner one provides an order to search for the conversions which make the two terms the same.

This can be described using the term calculus or even the derivations themselves, but is clearer with a simple graphical representation of the terms. With a term we can associate a term-graph, whose nodes represent the subterms of the term. Tupling and cotupling will be denoted with a triangle decorated with the channel name it operates on, which has "output" edges for each component of the (co)tuple. These output edges will be decorated with the event associated with each component. The typing of the term will indicate which rule (tuple or cotuple) each triangle represents. Injections and projections will be denoted by boxes decorated with the channel name it operates on. The output edge of an injection/projection will be decorated by the event in which it outputs. Similarly here, the typing of the term determines whether a box represents an injection or a projection. Atomic sequents will be represented by circular nodes containing the atomic term, as will identities

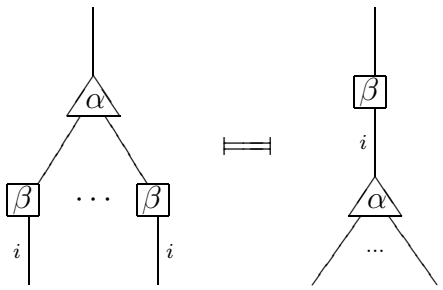
on atomic formulas.

With these conventions the permuting conversions may be represented by the following graph equivalences (where $\alpha \neq \beta$).

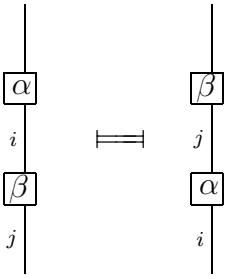
- Cotuple-cotuple, cotuple-tuple



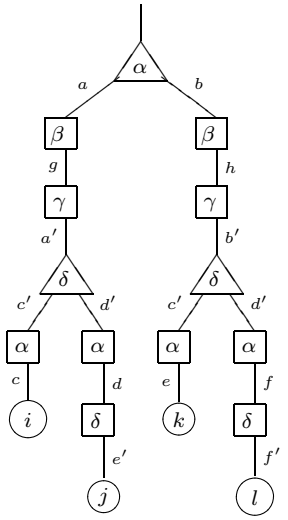
- Cotuple-injection, cotuple-projection, tuple-injection, tuple-projection



- Injection-injection, injection-projection, projection-projection



To illustrate the graphical representation, the third derivation in Example 2.1.1 can be represented by the following graph. Note that the graph is quite a direct representation of the derivation tree.

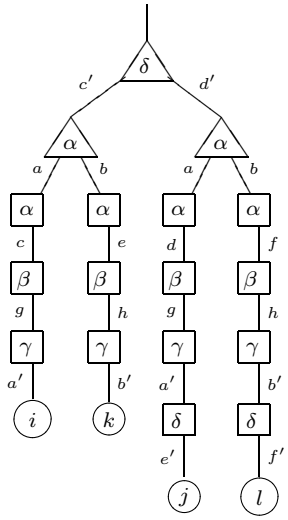


An equivalent derivation is given as follows

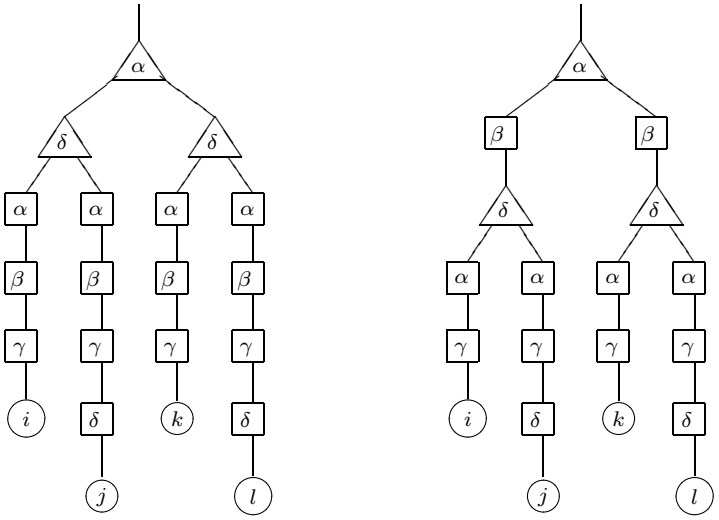
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A,\ E \vdash_i G,\ I}}{A,\ E \vdash G+H,\ I}}{A,\ E \times F \vdash G+H,\ I}}{A \times B, E \times F \vdash G+H, I} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{C,\ F \vdash_k H,\ I}}{C,\ F \vdash G+H,\ I}}{C,\ E \times F \vdash G+H,\ I}}{C \times D, E \times F \vdash G+H, I}}{(A \times B)+(C \times D),\ E \times F \vdash G+H,\ I} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B,\ F \vdash_j G,\ J}}{B,\ F \vdash G,\ J+K}}{B,\ F \vdash G+H,\ J+K}}{B,\ E \times F \vdash G+H,\ J+K}}{A \times B, E \times F \vdash G+H, J+K} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{D,\ F \vdash_l H,\ K}}{D,\ F \vdash H,\ J+K}}{D,\ F \vdash G+H,\ J+K}}{D,\ E \times F \vdash G+H,\ J+K}}{C \times D, E \times F \vdash G+H, J+K}}{(A \times B)+(C \times D),\ E \times F \vdash G+H,\ J+K}$$

$$(A \times B)+(C \times D),\ E \times F \vdash G+H,\ I \times (J+K)$$

which is given (in its annotated version) by the graph below:



We shall illustrate the decision procedure with this example. Take the first graph as a template. The first step in the procedure is to see if the second graph can start the same way as the first graph. This means we have to move an α-triangle up to the topmost level. This involves searching through the second graph until an α-triangle is found that can be moved upwards in the necessary manner. In this case there is one at the second level. Moving it up gives the graph on the left below. (The labels have been removed as this will not cause any ambiguity here, but in any case they may be inferred from the previous graph.)



Moving down a level we inductively repeat the process for all subterms at the second level nodes: the next step is to pull up the β-boxes on the left path and right path. This is done by interchanging the α-boxes and β-boxes in all the paths and then flipping the δ-triangles and β-boxes, which results in the graph on the right above. The last step would be to bring the γ-boxes up to the third level. This is done by interchanging the α-boxes and the γ-boxes in all the paths and then flipping the δ-triangles and γ-boxes; this produces the required graph (i.e., we were able to transform the second graph into the "template" graph), and so completes the proof that the two original derivations are equivalent. In general, the decision procedure will proceed in this recursive manner.

3.7.1 The details and proof of the decision procedure

A term is α**-inert** if it does not contain as subterms $\overleftarrow{\alpha}[a](t)$, $\overrightarrow{\alpha}[a](t)$, $\alpha\{a_i \mapsto t_i\}_{i\in I}$, or $\alpha(a_i \mapsto t_i)_{i\in I}$. Clearly if t is α-inert then there is no equality involving α which applies to it.

Let C_α be the constructors $\overleftarrow{\alpha}[\cdot](\_)$, $\overrightarrow{\alpha}[\cdot](\_)$, $\alpha\{\_\}_{i\in I}$, or $\alpha(\_)_{i\in I}$. We shall say a term starts with constructor C_α in case the first constructor in the term is C_α.

The C_α**-prefix** of a term t, denoted $\mathrm{prefix}_{C_\alpha}[t]$, is defined as follows.

- If t starts with C_α then $\mathrm{prefix}_{C_\alpha}[t] = *$ (where $*$ is the "anonymous" variable, i.e., a distinct variable which has not been used before and will not be used again).
- If t does not start with C_α then
  - if $t = \overleftarrow{\beta}[b](t')$ then $\mathrm{prefix}_{C_\alpha}[\overleftarrow{\beta}[b](t')] = \overleftarrow{\beta}[b](\mathrm{prefix}_{C_\alpha}[t'])$.
  - if $t = \overrightarrow{\beta}[b](t')$ then $\mathrm{prefix}_{C_\alpha}[\overrightarrow{\beta}[b](t')] = \overrightarrow{\beta}[b](\mathrm{prefix}_{C_\alpha}[t'])$.
  - if $t = \beta\{b_i \mapsto t_i\}_{i\in I}$ then
  $$\mathrm{prefix}_{C_\alpha}[\beta\{b_i \mapsto t_i\}_{i\in I}] = \beta\{b_i \mapsto \mathrm{prefix}_{C_\alpha}[t_i]\}_{i\in I}.$$
  - if $t = \beta(b_i \mapsto t_i)_{i\in I}$ then
  $$\mathrm{prefix}_{C_\alpha}[\beta(b_i \mapsto t_i)_{i\in I}] = \beta(b_i \mapsto \mathrm{prefix}_{C_\alpha}[t_i])_{i\in I}.$$

The C_α**-frontier** of a term with a C_α-prefix is those first occurrences across the term of the constructor C_α.

Lemma 3.7.1 *Suppose a term t starts with constructor C_α. Then in any series of equalities*

$$t \models\!=\!\models t_1 \models\!=\!\models t_2 \models\!=\!\models \cdots \models\!=\!\models t_n$$

the C_α-prefix of each t_i is α-inert.

Proof. Suppose that $t = \overleftarrow{\alpha}[a](t')$ and $\mathrm{prefix}_{\overleftarrow{\alpha}(\_)}[t_i] = w$, where w is α-inert. Either $t_i \models\!\!=\!\!\mid t_{i+1}$ is an application of an equality at the $\overleftarrow{\alpha}[\cdot](\_)$-frontier of the inert term w or it is not. If it is beyond the frontier then $\mathrm{prefix}_{\overleftarrow{\alpha}[\cdot](\_)}[t_i] = \mathrm{prefix}_{\overleftarrow{\alpha}[\cdot](\_)}[t_{i+1}]$ and if it is before the frontier then it simple shuffles the α-inert prefix. If it is on the frontier either it moves structure out of the inert term by shrinking the frontier (in which case $\mathrm{prefix}_{\overleftarrow{\alpha}[\cdot](\_)}[t_{i+1}]$ is certainly still inert (if smaller)), or it moves structure into the prefix by expanding the frontier. However, only α-inert structure can be moved over $\overleftarrow{\alpha}[\cdot](\_)$, so again $\mathrm{prefix}_{\overleftarrow{\alpha}[\cdot](\_)}[t_{i+1}]$ is α-inert. ∎

In a series of equalities beginning with a term which starts with C_α we may distinguish the steps which increase the C_α-inert prefix, $t_i \models\!\!\Longrightarrow t_{i+1}$, those which decrease the C_α-inert prefix, $t_i \Longleftarrow\!\!\mid t_{i+1}$, and those which do not affect the C_α-inert prefix, $t_i \models\!\!=\!\!\mid t_{i+1}$.

Lemma 3.7.2

$$t_i \models\!\!\overset{x}{\Longrightarrow} t_{i+1} \models\!\!\overset{y}{=}\!\!\mid t_{i+2}$$

can be rearranged as

$$t_i \models\!\!\overset{y}{=}\!\!\mid t'_{i+1} \models\!\!\overset{x}{\Longrightarrow} t_{i+2}$$

Proof. The redex of y cannot be within the inert prefix, nor by assumption is it on the frontier, and thus, it must be independent of x (on the frontier), and thus, the equalities can be rearranged. ∎

This means that we can rearrange the steps in any proof of equality so that no α-inert prefix-increasing step happens before a step which does not affect the inert prefix. However, we are not able to move these increasing steps past an inert prefix-decreasing step, but as a decreasing step is only possible if there has already been the corresponding (reverse) increasing step, this decreasing step may be cancelled with the increasing step. Thus, we may conclude the following.

Lemma 3.7.3 *In any series of steps*

$$t \models\!\!\underset{*}{=}\!\!\mid t_1 \models\!\!\overset{x}{\underset{*}{\Longrightarrow}} t_2 \Longleftarrow\!\!\overset{y}{\mid} t_3$$

the decreasing step y can be cancelled with its corresponding increasing step.

The following proposition now follows.

Proposition 3.7.4 *Any proof of equality from t to t' can be rearranged as*

$$t \overset{x}{\underset{*}{\longmapsto\!\!\!\dashv}} t_1 \overset{y}{\underset{*}{\Longmapsto}} t'$$

where the initial equalities do not touch the root constructor.

Note that the prefix-increasing steps y are essentially unique. There may be independent expansions of the inert frontier which can be reordered but every equivalent proof must do the same expansions. The point is that reading these steps in reverse gives a procedure which pulls the root structure of t to the root of t'.

Corollary 3.7.5 *The structure C_α may be pulled to the root of t if and only if the C_α-prefix of t is α-inert.*

Proof. It has already been observed that such a pulling up process results in an inert prefix (of the appropriate sort). Conversely given an inert prefix of the appropriate sort clearly means that the C_α-frontier can be contracted shrinking the C_α-prefix. ∎

As the equalities in the first part of the proof, x, do not touch the root constructor each equality must apply to one of the arguments of that constructor. Thus, for each argument we then have an equality proof, but each of these proofs can now be "normalized" into the form of the corollary. This gives a normal form for (directed) equality proofs and whence an algorithm for determining equality which amounts to matching the structure of the first term t starting from the root and pulling up that structure to the root of the second term and then proceeding inductively with its subterms.

Chapter 4

Polycategorical Semantics of Processes

In this chapter we establish that $\Sigma\Pi_{\mathbf{A}}$ is the free polycategory with sums and products generated from an arbitrary polycategory $\mathbf{A}$. The derivations, when cut is eliminated and treated up to conversion, will be the morphisms of this polycategory.

In Section 4.2, we define and show that $\Sigma\Pi_{\mathbf{A}}$ has poly-sums and poly-products. Then, in Section 4.2.1, it is shown that the sums and products are "soft". Softness of the sum and products is necessary to establish the "Whitman Theorem" which characterizes the free polycategory with sums and products up to equivalence.

We begin by establishing that $\Sigma\Pi_{\mathbf{A}}$ is a polycategory.

4.1 $\Sigma\Pi_{\mathbf{A}}$ is a polycategory

The reduction rules and the permuting conversions together define an equivalence relation (which we denote by $\sim$) on derivations of a sequent. Our categorical semantics will have derivations modulo this equivalence as morphisms. Thus, the goal of this section is to prove:

Theorem 4.1.1 *$\Sigma\Pi_{\mathbf{A}}$ is a polycategory whose objects are the formulas of the logic, and whose morphisms are $\sim$-equivalence classes of derivations.*

We shall start by presenting some technical results about the derivations in $\Sigma\Pi_{\mathbf{A}}$.

A notion which will be useful throughout the rest of this thesis is that of an output sequent. A sequent $\Gamma \vdash \Delta$ is an **output sequent** in case:

- $X \in \Gamma$ implies that X is an atom or of the form $X = \prod_{i \in I} X_i$,
- $Y \in \Delta$ implies that Y is an atom or of the from $Y = \sum_{j \in J} Y_j$, and
- $\Gamma \cup \Delta$ contains at least one compound formula.

Proposition 4.1.2 *In $\Sigma\Pi_{\mathbf{A}}$:*

(i) *Any cut-free derivation of a sequent $\Gamma, \sum_i X_i \vdash \Delta$ is equivalent to one whose principal rule is a cotupling applied to $\sum_i X_i$.*

(ii) *Any cut-free derivation of a sequent $\Gamma \vdash \prod_i Y_i, \Delta$ is equivalent to one whose principal rule is a tupling applied to $\prod_i Y_i$.*

(iii) *Any cut-free derivation of an output sequent $\Gamma \vdash \Delta$ has as its principal rule either an injection or a projection.*

(iv) *Any cut-free derivation of a sequent $A_1, \ldots, A_m \vdash B_1, \ldots, B_n$, where the A_i's and B_j's are atoms, must be an axiom (i.e., a morphism of $\mathbf{A}$).*

Notice that this result can be extended to arbitrary derivations (not only cut-free ones) using the cut-elimination procedure. For example, any derivation of a sequent $\Gamma, \sum_i X_i \vdash \Delta$ can be transformed to one whose principal rule is cotupling.

Proof. Notice that (iii) and (iv) are immediate given which (non-cut) sequent rules may be applied in the respective situations. (ii) is the dual of (i) and thus, it is sufficient to prove (i).

- If $\sum_i X_i$ is the only compound formula (i.e., all the other formulas are atoms) then the cotuple rule is the only one which applies, so this must be the principal rule.

- If the principal rule is another cotupling

$$\frac{\{Y_j, \Gamma, \sum_i X_i \vdash \Delta\}_j}{\sum_j Y_j, \Gamma, \sum_i X_i \vdash \Delta}$$

  then, using our inductive hypothesis, the proofs above may be transformed to have cotupling applied to $\sum_i X_i$ as their principal rule. This allows us to use the cotuple-cotuple interchange to transform the derivation so that cotupling applied to $\sum_i X_i$ is the principal rule for the original proof.

- If the principal rule is a tupling

$$\frac{\{\Gamma, \sum_i X_i \vdash Y_j, \Delta'\}_j}{\Gamma, \sum_i X_i \vdash \prod_j Y_j, \Delta'}$$

  then, using our inductive hypothesis, the proofs above may be transformed to have cotupling applied to $\sum_i X_i$ as their principal rule. This allows us to use the cotuple-tuple interchange to transform the derivation so that cotupling applied to $\sum_i X_i$ is the principal rule for the original proof.

- If the principal rule is a projection

$$\frac{Y_k, \Gamma, \sum_i X_i \vdash \Delta}{\prod_j Y_j, \Gamma, \sum_i X_i \vdash \Delta}$$

  then, using our inductive hypothesis, the proof above may be transformed to have cotupling applied to $\sum_i X_i$ as its principal rule. This allows us to use the cotuple-projection interchange to transform the derivation so that cotupling applied to $\sum_i X_i$ is the principal rule for the original proof.

- If the principal rule is an injection

$$\frac{\Gamma, \sum_i X_i \vdash Y_k, \Delta'}{\Gamma, \sum_i X_i \vdash \sum_j Y_j, \Delta'}$$

  then, using our inductive hypothesis, the proof above may be transformed to

have cotupling applied to $\sum_i X_i$ as its principal rule. This allows us to use the cotuple-injection interchange to transform the derivation so that cotupling applied to $\sum_i X_i$ is the principal rule for the original proof.

■

Lemma 2.3.1 has already shown that there are identity derivations which act in the correct manner. It is left to show that the composition given by cut satisfies the associativity and interchange laws.

Proposition 4.1.3 *Cut satisfies the associative law. That is, given sequents of the form*

$$\Gamma \vdash_f \Delta, \gamma : X \qquad \gamma : X, \Gamma' \vdash_g \Delta', \delta : Y \qquad \delta : Y, \Gamma'' \vdash_h \Delta''$$

the composites $(f;_\gamma g);_\delta h$ *and* $f;_\gamma (g;_\delta h)$ *are* $\sim$*-equivalent.*

Proof. By structural induction on f, g, and h; without loss of generality we may assume that f, g, and h are all cut-free.

(i) $f = 1_X$. In this case we have $(1_X;_\gamma g);_\delta h \Longrightarrow g;_\delta h$ and $1_X;_\gamma (g;_\delta h) \Longrightarrow g;_\delta h$ which are equivalent. The case where $h = 1_Y$ is dual.

(ii) $f = \alpha\{a_i \mapsto f_i\}_i$ and $\Gamma = \Phi, \alpha : \sum_i W_i$. Composing on the left first gives

$$(\alpha\{a_i \mapsto f_i\}_i;_\gamma g);_\delta h \Longrightarrow \alpha\{a_i \mapsto f_i;_\gamma g\}_i;_\delta h \Longrightarrow \alpha\{a_i \mapsto (f_i;_\gamma g);_\delta h\}_i$$

whereas composing on the right first gives

$$\alpha\{a_i \mapsto f_i\}_i;_\gamma (g;_\delta h) \Longrightarrow \alpha\{a_i \mapsto f_i;_\gamma (g;_\delta h)\}_i$$

which by induction are $\sim$-equivalent. Since any derivation of a sequent $\Gamma, \alpha : \sum_i W_i \vdash \Delta$ may be equivalently written to end with cotupling applied on α, this suffices. The case where $h = (b_j \mapsto h_j)_j$ and $\Delta'' = \prod_j W_j, \Psi$ is dual.

(iii) We now consider the case where f is a map to a coproduct. There are two cases to consider.

(a) $f = \overrightarrow{\alpha}[a_k](f')$, $\alpha \neq \gamma$, and $\Delta = \alpha : \sum_i W_i, \Phi$. Composing on the left first gives

$$(\overrightarrow{\alpha}[a_k](f');_\gamma g);_\delta h \Longrightarrow \overrightarrow{\alpha}[a_k](f';_\gamma g);_\delta h \Longrightarrow \overrightarrow{\alpha}[a_k]((f';_\gamma g);_\delta h)$$

whereas composing on the right first gives

$$\overrightarrow{\alpha}[a_k](f');_\gamma (g;_\delta h) \Longrightarrow \overrightarrow{\alpha}[a_k](f';_\gamma (g;_\delta h))$$

which by induction are $\sim$-equivalent. The case where $h = \overleftarrow{\beta}[b_k](h')$ with $\beta \neq \delta$ and $\Gamma'' = \Psi, \beta : \prod_j W_j$ is dual.

(b) $f = \overrightarrow{\gamma}[a_k](f')$ and $X = \gamma : \sum_i X_i$. Here we may suppose that $g = \gamma\{a_i \mapsto g_i\}_i$ (as X is a sum) and so composing on the left first gives

$$(\overrightarrow{\gamma}[a_k](f');_\gamma \gamma\{a_i \mapsto g_i\}_i);_\delta h \Longrightarrow (f';_\gamma g_k);_\delta h$$

whereas composing on the right first gives

$$\begin{aligned}\overrightarrow{\gamma}[a_k](f');_\gamma (\gamma\{a_i \mapsto g_i\}_i;_\delta h) &\Longrightarrow \overrightarrow{\gamma}[a_k](f');_\gamma \gamma\{a_i \mapsto g_i;_\delta h\}_i \\ &\Longrightarrow f';_\gamma (g_k;_\delta h)\end{aligned}$$

which by induction are $\sim$-equivalent. The case where $h = \overleftarrow{\delta}[b_k](h')$ and $Y = \delta : \prod_j Y_j$ is dual.

(iv) $f = \overleftarrow{\alpha}[a_k](f')$ and $\Gamma = \Phi, \alpha : \prod_i W_i$. Composing on the left first gives

$$(\overleftarrow{\alpha}[a_k](f');_\gamma g);_\delta h \Longrightarrow \overleftarrow{\alpha}[a_k](f';_\gamma g);_\delta h \Longrightarrow \overleftarrow{\alpha}[a_k]((f';_\gamma g);_\delta h)$$

whereas composing on the right first gives

$$\overleftarrow{\alpha}[a_k](f');_\gamma (g;_\delta h) \Longrightarrow \overleftarrow{\alpha}[a_k](f';_\gamma (g;_\delta h))$$

which by induction are $\sim$-equivalent. The case where $h = \overrightarrow{\beta}(h')$ and $\Delta'' = \beta :$

$\sum_j W_j, \Psi$ is dual.

(v) We now consider the case where f is a map to a product. There are a number of subcases to consider.

(a) $f = \alpha(a_i \mapsto f_i)_i$, $\alpha \neq \gamma$, and $\Delta = \alpha : \prod_i W_i, \Phi$. Composing on the left first gives

$$(\alpha(a_i \mapsto f_i)_i ;_\gamma g) ;_\delta h \Longrightarrow \alpha(a_i \mapsto f_i ;_\gamma g)_i ;_\delta h \Longrightarrow \alpha(a_i \mapsto (f_i ;_\gamma g) ;_\delta h)_i$$

whereas composing on the right first gives

$$\alpha(a_i \mapsto f_i)_i ;_\gamma (g ;_\delta h) \Longrightarrow \alpha(a_i \mapsto f_i ;_\gamma (g ;_\delta h))_i$$

which by induction are $\sim$-equivalent. Since any derivation of a sequent $\Gamma \vdash \alpha : \prod_i W_i, \Delta$ may be equivalently written to end with tupling applied on α, this suffices. The case where $h = \beta\{b_j \mapsto h_j\}_j$ with $\beta \neq \delta$ and $\Gamma'' = \Psi, \beta : \sum_j W_j$ is dual.

(b) $f = \gamma(a_i \mapsto f_i)_i$, $X = \gamma : \prod_i X_i$, and $g = \overleftarrow{\gamma}[a_k](g')$. Composing on the left first gives

$$(\gamma(a_i \mapsto f_i)_i ;_\gamma \overleftarrow{\gamma}[a_k](g')) ;_\delta h \Longrightarrow (f_k ;_\gamma g') ;_\delta h$$

whereas composing on the right first gives

$$\begin{aligned} \gamma(a_i \mapsto f_i)_i ;_\gamma (\overleftarrow{\gamma}[a_k](g') ;_\delta h) &\Longrightarrow \gamma(a_i \mapsto f_i)_i ;_\gamma \overleftarrow{\gamma}[a_k](g' ;_\delta h) \\ &\Longrightarrow f_k ;_\gamma (g' ;_\delta h) \end{aligned}$$

which by induction are $\sim$-equivalent. Since any derivation of a sequent $\Gamma \vdash \alpha : \prod_i W_i, \Delta$ may be equivalently written to end with tupling applied on α, this suffices. The case where $h = \delta\{b_j \mapsto h_j\}_j$ and $Y = \delta : \sum_j Y_j$ is dual.

(c) $f = \gamma(a_i \mapsto f_i)_i$, $X = \gamma : \prod_i X_i$, and g operates on a channel $\beta \neq \gamma$. In this case we must explore the structure of g. First, let us assume $g = \beta\{b_j \mapsto g_j\}_j$. Composing on the left first gives

$$\begin{aligned}(\gamma(a_i \mapsto f_i)_i ;_\gamma \beta\{b_j \mapsto g_j\}_j) ;_\delta h &\Longrightarrow \beta\{b_j \mapsto \gamma(a_i \mapsto f_i)_i ;_\gamma g_j\}_j ;_\delta h \\ &\Longrightarrow \beta\{b_j \mapsto (\gamma(a_i \mapsto f_i)_i ;_\gamma g_j) ;_\delta h\}_j\end{aligned}$$

whereas composing on the right first gives

$$\begin{aligned}\gamma(a_i \mapsto f_i)_i ;_\gamma (\beta\{b_j \mapsto g_j\}_j ;_\delta h) &\Longrightarrow \gamma(a_i \mapsto f_i)_i ;_\gamma \beta\{b_j \mapsto g_j ;_\delta h\}_j \\ &\Longrightarrow \beta\{\gamma(a_i \mapsto f_i)_i ;_\gamma (b_j \mapsto g_j ;_\delta h)\}_j\end{aligned}$$

The remaining cases where g is either $\beta(b_j \mapsto g_j)_j$, $\overleftarrow{\beta}[b_k](g')$, or $\overrightarrow{\beta}[b_k](g')$ are all handled similarly.

This concludes all the essential cases. If all of f, g, and h are atomic then composition is associative because it is associative in the underlying polycategory. If some of f, g, and h are atomic a quick check of the possibilities will show that one ends up with a case essentially like one of the cases above.

■

For polycategories one must also prove that composition satisfies the interchange law.

Proposition 4.1.4 *Cut satisfies the interchange law. That is, given sequents of the form*

$$\Gamma \vdash_f \gamma : X, \Delta, \delta : Y \qquad \gamma : X, \Gamma' \vdash_g \Delta' \qquad \delta : Y, \Gamma'' \vdash_h \Delta''$$

the composites $(f ;_\gamma g) ;_\delta h$ and $(f ;_\gamma h) ;_\delta g$ are $\sim$-equivalent. Dually, given sequents of the form

$$\Gamma \vdash_f \gamma : X, \Delta \qquad \Gamma' \vdash_g \delta : Y, \Delta' \qquad \gamma : X, \Gamma'', \delta : Y \vdash_h \Delta''$$

then $f ;_\gamma (g ;_\delta h)$ and $g ;_\delta (f ;_\gamma h)$ are $\sim$-equivalent.

Proof. We prove the first statement by structural induction on f, g, and h, duality handles the second statement. Without loss of generality we may assume that f, g, and h are all cut-free.

(i) $f = \alpha\{a_i \mapsto f_i\}_i$ and $\Gamma = \Phi, \alpha : \sum_i W_i$. Composing with g first gives

$$(\alpha\{a_i \mapsto f_i\}_i ;_\gamma g) ;_\delta h \Longrightarrow \alpha\{a_i \mapsto f_i ;_\gamma g\}_i ;_\delta h \Longrightarrow \alpha\{a_i \mapsto (f_i ;_\gamma g) ;_\delta h\}_i$$

whereas composing with h first gives

$$(\alpha\{a_i \mapsto f_i\}_i ;_\delta h) ;_\gamma h \Longrightarrow \alpha\{a_i \mapsto f_i ;_\delta h\}_i ;_\gamma g \Longrightarrow \alpha\{a_i \mapsto (f_i ;_\delta h) ;_\gamma g\}_i$$

which, by induction, are $\sim$-equivalent. Since any derivation of a sequent $\Gamma, \sum_i W_i \vdash \Delta$ may be equivalently written to end with cotupling applied on α, this suffices.

(ii) We now look at the case where f is a map to a coproduct. There are three subcases to consider.

(a) $f = \overrightarrow{\gamma}[a_k](f')$ and $X = \gamma : \sum_i X_i$. Here we may suppose that $g = \gamma\{a_i \mapsto g_i\}_i$ (as X is a sum) and so, composing with g first gives

$$(\overrightarrow{\gamma}[a_k](f') ;_\gamma \gamma\{a_i \mapsto g_i\}_i) ;_\delta h \Longrightarrow (f' ;_\gamma g_k) ;_\delta h$$

whereas composing with h first gives

$$\begin{aligned}(\overrightarrow{\gamma}[a_k](f') ;_\delta h) ;_\gamma \gamma\{a_i \mapsto g_i\}_i &\Longrightarrow \overrightarrow{\gamma}[a_k](f' ;_\delta h) ;_\gamma \gamma\{a_i \mapsto g_i\}_i \\ &\Longrightarrow (f' ;_\delta h) ;_\gamma g_k\end{aligned}$$

which, by induction, are $\sim$-equivalent.

(b) $f = \overrightarrow{\alpha}[a_k](f')$, $\alpha \neq \gamma$, $\alpha \neq \delta$, and $\Delta = \Phi, \alpha : \sum_i W_i, \Psi$. Composing with

g first gives

$$(\overrightarrow{\alpha}[a_k](f');_\gamma g);_\delta h \Longrightarrow \overrightarrow{\alpha}[a_k](f';_\gamma g);_\delta h \Longrightarrow \overrightarrow{\alpha}[a_k]((f';_\gamma g);_\delta h)$$

whereas composing with h first gives

$$(\overrightarrow{\alpha}[a_k](f');_\delta h);_\gamma g \Longrightarrow \overrightarrow{\alpha}[a_k](f';_\delta h);_\gamma g \Longrightarrow \overrightarrow{\alpha}[a_k]((f';_\delta h);_\gamma g)$$

which, by induction, are $\sim$-equivalent.

(c) $f = \overrightarrow{\delta}[a_k](f')$ and $Y = \delta : \sum_i Y_i$. Here we may suppose that $h = \delta\{a_i \mapsto h_i\}_i$ (as Y is a sum) and so, composing with g first gives

$$\begin{aligned}(\overrightarrow{\gamma}[a_k](f');_\gamma g);_\delta \delta\{a_i \mapsto h_i\}_i &\Longrightarrow \overrightarrow{\gamma}[a_k](f';_\gamma g);_\delta \delta\{a_i \mapsto h_i\}_i \\ &\Longrightarrow (f';_\gamma g);_\delta h_k\end{aligned}$$

whereas composing with h first gives

$$(\overrightarrow{\gamma}[a_k](f');_\delta \delta\{a_i \mapsto h_i\}_i);_\gamma g \Longrightarrow (f';_\delta h_k);_\gamma g$$

which, by induction, are $\sim$-equivalent.

(iii) $f = \overleftarrow{\alpha}[a_k](f')$ and $\Gamma = \Phi, \alpha : \prod_i W_i$. Composing with g first gives

$$(\overleftarrow{\alpha}[a_k](f');_\gamma g);_\delta h \Longrightarrow \overleftarrow{\alpha}[a_k](f';_\gamma g);_\delta h \Longrightarrow \overleftarrow{\alpha}[a_k]((f';_\gamma g);_\delta h)$$

whereas composing with h first gives

$$(\overleftarrow{\alpha}[a_k](f');_\delta h);_\gamma g \Longrightarrow \overleftarrow{\alpha}[a_k](f';_\delta h);_\gamma g \Longrightarrow \overleftarrow{\alpha}[a_k]((f';_\delta h);_\gamma g)$$

which, by induction, are $\sim$-equivalent.

(iv) We now look at the case where f is a map to a product. Again, there are a

number of subcases to consider.

(a) $f = \alpha(a_i \mapsto f_i)_i,\ \alpha \neq \gamma,\ \alpha \neq \delta$, and $\Delta = \alpha : \prod_i W_i, \Phi$. Composing with g first gives

$$(\alpha(a_i \mapsto f_i)_i;_\gamma g);_\delta h \Longrightarrow \alpha(a_i \mapsto f_i;_\gamma g)_i;_\delta h \Longrightarrow \alpha(a_i \mapsto (f_i;_\gamma g);_\delta h)_i$$

whereas composing with h first gives

$$(\alpha(a_i \mapsto f_i)_i;_\delta h);_\gamma g \Longrightarrow \alpha(a_i \mapsto f_i;_\delta h)_i;_\gamma g \Longrightarrow \alpha(a_i \mapsto (f_i;_\delta h);_\gamma g)_i$$

which, by induction, are $\sim$-equivalent.

(b) $f = \gamma(a_i \mapsto f_i)_i,\ X = \gamma : \prod_i X_i$, and $g = \overleftarrow{\gamma}[a_k](g')$. Composing with g first gives

$$(\gamma(a_i \mapsto f_i);_\gamma \overleftarrow{\gamma}[a_k](g'));_\delta h \Longrightarrow (f_k;_\gamma g');_\delta h$$

whereas composing with h first gives

$$\begin{aligned}(\gamma(a_i \mapsto f_i)_i;_\delta h);_\gamma \overleftarrow{\gamma}[a_k](g') &\Longrightarrow \gamma(a_i \mapsto f_i;_\delta h)_i;_\gamma \overleftarrow{\gamma}[a_k](g') \\ &\Longrightarrow (f_k;_\delta h);_\gamma g'\end{aligned}$$

which, by induction, are $\sim$-equivalent.

(c) $f = \gamma(a_i \mapsto f_i)_i,\ X = \gamma : \prod_i X_i$, and g operates on a channel $\beta \neq \gamma$. In this case we must explore the structure of g. First, let us assume that $g = \beta\{b_j \mapsto g_j\}_j$. Composing with g first gives

$$\begin{aligned}(\gamma(a_i \mapsto f_i)_i;_\gamma \beta\{b_j \mapsto g_j\}_j);_\delta h &\Longrightarrow \beta\{b_j \mapsto \gamma(a_i \mapsto f_i)_i;_\gamma g_j\}_j;_\delta h \\ &\Longrightarrow \beta\{b_j \mapsto (\gamma(a_i \mapsto f_i)_i;_\gamma g_j);_\delta h\}_j\end{aligned}$$

whereas composing with h first gives

$$\begin{aligned}(\gamma(a_i \mapsto f_i)_i;_\delta h);_\gamma \beta\{b_j \mapsto g_j\}_j &\Longrightarrow \gamma(a_i \mapsto f_i;_\delta h)_i;_\gamma \beta\{b_j \mapsto g_j\}_j \\ &\Longrightarrow \beta\{b_j \mapsto \gamma(a_i \mapsto f_i;_\delta h)_i;_\gamma g_j\}_j\end{aligned}$$

which, by induction, are $\sim$-equivalent. The remaining cases on the structure of g are handled similarly.

(d) $f = \delta(a_i \mapsto f_i)_i$, $Y = \delta : \prod_i Y_i$ and $h = \overleftarrow{\delta}\,[a_k](h')$. Composing with g first gives

$$\begin{aligned}(\delta(a_i \mapsto f_i)_i;_\gamma g);_\delta \overleftarrow{\delta}\,[a_k](h') &\Longrightarrow \delta(a_i \mapsto f_i;_\gamma g)_i;_\delta \overleftarrow{\delta}\,[a_k](h') \\ &\Longrightarrow (f_k;_\gamma g);_\delta h'\end{aligned}$$

whereas composing with h first gives

$$(\delta(a_i \mapsto f_i)_i;_\delta \overleftarrow{\delta}\,[a_k](h'));_\gamma g \Longrightarrow (f_k;_\delta h');_\gamma g$$

which, by induction, are $\sim$-equivalent.

(e) $f = \delta(a_i \mapsto f_i)_i$, $Y = \delta : \prod_i Y_i$ and h operates on a channel $\beta \neq \delta$. In this case we must explore the structure of h. First, let us assume that $h = \beta\{b_j \mapsto h_j\}_j$. Composing with g first gives

$$\begin{aligned}(\gamma(a_i \mapsto f_i)_i;_\gamma g);_\delta \beta\{b_j \mapsto h_j\}_j &\Longrightarrow \gamma(a_i \mapsto f_i;_\gamma g)_i;_\delta \beta\{b_j \mapsto h_j\}_j \\ &\Longrightarrow \beta\{b_j \mapsto \gamma(a_i \mapsto f_i;_\gamma g)_i;_\delta h_j\}_j\end{aligned}$$

whereas composing with h first gives

$$\begin{aligned}(\gamma(a_i \mapsto f_i)_i;_\delta \beta\{b_j \mapsto h_j\}_j);_\gamma g &\Longrightarrow \beta\{b_j \mapsto \gamma(a_i \mapsto f_i)_i;_\delta h_j\}_j;_\gamma g \\ &\Longrightarrow \beta\{b_j \mapsto (\gamma(a_i \mapsto f_i)_i;_\delta h_j);_\gamma g\}_j\end{aligned}$$

which, by induction, are $\sim$-equivalent. The remaining cases on the structure of h are handled similarly.

Since any derivation of a sequent $\Gamma \vdash \alpha : \prod_i W_i, \Delta$ may be equivalently written to end with tupling applied on α, this suffices.

This concludes all the essential cases. If all of f, g, and h are atomic then composition satisfies the interchange property because it satisfies the interchange property in the underlying polycategory. If some of f, g, and h are atomic a quick check of the possibilities will show that one ends up with a case essentially like one of the cases above.

■

This now shows that $\Sigma\Pi_{\mathbf{A}}$ is a polycategory proving Theorem 4.1.1.

4.2 Poly-sums and poly-products

In this section we show that $\Sigma\Pi_{\mathbf{A}}$ is the free polycategory generated from (the polycategory) $\mathbf{A}$ under (finite) poly-sums and poly-products. We begin by defining sums and products in a polycategory.

In a polycategory $\mathbf{A}$, an object $\sum_{i\in I} X_i \in \mathbf{A}$ is said to be the **poly-sum** (or **poly-coproduct**) of a family of objects $X_i \in \mathbf{A}$, for $i \in I$, in case there is a poly-natural correspondence

$$(*) \qquad \frac{\{\Gamma, \alpha : X_i \vdash_{f_i} \Delta\}_{i\in I}}{\Gamma, \alpha : \sum_{i\in I} X_i \vdash_{\alpha\{f_i\}_i} \Delta}$$

where by poly-natural we mean that the following two equivalences

$$h;_\gamma \alpha\{f_i\}_i = \alpha\{h;_\gamma f_i\}_i \qquad \text{and} \qquad \alpha\{f_i\}_i;_\gamma h = \alpha\{f_i;_\gamma h\}_i$$

hold (when $\alpha \neq \gamma$). These equivalences assert that cutting on an object and then forming the coproduct is the same as first forming the coproduct and then performing the cut.

Products in polycategories are (as we expect) dual to coproducts. Explicitly, an object $\prod_{i\in I} X_i \in \mathbf{A}$ is said to be the **poly-product** of a family of objects $X_i \in \mathbf{A}$, for $i \in I$, in case there is a poly-natural correspondence

$$(\star) \qquad \frac{\{\Gamma \vdash_{f_i} \beta : X_i, \Delta\}_{i\in I}}{\Gamma \vdash_{\beta(f_i)_i} \beta : \prod_{i\in I} X_i, \Delta}$$

It will now be useful to define more "standard" injection and projection maps. The reader may have noticed that the injections and projections seem to be a little unfamiliar in their presentation.

$$\frac{\Gamma \vdash X_k, \Delta}{\Gamma \vdash \sum_i X_i, \Delta}\ \text{(injection)} \qquad\qquad \frac{\Gamma, X_k \vdash \Delta}{\Gamma, \prod_i X_i \vdash \Delta}\ \text{(projection)}$$

This may be "remedied" as follows. There are injection derivations $X_k \vdash_{b_k} \sum_i X_i$ for $k \in I$ given by $b_k = b_k(\iota_{X_k})$. With these more "standard" injections, the general injection terms may be identified with $f; b_k$. Note this is a valid identification, since there is a reduction of derivations

$$\frac{\Gamma \vdash X_k, \Delta \quad \dfrac{X_k \vdash X_k}{X_k \vdash \sum X_i}}{\Gamma \vdash \sum X_i, \Delta} \quad \Longrightarrow \quad \frac{\Gamma \vdash X_k, \Delta}{\Gamma \vdash \sum X_i, \Delta}$$

Dually, the general projection terms may be identified with $p_k; f$ where $\prod_i X_i \vdash_{p_k} X_k$.

Remark 4.2.1 The typing will sometimes be left off the injection/projection maps as there is only one formula on each side of the turnstile and hence, it may be inferred from the annotated composition symbol "$;_\gamma$". For example, if we write $b_k \;;_\gamma \Gamma, \gamma : \sum_i X_i \vdash \Delta$ it is clear that we are cutting on $\gamma : \sum_i X_i$.

We will now work with the poly-coproduct (the dual observations hold for the poly-product). The first thing to establish is the connection between the poly-coproduct defined here and the standard way of viewing a coproduct.

Proposition 4.2.2 *For a polycategory* $\mathbf{A}$ *the following are equivalent:*

(i) **A** *has poly-coproducts (for a set* I*).*

(ii) *There is an object* $\sum_i X_i$ *for each family of objects* $\{X_i\}_i$ *which has injection maps* $X_i \vdash_{b_i} \sum_i X_i$ *and a cotupling operation producing a unique map* $\Gamma, \gamma : \sum_i X_i \vdash_{\gamma\{f_i\}_i} \Delta$*, where* $\Gamma, \gamma : X_i \vdash_{f_i} \Delta$*, such that*

- $\{b_i\}_i = 1_{\sum_i X_i}$
- $b_i \;;_\gamma \gamma\{f_i\}_i = f_i$
- $\alpha\{f_i\}_i \;;_\gamma h = \alpha\{f_i \;;_\gamma h\}_i$*, where* $\alpha \neq \gamma$
- $h \;;_\gamma \alpha\{f_i\}_i = \alpha\{h \;;_\gamma f_i\}_i$*, where* $\alpha \neq \gamma$

Proof. To show that (i) implies (ii) consider the identity map,

$$\sum_i X_i \vdash_{1_{\sum_i X_i}} \sum_i X_i$$

By the definition of poly-coproducts, this gives

$$\frac{\{X_i \vdash_{b_i} \sum_i X_i\}_{i \in I}}{\sum_i X_i \vdash_{1_{\sum_i X_i}} \sum_i X_i}$$

and hence, $\{b_i\}_i = 1_{\sum_i X_i}$.

Now consider the composite

$$X_k \vdash_{b_k} \gamma : \sum_i X_i \quad ;_\gamma \quad \gamma : \sum_i X_i, \Gamma \vdash_{\gamma:\{f_i\}} \Delta$$

From the definition of poly-coproducts we have the following correspondence

$$\frac{\{X_i \vdash_{b_i} \gamma : \sum_i X_i \quad ;_\gamma \quad \gamma : \sum_i X_i, \Gamma \vdash_{\gamma\{f_i\}_i} \Delta\}_{i \in I}}{\sum_i X_i \vdash_{1_{\sum_i X_i}} \gamma : \sum_i X_i \quad ;_\gamma \quad \gamma : \sum_i X_i, \Gamma \vdash_{\gamma\{f_i\}_{i \in I}} \Delta}$$

so that $b_i \;;_\gamma \gamma\{f_i\}_i = f_i$.

Similarly, the following poly-natural correspondences

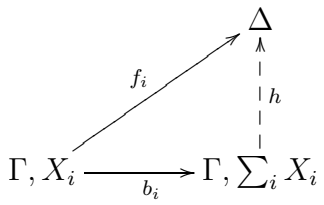
$$\frac{\{\Gamma, \alpha : X_i \vdash_{f_i} \Delta, \gamma : Z \quad ;_\gamma \quad \gamma : Z, \Gamma' \vdash_h \Delta'\}_{i \in I}}{\Gamma, \alpha : \sum_i X_i \vdash_{\alpha\{f_i\}_{i \in I}} \Delta, \gamma : Z \quad ;_\gamma \quad \gamma : Z, \Gamma' \vdash_h \Delta'}$$

and

$$\frac{\{\Gamma \vdash_h \Delta, \gamma : Z \quad ;_\gamma : \quad \gamma : Z, \Gamma', \alpha : X_i \vdash_{f_i} \Delta'\}_{i \in I}}{\Gamma \vdash_h \Delta, \gamma : Z \quad ;_\gamma \quad \gamma : Z, \Gamma', \alpha : \sum_i X_i \vdash_{\alpha\{f_i\}_{i \in I}} \Delta'}$$

show that $\alpha\{f_i\}_i \;;_\gamma h = \alpha\{f_i \;;_\gamma h\}_i$ and $h \;;_\gamma \alpha\{f_i\}_i = \alpha\{h \;;_\gamma f_i\}_i$.

The argument for the uniqueness of the comparison map



is $\{f_i\}_i = \{b_i \;; h\}_i = \{b_i\}_i \;; h = 1_{\sum_i X_i} \;; h = h$.

For the implication from (ii) to (i) we may immediately conclude the top-to-bottom direction of the correspondence from the definition of (ii). The argument for the bottom-to-top direction is as follows:

$$\begin{aligned} \gamma\{f_i\}_i &= 1_{\sum_i X_i} \;;_\gamma \gamma\{f_i\}_i \\ &= \alpha\{b_i\}_i \;;_\gamma \gamma\{f_i\}_i \\ &= \alpha\{b_i \;;_\gamma \gamma\{f_i\}_i\}_i \\ &= \alpha\{f_i\}_i \end{aligned}$$

Writing out the terms explicitly, this says that

$$\Gamma, \gamma : \textstyle\sum_i X_i \vdash_{\{f_i\}_i} \Delta = \{\Gamma, \alpha : X_i \vdash_{f_i} \Delta\}_i$$

This establishes both directions of the correspondence. That it is poly-natural follows from the definition (the last two conditions). ∎

The next two propositions show that $\Sigma\Pi_{\mathbf{A}}$ is the free polycategory generated from $\mathbf{A}$ with finite sums and products.

Proposition 4.2.3 *$\Sigma\Pi_{\mathbf{A}}$ has finite poly-sums and finite poly-products.*

Proof. In order to establish that $\Sigma\Pi_{\mathbf{A}}$ has finite poly-sums and finite poly-products

we must show that the inferences

$$\frac{\{\Gamma, \alpha : X_i \vdash_{f_i} \Delta\}_{i \in I}}{\Gamma, \alpha : \sum_{i \in I} i : X_i \vdash_{\alpha\{i \mapsto f_i\}_{i \in I}} \Delta} \qquad \text{and} \qquad \frac{\{\Gamma \vdash_{g_i} \beta : Y_j, \Delta\}_{j \in J}}{\Gamma \vdash_{\beta(j \mapsto g_j)_{j \in J}} \beta : \prod_{j \in J} j : Y_j, \Delta}$$

are two-way and poly-natural.

We begin with coproducts. Going from top-to-bottom is immediate via the cotupling derivation so we need only prove the other direction. By Proposition 4.1.2 we know that a sequent of the form $\Gamma, \sum_{i \in I} X_i \vdash \Delta$ may be written in a cut-free manner so that its principal rule is a cotupling which leaves us with the set of sequents $\{\Gamma, X_i \vdash \Delta\}_{i \in I}$. That this correspondence respects the equivalence relation $\sim$ is immediate from our decision procedure as once the cotuple structure is made principal, equality is determined by equality of the arguments. This proves the bottom-to-top direction.

It remains to show that this correspondence is poly-natural. However, this follows immediately from rewrite (3). Products are handled dually and therefore this establishes that $\Sigma\Pi_{\mathbf{A}}$ has products and coproducts. ∎

Proposition 4.2.4 *$\Sigma\Pi_{\mathbf{A}}$ is the free polycategory generated from $\mathbf{A}$ with finite sums and products.*

Proof. To show that $\Sigma\Pi_{\mathbf{A}}$ is the free polycategory generated from $\mathbf{A}$ with products and coproducts it suffices to show that all the identities (1)–(22) must hold in any polycategory with poly-products and poly-coproducts.

The identities (1) and (2) clearly hold. (3) and (9) (dually (4) and (10)) follow by poly-naturality, i.e.,

$$\frac{\{\Gamma, X_i \vdash_{f_i} \Delta, Z \;;\; Z, \Gamma' \vdash_g \Delta'\}_i}{\Gamma, \sum_i X_i \vdash_{\{f_i\}_i} \Delta, Z \;;\; Z, \Gamma' \vdash_g \Delta'}$$

(5) and (7) (dually (6) and (8)) follow from the interchange law, i.e.,

$$\frac{(\Gamma \vdash_f X_k, \Delta, Z \;;\; X_k \vdash_{b_k} \sum_i X_i) \;;\; Z, \Gamma' \vdash_g \Delta'}{(\Gamma \vdash_f X_k, \Delta, Z \;;\; Z, \Gamma' \vdash_g \Delta') \;;\; X_k \vdash_{b_k} \sum_i X_i}$$

(11) and dually (12) follow from the associativity of cut

$$\frac{(\Gamma \vdash_f \Delta, X_k,\ ;\ X_k \vdash_{b_k} \sum_i X_i)\ ;\ \Gamma', \sum_i X_i \vdash_{\{g_i\}_i} \Delta')}{\Gamma \vdash_f \Delta, X_k,\ ;\ (X_k \vdash_{b_k} \sum_i X_i\ ;\ \Gamma', \sum_i X_i \vdash_{\{g_i\}_i} \Delta')}$$

as $b_k\ ;\ \{g_i\}_i = g_k$.

(13) and dually (14) follow by the following argument (applications of poly-naturality):

$$\begin{aligned}
\{\{f_{ij}\}_i\}_j &= 1_{\sum_i X_i}\ ;\ \{\{f_{ij}\}_i\}_j \\
&= \{b_i\}_i\ ;\ \{\{f_{ij}\}_i\}_j \\
&= \{b_i\ ;\ \{\{f_{ij}\}_i\}_j\}_i \\
&= \{\{b_i\ ;\ \{f_{ij}\}_i\}_j\}_i \\
&= \{\{f_{ij}\}_j\}_i
\end{aligned}$$

A similar proof shows that (19) holds. (15) and (17) (dually (16) and (18)) follow by poly-naturality. Explicitly, (15) is as follows:

$$\frac{\{\Gamma, X_i \vdash_{f_i} \Delta, Y_k\ ;\ Y_k \vdash_{b_k} \sum_j Y_j\}_i}{\Gamma, \sum_i X_i \vdash_{\{f_i\}_i} \Delta, Y_k\ ;\ Y_k \vdash_{b_k} \sum_j Y_j}$$

(20) and (21) follow by the interchange law $(f\ ;\ b_k)\ ;\ b_l = (f\ ;\ b_l)\ ;\ b_k$, while (22) follows by associativity $p_k\ ;\ (f\ ;\ b_k) = (p_k\ ;\ f)\ ;\ b_k$. ∎

In the next section we show that the sums and products in $\Sigma\Pi_{\mathbf{A}}$ satisfy another important property.

4.2.1 Softness of poly-sums and poly-products

The purpose of this section is to characterize the free sum and product completion of a polycategory. We begin with a discussion of poly-hom-sets.

Given the types Γ and Δ, what does the poly-hom-set $\mathrm{Hom}(\Gamma;\Delta)$ look like? If the domain Γ contains a coproduct $\sum_{i\in I} X_i$, or the codomain Δ contains a product $\prod_{j\in J} Y_j$, then (by the inferences $(*)$ and $(\star)$) we may "break-up" the poly-hom-sets

$\mathrm{Hom}(\Gamma, \sum_{i\in I} X_i \ ; \ \Delta)$ and $\mathrm{Hom}(\Gamma \ ; \ \prod_{j\in J} Y_j, \Delta)$ into a product (in the category of sets) of poly-hom-sets, respectively,

$$\prod_{i\in I} \mathrm{Hom}(\Gamma, X_i \ ; \ \Delta) \quad \text{and} \quad \prod_{j\in J} \mathrm{Hom}(\Gamma \ ; \ Y_j, \Delta)$$

In the case where there are only products or atoms in the domain and coproducts or atoms in the codomain there is not in general a description of the poly-hom-set. In the free case for (non-poly) categories however there is a resolution due to Joyal [Joy95a, Joy95b, and other references], which he derived from Whitman's observation on lattices [Whi41]. Here we generalize these ideas to polycategories.

Define the **output index** of a pair of types (Γ, Δ), denoted $\mathfrak{I}(\Gamma; \Delta)$, as:

- if $\alpha : \prod_{i\in I} X_i \in \Gamma$ then the pairs (α, i), for $i \in I$, are in $\mathfrak{I}(\Gamma; \Delta)$.
- if $\beta : \sum_{j\in J} Y_j \in \Delta$ then the pairs (β, j), for $j \in J$, are in $\mathfrak{I}(\Gamma; \Delta)$.

For example, given

$$\Gamma = \alpha : A, \ \beta : \textstyle\prod_{i\in I} W_i, \ \gamma : \sum_{j\in J} X_j \qquad \text{and} \qquad \Delta = \delta : \textstyle\sum_{k\in K} Y_k, \ \epsilon : \prod_{l\in L} Z_l$$

where A is atomic, the output index $\mathfrak{I}(\Gamma; \Delta) = \{(\beta, i), (\delta, k) \mid i \in I, k \in K\}$.

Given a product $\alpha : \prod_i X_i$ on a domain channel and (α, k), $k \in I$, we will denote by

$$[\mathrm{Hom}(\Gamma, \alpha : \textstyle\prod_i X_i \ ; \ \Delta)]_{(\alpha,k)}$$

the poly-hom-set $\mathrm{Hom}(\Gamma, \alpha : X_k \ ; \ \Delta)$. The same notation will be used for a sum on a codomain channel. The following

$$\sum_{(\alpha,i)\in\mathfrak{I}(\Gamma;\Delta)} [\mathrm{Hom}(\Gamma \ ; \ \Delta)]_{(\alpha,i)}$$

will then be used to indicate the disjoint union of poly-hom-sets where each product in the domain and coproduct in the codomain has been "broken down" into its composite elements. For example, applying this construction to Γ and Δ as defined above results in the following set of poly-hom-sets:

$$\left\{\mathrm{Hom}(A, W_i, \textstyle\sum_{j\in J} X_j \ ; \ Y_k, \prod_{l\in L} Z_l) \mid i \in I, k \in K\right\}$$

Suppose $\mathbf{A}$ is a polycategory and $\mathbf{B}$ is a polycategory with sums and products such that $\mathbf{A} \xrightarrow{\mathcal{I}} \mathbf{B}$ is an inclusion morphism of polycategories. Consider a poly-hom-set $\mathrm{Hom}(\Gamma;\Delta)$ in $\mathbf{B}$ such that the output index $\mathfrak{I}(\Gamma;\Delta)$ is non-empty and any channel not in the output index is of the form $\mathcal{I}(A)$ where $A \in \mathbf{A}$. That is, each object of Γ is either "atomic" or a product, and each object of Δ is either "atomic" or a sum, so that this is the polycategorical analogue of an output sequent (which we call an **output poly-hom-set**). The morphism $\mathcal{I} : \mathbf{A} \to \mathbf{B}$ is called a **semi-soft extension** of polycategories if for any output poly-hom-set $\mathrm{Hom}(\Gamma;\Delta) \in \mathbf{B}$ the map

$$\sum_{(\alpha,i)\in\mathfrak{I}(\Gamma;\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{(\alpha,i)} \xrightarrow{\{\overline{\alpha}[i](\_)\}_{(\alpha,i)}} \mathrm{Hom}(\Gamma;\Delta)$$

is a surjection, where $\{\overline{\alpha}[i](\_)\}_{(\alpha,i)}$ is the cotupling map (in **Set**) and the underscore in $\overline{\alpha}[i](\_)$ is used to represent any map in $[\mathrm{Hom}(\Gamma;\Delta)]_{(\alpha,i)}$. (The overline $\overline{\alpha}$ is used to indicate that the map $\overline{\alpha}[i](\_)$ may represent an injection or a projection; it will depend upon the type of the α channel.)

In terms of processes, a semi-soft extension guarantees that every process in an output poly-hom-set $\mathrm{Hom}(\Gamma;\Delta)$ has as its next action an output event.

Example 4.2.5 (Semi-soft extension) Consider the simple output poly-hom-set $\mathrm{Hom}(\alpha : A \times B \ ; \ \beta : C + D)$. Dropping the channel names we have:

$$\sum_{(\gamma,k)\in\mathfrak{I}(A\times B;C+D)} \mathrm{Hom}(A \times B \ ; \ C + D) = \left\{ \begin{array}{l} \mathrm{Hom}(A \ ; \ C + D) \\ \mathrm{Hom}(B \ ; \ C + D) \\ \mathrm{Hom}(A \times B \ ; \ C) \\ \mathrm{Hom}(A \times B \ ; \ D) \end{array} \right\}$$

The (injection and projection) maps are

$$\begin{aligned}
\overleftarrow{\alpha}[1](_) &: \mathrm{Hom}(A\ ;\ C+D) \to \mathrm{Hom}(A\times B\ ;\ C+D)\\
\overleftarrow{\alpha}[2](_) &: \mathrm{Hom}(B\ ;\ C+D) \to \mathrm{Hom}(A\times B\ ;\ C+D)\\
\overrightarrow{\beta}[1](_) &: \mathrm{Hom}(A\times B\ ;\ C) \to \mathrm{Hom}(A\times B\ ;\ C+D)\\
\overrightarrow{\beta}[2](_) &: \mathrm{Hom}(A\times B\ ;\ D) \to \mathrm{Hom}(A\times B\ ;\ C+D)
\end{aligned}$$

so that considering them as a whole gives

$$\{\overline{\gamma}[k](\_)\}_{(\gamma,k)} : \sum_{(\gamma,k)} \mathrm{Hom}(A\times B; C+D) \to \mathrm{Hom}(A\times B; C+D)$$

where $(\gamma,k)\in\mathfrak{I}(A\times B, C+D)$.

The inclusion $\mathcal{I} : \mathbf{A}\to\mathbf{B}$ is called a **soft extension** in case for any output poly-hom-set $\mathrm{Hom}(\Gamma;\Delta)$ the following diagram

$$(\bigstar)\qquad
\begin{array}{ccc}
 & \sum\limits_{((\alpha,i),(\beta,j))\in\mathfrak{I}(\Gamma,\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{((\alpha,i),(\beta,j))} & \\
{\scriptstyle \sum\limits_{(\alpha,i)}\{\overline{\beta}[j](_)\}_{((\alpha,i),(\beta,j))}}\downarrow & & \downarrow{\scriptstyle \sum\limits_{(\beta,j)}\{\overline{\alpha}[i](_)\}_{((\alpha,i),(\beta,j))}} \\
 & \sum\limits_{(\gamma,k)\in\mathfrak{I}(\Gamma,\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{(\gamma,k)} & \\
 & \downarrow{\scriptstyle \{\overline{\gamma}[k](_)\}_{(\gamma,k)}} & \\
 & \mathrm{Hom}(\Gamma;\Delta) &
\end{array}$$

is a coequalizer diagram in the category of sets, where $((\alpha,i),(\beta,j))$ for $(\alpha,i),\ (\beta,j)\in\mathfrak{I}(\Gamma;\Delta)$, $\alpha\neq\beta$, represents choosing pairs from the members of $\mathfrak{I}(\Gamma;\Delta)$ and the map $\sum_{(\beta,j)}\{\overline{\alpha}[i](\_)\}_{((\alpha,i),(\beta,j))}$ represents the coproduct (of the β components) of the co-tupling maps of the α components, and vise versa when α and β are switched.

In terms of processes a soft extension means that given a process in $\mathrm{Hom}(\Gamma;\Delta)$ which is able to output on two distinct channels, the order in which one chooses

to do the outputs is irrelevant, e.g., the maps $\overleftarrow{\alpha}(\overrightarrow{\beta}(f))$ and $\overrightarrow{\beta}(\overleftarrow{\alpha}(f))$ will be $\sim$-equivalent.

Remark 4.2.6 In **Set**, given two parallel arrows $f, g : A \rightrightarrows B$, the coequalizer of this pair always exists (see, e.g., [BW99]). Thus, in order to show that the above diagram (★) is a coequalizer diagram we may assume that the coequalizer is given by (q, Q) and then show that the unique map $h : Q \to \mathrm{Hom}(\Gamma; \Delta)$ is an isomorphism, i.e., that the following diagram commutes:

$$\begin{array}{ccc}
\sum\limits_{((\alpha,i),(\beta,j))\in\mathfrak{I}(\Gamma,\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{((\alpha,i),(\beta,j))} & & \\
\Big\downarrow {\scriptstyle \sum\limits_{(\alpha,i)} \{\overline{\beta}[j](_)\}_{((\alpha,i),(\beta,j))}} \quad \Big\downarrow {\scriptstyle \sum\limits_{(\beta,j)} \{\overline{\alpha}[i](_)\}_{((\alpha,i),(\beta,j))}} & & \\
\sum\limits_{(\gamma,k)\in\mathfrak{I}(\Gamma,\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{(\gamma,k)} & & \\
\Big\downarrow {\scriptstyle q} \qquad \searrow {\scriptstyle \{\overline{\gamma}[k](_)\}_{(\gamma,k)}} & & \\
Q \underset{h}{\overset{\sim}{\dashrightarrow}} \mathrm{Hom}(\Gamma;\Delta) & &
\end{array}$$

Example 4.2.7 (Soft extension) Consider the poly-hom-set $\mathrm{Hom}(\alpha : A \times B \ ; \ \beta : C + D)$ from the example above and the following coequalizer diagram (again dropping the channel names):

$$D_1 = \begin{Bmatrix} \mathrm{Hom}(A \ ; \ C) & \mathrm{Hom}(B \ ; \ C) \\ \mathrm{Hom}(A \ ; \ D) & \mathrm{Hom}(B \ ; \ D) \end{Bmatrix}$$

$$\Big\downarrow {\scriptstyle \sum_\beta \overleftarrow{\alpha}} \qquad \Big\downarrow {\scriptstyle \sum_\alpha \overrightarrow{\beta}}$$

$$D_2 = \begin{Bmatrix} \mathrm{Hom}(A \times B \ ; \ C) & \mathrm{Hom}(A \ ; \ C + D) \\ \mathrm{Hom}(A \times B \ ; \ D) & \mathrm{Hom}(B \ ; \ C + D) \end{Bmatrix}$$

$$\Big\downarrow {\scriptstyle \overline{\gamma}}$$

$$\mathrm{Hom}(A \times B \ ; \ C + D)$$

where $\sum_\beta \overleftarrow{\alpha}$ and $\sum_\alpha \overrightarrow{\beta}$ will be constructed in what follows and $\overline{\gamma}$ is the map from the example above.

In D_1 all the "splittings" of pairs has been done. Fixing the first component of β, the pairs $((\alpha,1),(\beta,1))$ and $((\alpha,2),(\beta,1))$ pick out the poly-hom-sets $\mathrm{Hom}(A\ ;\ C)$ and $\mathrm{Hom}(B\ ;\ C)$ respectively; a map from each of these is:

$$\begin{aligned} \overleftarrow{\alpha}[1](\_)_{(\beta,1)} &: \mathrm{Hom}(A\ ;\ C) \to \mathrm{Hom}(A \times B\ ;\ C) \\ \overleftarrow{\alpha}[2](\_)_{(\beta,1)} &: \mathrm{Hom}(B\ ;\ C) \to \mathrm{Hom}(A \times B\ ;\ C) \end{aligned}$$

where the subscripted $(\beta,1)$ is used to indicate that the first component of β is fixed. The cotupling map of both of these maps gives a map

$$\left\{ \overleftarrow{\alpha}[1](\_)_{(\beta,1)},\ \overleftarrow{\alpha}[2](\_)_{(\beta,1)} \right\} : D_1 \to D_2$$

Similarly, fixing the second component of β gives the maps

$$\begin{aligned} \overleftarrow{\alpha}[1](\_)_{(\beta,2)} &: \mathrm{Hom}(A\ ;\ D) \to \mathrm{Hom}(A \times B\ ;\ D) \\ \overleftarrow{\alpha}[2](\_)_{(\beta,2)} &: \mathrm{Hom}(B\ ;\ D) \to \mathrm{Hom}(A \times B\ ;\ D) \end{aligned}$$

and thus the cotupling map,

$$\left\{ \overleftarrow{\alpha}[1](\_)_{(\beta,2)},\ \overleftarrow{\alpha}[2](\_)_{(\beta,2)} \right\} : D_1 \to D_2$$

Together these (cotupling) maps give the following map

$$\sum_{j \in \{1,2\}} \left\{ \overleftarrow{\alpha}[1](\_)_{(\beta,j)},\ \overleftarrow{\alpha}[2](\_)_{(\beta,j)} \right\} : D_1 \to D_2$$

In the same way a second map may be constructed where α is fixed instead:

$$\sum_{i \in \{1,2\}} \left\{ \overrightarrow{\beta}[1](\_)_{(\alpha,i)},\ \overrightarrow{\beta}[2](\_)_{(\alpha,i)} \right\} : D_1 \to D_2$$

Definition 4.2.8 An inclusion $\mathcal{I} : \mathbf{A} \longrightarrow \mathbf{B}$ which is both a semi-soft and soft

extension is said to be **soft**.

Lemma 4.2.9 *The inclusion morphism of polycategories* $\mathbf{A} \xrightarrow{\mathcal{I}} \Sigma\Pi_{\mathbf{A}}$ *is soft.*

Proof. That the inclusion $\mathcal{I}$ is semi-soft follows by Proposition 4.1.2 and which (non-cut) rules we may apply.

As mentioned in Remark 4.2.6, to show that it is a soft extension we will assume that the diagram (★) has a coequalizer (q, Q), and show that Q is isomorphic to $\mathrm{Hom}(\Gamma, \Delta)$. As we are in **Set**, a bijective correspondence between the two objects suffices to establish the isomorphism.

As Q is the coequalizer, there exists a unique map $h : Q \to \mathrm{Hom}(\Gamma; \Delta)$ such that $q; h = \{\overline{\gamma}[k](\_)\}_{(\gamma,k)}$. But $\{\overline{\gamma}[k](\_)\}_{(\gamma,k)}$ is a surjection and hence, so is h. It is left to establish that h is an injection.

Suppose that $\overline{\alpha}[i](f)$ and $\overline{\beta}[j](g)$ are equivalent morphisms in $\mathrm{Hom}(\Gamma; \Delta)$. From the decision procedure for $\Sigma\Pi$-morphisms we know that the equality of morphisms is determined by the equality of their arguments so that $\overline{\alpha}[i](f)$ and $\overline{\beta}[j](g)$ are equivalent to morphisms of the form $\overline{\alpha}[i](\overline{\beta}[j](f'))$ and $\overline{\beta}[j](\overline{\alpha}[i](g'))$ respectively. This implies that f' and g' are equivalent in $\sum_{((\alpha,i),(\beta,j))} [\mathrm{Hom}(\Gamma; \Delta)]_{((\alpha,i),(\beta,j))}$, and therefore, must be coequalized in Q establishing that h is injective.

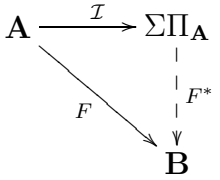
Thus, the map h is a bijection and $Q \cong \mathrm{Hom}(\Gamma; \Delta)$, establishing that

$$\big(\{\overline{\gamma}[k](\_)\}_{(\gamma,k)},\ \mathrm{Hom}(\Gamma; \Delta)\big)$$

is the coequalizer. ∎

Let $\mathbf{A} \xrightarrow{\mathcal{I}} \Sigma\Pi_{\mathbf{A}}$ be an inclusion morphism of polycategories and $\mathbf{A} \xrightarrow{F} \mathbf{B}$ be a full inclusion morphism of polycategories (injective on objects and an isomorphism on each poly-hom-set), where the objects of $\mathbf{B}$ are generated from the objects of $\mathbf{A}$ under finite sums and products. That $\Sigma\Pi_{\mathbf{A}}$ is free guarantees that there is a unique comparison morphism of polycategories $\Sigma\Pi_{\mathbf{A}} \xrightarrow{F^*} \mathbf{B}$ such that the following

diagram commutes



Given this data, the "Whitman theorem" which characterizes the free polycategory with sums and products is as follows.

Theorem 4.2.10 (Whitman theorem) *For F and F^* as above:*

(i) *If F is a semi-soft extension then F^* is full.*

(ii) *If F is soft then F^* is an equivalence of polycategories.*

Proof.

(i) That F^* is full follows from the following induction. Suppose $f : F^*(\Gamma) \to F^*(\Delta) \in \mathbf{B}$ (where $F^*(X_1, \ldots, X_n) = F^*(X_1), \ldots, F^*(X_n)$). If f is atomic this implies that $f \in \mathbf{A}$, and thus also in $\Sigma\Pi_{\mathbf{A}}$. So suppose that f is not atomic. The point will be to show that f may be "decomposed" into a "$\Sigma\Pi$-word" of simpler (smaller) functions which, by induction, are in the image of F^*. Then, as F^* preserves sums and products the same $\Sigma\Pi$-word may be used to place f in the image of F^*.

To see that f may be decomposed consider the form of $F^*(\Gamma)$ and $F^*(\Delta)$. If $F^*(\Gamma)$ contains a sum or $F^*(\Delta)$ contains a product, then by the equivalences $(*)$ and $(\star)$, f may be decomposed on the sum or the product respectively; if neither of these is the case then, as F is a semi-soft extension, f must have been the result of an injection or a projection map. In either case, we are able to decompose f into smaller functions.

(ii) To show that F^* is an equivalence of polycategories it suffices to show that F^* is full, faithful, and essentially surjective, where by essentially surjective we mean that each object in $\mathbf{B}$ is isomorphic to $F^*(X)$ for some X in $\Sigma\Pi_{\mathbf{A}}$. If F is soft, then by definition it is a semi-soft extension, and thus, F^* is full.

To show that F^* is faithful, consider a parallel pair of arrows $F^*(f), F^*(g) : F^*(\Gamma) \rightrightarrows F^*(\Delta)$ such that $F^*(f) = F^*(g)$; as above we may decompose $F^*(f)$ and $F^*(g)$ into $\Sigma\Pi$-words. Again, the decomposition is obvious if either $F^*(\Gamma)$ contains a sum or $F^*(\Delta)$ contains a product, so suppose not, and moreover, suppose that we are able to decompose $F^*(f)$ and $F^*(g)$ into different $\Sigma\Pi$-words (via injection or projection maps). Then, by softness, these words may again be decomposed so that they are equivalent. Thus, in both cases, $F^*(f)$ and $F^*(g)$ may be decomposed into different substitution instances of the same $\Sigma\Pi$-word. As F^* is full, these words involve simpler functions of the form $F^*(h)$. By induction the corresponding subterms h are equal, and thus $f = g$.

To see that F^* is essentially surjective, notice that since $\mathbf{B}$ is also generated from $\mathbf{A}$ under finite sums and products, each object of $\mathbf{B}$ must be isomorphic to an object in the image of F^*.

■

Chapter 5

Process Semantics

The term calculus for $\Sigma\Pi$ used in the previous chapters was motivated by the interpretation of $\Sigma\Pi$-derivations as processes. The purpose of this chapter is to explain precisely how these terms correspond to processes. To this end a semantics for processes is introduced. This semantics could alternatively have been described as a game theoretic semantics, and indeed many of the ideas and terminology are derived from that view (see, e.g., [AM98b, AJM00, HO00]).

5.1 Behaviours

5.1.1 Legal transitions

A protocol may be in an atomic state, an output state, or an input state. These states may be assigned one of three different **roles** which will be used to determine the legality of transitions:

- **Source roles**: these are denoted by superscripting a $\mathbf{0}$.
- **Sink roles**: these are denoted by superscripting a $\mathbf{1}$.
- **Flow roles**: these are denoted by superscripting a $+$.

Given a protocol on a domain channel we may calculate its role inductively as follows:

- any atomic formula A has a flow role: A^{+};
- the empty coproduct has a source role: $\{\,\}^{\mathbf{0}}$;
- the empty product has a sink role: $(\,)^{\mathbf{1}}$;
- the role x of a coproduct, $\{a_i : A_i^{x_i}\}_i^x$ is determined by the roles x_i of its subformula as follows:

$$x = \begin{cases} \mathbf{0} & \text{if and only if } (\forall i)\ x_i = \mathbf{0} \\ \mathbf{1} & \text{if and only if } (\exists! k)\ x_k = \mathbf{1} \text{ and } (\forall i \neq k)\ x_i = \mathbf{0} \\ + & \text{otherwise} \end{cases}$$

- the role x of a product, $(a_i : A_i^{x_i})_i^x$ is determined by the roles x_i of its subformula as follows:

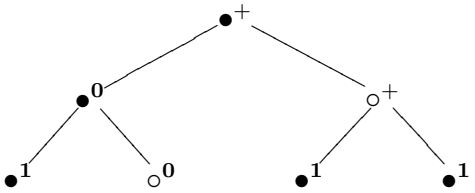
$$x = \begin{cases} \mathbf{0} & \text{if and only if } (\exists! k)\ x_k = \mathbf{0} \text{ and } (\forall i \neq k)\ x_i = \mathbf{1} \\ \mathbf{1} & \text{if and only if } (\forall i)\ x_i = \mathbf{1} \\ + & \text{otherwise} \end{cases}$$

The role of a protocol on a codomain channel is calculated dually, i.e., swap the **0** and the **1** while leaving + alone. It is easy to see that any protocol which contains an atomic formula will have a flow role.

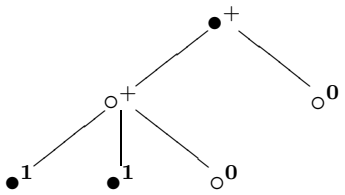
It should be mentioned that the purpose of the role of a protocol is to identify the initial (source), final (sink), and other (flow) protocols, in the usual categorical sense.

Example 5.1.1 (Roles) Some example protocols (on domain channels) with their roles calculated.

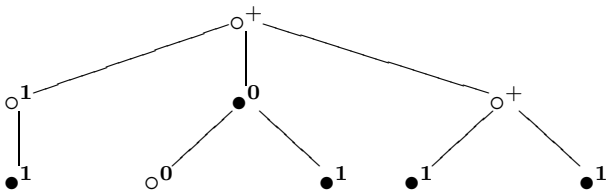
1.



2.



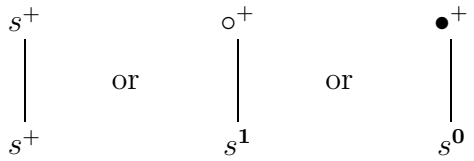
3.



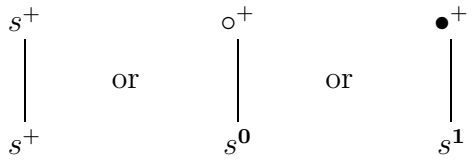
A protocol transition is **legal** (is a **legal transition**) in case it starts at a state in a flow role and is either

- an output transition which ends at a state which does not have a sink role, or
- an input transition which ends at a state which does not have a source role.

Graphically, a transition in the domain is legal in case it is either

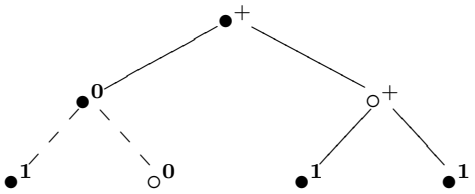


where s represents a product or a coproduct state. In the codomain a transition is legal in case it is either

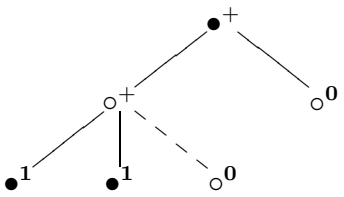


Example 5.1.2 (Legal transitions) Using the protocols from the above example (Example 5.1.1), we indicate the legal transitions as solid lines and the non-legal transitions as dashed lines.

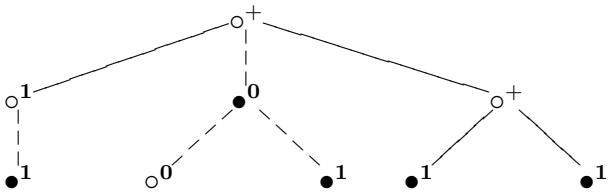
1.



2.



3.



Lemma 5.1.3 *For any protocol X in an input flow state there is a sequence of legal input events which drives the protocol into either an atomic, output or sink state.*

Proof. We prove this for a protocol X on a domain channel; duality covers the case where X is on a codomain channel.

For X on a domain channel, to be in an input flow state it must be of the form $X = \{a_i : X_i\}_{i \in I}$ where $I \neq \emptyset$. As X has a flow role, this means that not every X_i has a source role, i.e., there is at least one $k \in I$ such that $X_k^{\mathbf{1}}$ or X_k^+. Thus, input a_k to X. In the former case we are done as $X_k^{\mathbf{1}}$ has a sink role. In the latter case X_k^+ has a flow role which may be either an atomic, output or input state; if it is an atomic or output state then the desired conclusion has be reached, and otherwise, it is an input state which we may inductively assume has the desired property. ∎

5.1.2 Legal behaviours

Suppose a channel α is assigned a protocol X. A **legal channel behaviour** (or simply **channel behaviour**) for α, denoted $\mathfrak{p}_\alpha$, is a finite sequence of legal events on α satisfying its assigned protocol X (see Section 3.2). For example, the behaviour $\langle a, \overline{b} \rangle$ for a channel α represents that on α the input event a was received, after which, the event $\overline{b}$ was output.

An (input or output) event e may be appended to a channel behaviour $\mathfrak{p}_\alpha$, denoted $\mathfrak{p}_\alpha * e$. Similarly, an (input or output) event e' may be prepended to $\mathfrak{p}_\alpha$, which is denoted $e' * \mathfrak{p}_\alpha$. For example, $\langle a, \overline{b} \rangle * c = \langle a, \overline{b}, c \rangle$ and $c * \langle a, \overline{b} \rangle = \langle c, a, \overline{b} \rangle$. In this manner (finite) sequences of events may be appended or prepended (or both) to a channel behaviour.

If $\mathfrak{p}_\alpha$ and $\mathfrak{q}_\alpha$ are two channel behaviours on α, we say that $\mathfrak{p}_\alpha$ is a **prefix** of $\mathfrak{q}_\alpha$, denoted $\mathfrak{p}_\alpha \sqsubseteq \mathfrak{q}_\alpha$, if there exists a sequence of events $\mathfrak{p}'_\alpha$ such that $\mathfrak{p}_\alpha * \mathfrak{p}'_\alpha = \mathfrak{q}_\alpha$. It is a **proper prefix**, denoted $\mathfrak{p}_\alpha \sqsubset \mathfrak{q}_\alpha$, if $\mathfrak{p}_\alpha \sqsubseteq \mathfrak{q}_\alpha$ and $\mathfrak{p}_\alpha \neq \mathfrak{q}_\alpha$. The notation $\mathfrak{p} \sqsubseteq_i \mathfrak{q}$ and $\mathfrak{p} \sqsubseteq_o \mathfrak{q}$ is used to indicate that $\mathfrak{p}$ is a prefix of $\mathfrak{q}$ separated only by input or output events respectively.

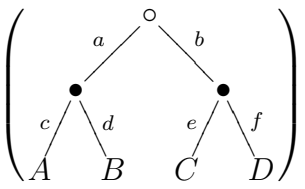
A **legal behaviour** (or simply **behaviour**) $\mathfrak{p} = (\mathfrak{p}_{\alpha_1}, \ldots, \mathfrak{p}_{\alpha_n})$ is a tuple of channel behaviours, one for each channel along which the process interacts. We will typically make use of **tables** to denote behaviours. For example, if $\langle a, \overline{b} \rangle_\alpha$, $\langle c, \overline{d}, \overline{e} \rangle_\beta$, $\langle f \rangle_\gamma$, and $\langle g, h \rangle_\delta$ are channel behaviours for α, β, γ, and δ respectively, where the domain consists of α and β and the codomain consists of γ and δ, the behaviour consisting of these channel behaviours may be denoted

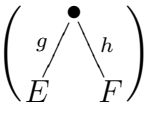
	$\overline{e}$		
$\overline{b}$	$\overline{d}$		h
a	c	f	g
α	β	γ	δ

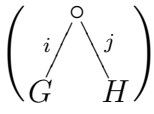
The **dual behaviour** $\mathfrak{p}^*$ of $\mathfrak{p}$ is the behaviour $\mathfrak{p}$ in which input events are considered as output events and output events are considered as input events. That is, any input event in $\mathfrak{p}$ is an output event in $\mathfrak{p}^*$, and similarly any output event in $\mathfrak{p}$ is an input event in $\mathfrak{p}^*$. For example, the dual of the behaviour above is:

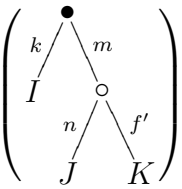
	e		
b	d		$\overline{h}$
$\overline{a}$	$\overline{c}$	$\overline{f}$	$\overline{g}$
α	β	γ	δ

The last states of a behaviour is called its **frontier**. For example, given the protocols

α : 

β : 

γ : 

δ : 

and the behaviour

$\overline{c}$			
a		$\overline{j}$	k
α	β	γ	δ

The frontier consists of $\alpha : A,\ \beta : (g : E, h : F),\ \gamma : H$, and $\delta : I$. Notice that the frontier is actually representing a sequent: it is what is yet to be proved. In the above example, the sequent the frontier represents is:

$$\alpha : A,\ \beta : (g : E, h : F) \vdash \gamma : H,\ \delta : I$$

Given any behaviour $\mathfrak{p}$ in which α is a channel of this behaviour, we may form the **restriction** of $\mathfrak{p}$ to α, denoted $\mathfrak{p}_\alpha$. For example,

$$\mathfrak{p} = \begin{array}{|c|c||c|c|}\hline \overline{b} & \overline{e} & & \\ a & \overline{d} & f & h \\ & c & & g \\ \hline \alpha & \beta & \gamma & \delta \\ \hline \end{array} \qquad \mathfrak{p}_\alpha = \begin{array}{|c|}\hline \overline{b} \\ a \\ \hline \alpha \\ \hline \end{array}$$

The prefix and proper prefix of behaviours are defined in the obvious way, i.e., $\mathfrak{p} \sqsubseteq \mathfrak{q}$ if for each channel α of $\mathfrak{p}$ we have $\mathfrak{p}_\alpha \sqsubseteq \mathfrak{q}_\alpha$, and $\mathfrak{p} \sqsubset \mathfrak{q}$ if $\mathfrak{p} \sqsubseteq \mathfrak{q}$ and $\mathfrak{p} \neq \mathfrak{q}$. Appending and prepending events is similar to the case for channel behaviours except that the channel to append/prepend on must be specified, e.g., $\mathfrak{p} * \beta[a]$ is interpreted as $(\mathfrak{p}_{\alpha_1}, \ldots, \mathfrak{p}_\beta * \beta[a], \ldots, \mathfrak{p}_{\alpha_n})$ and $\beta[b] * \mathfrak{p}$ is interpreted as $(\mathfrak{p}_{\alpha_1}, \ldots, \beta[a] * \mathfrak{p}_\beta, \ldots, \mathfrak{p}_{\alpha_n})$.

A behaviour in which all the states at the frontier have either flow (+) or sink (**1**) roles is called an **antecedent behaviour**. An antecedent behaviour is called **saturated** if for any state at the frontier, it is either atomic, has a sink role, or has an output flow role. If all the states at the frontier are atomic, we may refer to this behaviour as being **atomic saturated**.

Two behaviours $\mathfrak{p}$ and $\mathfrak{q}$ are **compatible**, denoted $\mathfrak{p} \smile \mathfrak{q}$, if for each channel α either $\mathfrak{p}_\alpha \sqsubseteq \mathfrak{q}_\alpha$ or $\mathfrak{q}_\alpha \sqsubseteq \mathfrak{p}_\alpha$. Given two compatible behaviours $\mathfrak{p}$ and $\mathfrak{q}$, their **join**, denoted $\mathfrak{p} \vee \mathfrak{q}$, is defined as:

$$(\mathfrak{p} \vee \mathfrak{q})_\alpha = \begin{cases} \mathfrak{q}_\alpha & \text{if } \mathfrak{p}_\alpha \sqsubseteq \mathfrak{q}_\alpha \\ \mathfrak{p}_\alpha & \text{if } \mathfrak{q}_\alpha \sqsubseteq \mathfrak{p}_\alpha \end{cases} \quad \text{for all } \alpha$$

Example 5.1.4 (Compatable behaviours) The following behaviours

$$\mathfrak{p} = \begin{array}{|c|c||c|c|}\hline \overline{b} & & \overline{a} & \\ a & \overline{c} & d & \\ \hline \alpha & \beta & \gamma & \delta \\ \hline \end{array} \qquad \text{and} \qquad \mathfrak{q} = \begin{array}{|c|c||c|c|}\hline & e & & \\ \overline{b} & d & & \overline{c} \\ a & \overline{c} & d & a \\ \hline \alpha & \beta & \gamma & \delta \\ \hline \end{array}$$

are compatible with their join given by:

$$\mathfrak{p} \vee \mathfrak{q} = \begin{array}{|c|c||c|c|} \hline & e & & \\ \overline{b} & d & \overline{a} & \overline{c} \\ a & \overline{c} & d & a \\ \hline \alpha & \beta & \gamma & \delta \\ \hline \end{array}$$

Definition 5.1.5 A **behavioural entailment** (or simply **entailment**) is either of the form

$$\mathfrak{p} \vdash \overline{\alpha}[a]$$

where $\mathfrak{p}$ is an antecedent behaviour and a is a legal output event on channel α so that $\mathfrak{p} * \overline{\alpha}[a]$ is itself a behaviour, or

$$\mathfrak{p} \vdash f$$

where $f : \Gamma \to \Delta$ is an atomic morphism on the frontier of $\mathfrak{p}$. We call $\mathfrak{p}$ the **antecedent** of the entailment and the $\overline{\alpha}[a]$ or f the **conclusion** of the entailment. If the conclusion of an entailment is an atomic morphism it is called an **atomic entailment**. The notation $\mathfrak{p} \vdash \times$ will be used to denote that conclusion may be either an output event or an atomic morphism.

Notice that no events may be added (or removed) to the antecedent of an atomic entailment, effectively "ending" the entailment. A **hanging entailment** is an entailment in which the conclusion is an output event which gives a transition to a state with a source ($\mathbf{0}$) role. Observe that after a hanging entailment $\mathfrak{p} \vdash \overline{\alpha}[a]$, the ensuing behaviour $\mathfrak{p} * \overline{\alpha}[a]$ will no longer be an antecedent behaviour.

We will usually talk about a set of entailments $\mathcal{Q} : \Gamma \to \Delta$, where every entailment in $\mathcal{Q}$ has as domain channels Γ and codomain channels Δ. An antecedent behaviour which occurs in $\mathcal{Q}$ (i.e., an antecedent of some entailment in $\mathcal{Q}$) will be called a $\mathcal{Q}$-**antecedent**. The notation $\alpha[a]*\mathcal{Q}$ will be used to denote the set of entailments $\mathcal{Q}$ with the event $\alpha[a]$ prepended to each of them, e.g., if $\mathcal{Q} = \{\mathfrak{p}_i \vdash \times_i \mid i \in I\}$ then $\alpha[a] * \mathcal{Q} = \{\alpha[a] * \mathfrak{p}_i \vdash \times_i \mid i \in I\}$.

An output event $\overline{\alpha}[a] \in \mathfrak{p}$ is $\mathcal{Q}$-**justified** if there is an entailment $\mathfrak{u} \vdash \overline{\alpha}[a] \in \mathcal{Q}$ such that $\mathfrak{u} * \overline{\alpha}[a] \sqsubseteq \mathfrak{p}$. A $\mathcal{Q}$-**justified behaviour** is a behaviour in which all the

output events are $\mathcal{Q}$-justified. A $\mathcal{Q}$-justified behaviour which is also an antecedent behaviour (not necessarily a $\mathcal{Q}$-antecedent) will be called a $\mathcal{Q}$-**preantecedent**. Note that this notion of justification is not related to the Hyland-Ong [HO00, and other references] notion of justification, and is actually a reachability condition.

Lemma 5.1.6 *Let $\mathcal{Q}$ be a set of entailments. Every $\mathcal{Q}$-preantecedent can, via a sequence (possible empty) of legal inputs, evolve to a saturated $\mathcal{Q}$-preantecedent.*

Proof. Let $\mathfrak{p}$ be a $\mathcal{Q}$-justified antecedent behaviour. Then all the states at the frontier of $\mathfrak{p}$ must have either flow or sink roles. Consider the states with flow roles. By Lemma 5.1.3 we are able, via a sequence of input events, to drive each of these into either an atomic, output, or sink state. Thus, we are able to produce a behaviour $\mathfrak{p}'$ such that every state in its frontier is either an atomic state, an output state, or a sink state, i.e., a saturated behaviour. Moreover, since input events do not affect justification we have that $\mathfrak{p}'$ is $\mathcal{Q}$-justified. ∎

5.1.3 Extensional processes

Definition 5.1.7 A set of entailments $\mathcal{Q} : \Gamma \to \Delta$ is an **extensional process** if it satisfies the following conditions:

EP-1 All $\mathcal{Q}$-antecedents are $\mathcal{Q}$-justified.

EP-2 If $\mathfrak{p} \vdash \overline{\alpha}[a],\ \mathfrak{q} \vdash \overline{\alpha}[b] \in \mathcal{Q}$ such that $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} \smile \mathfrak{q}$, then $a = b$.

EP-3 For any $\mathcal{Q}$-preantecedent $\mathfrak{p}$ and saturated $\mathcal{Q}$-preantecedent $\mathfrak{q}$ where $\mathfrak{p} \sqsubseteq_i \mathfrak{q}$, there is a $\mathcal{Q}$-antecedent $\mathfrak{p}'$ such that $\mathfrak{p} \sqsubseteq_i \mathfrak{p}' \sqsubseteq_i \mathfrak{q}$.

EP-4 If α and β are distinct channels and $\mathfrak{p} \vdash \overline{\alpha}[a],\ \mathfrak{q} \vdash \overline{\beta}[b] \in \mathcal{Q}$ such that $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} \sqsubseteq \mathfrak{q}$, then $\mathfrak{q} * \overline{\alpha}[a] \vdash \overline{\beta}[b] \in \mathcal{Q}$.

EP-5 If α and β are distinct channels such that $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{Q}$ and $\beta[b]$ is a legal input event given $\mathfrak{p}$, then $\mathfrak{p} * \beta[b] \vdash \overline{\alpha}[a] \in \mathcal{Q}$.

EP-6 If α and β are distinct channels such that $\mathfrak{p} * \overline{\beta}[b] \vdash \overline{\alpha}[a] \in \mathcal{Q}$, then $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{Q}$.

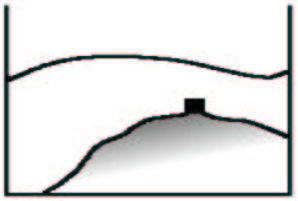
EP-7 If α and β are distinct channels and $\mathfrak{p}$ is a $\mathcal{Q}$-preantecedent, such that $\mathfrak{p} * \beta[b] \vdash \overline{\alpha}[a] \in \mathcal{Q}$ for every possible legal input b on β, then $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{Q}$.

Remark 5.1.8 The definition for an extensional process is rather dense, and so it may help to have some intuition behind the rules, which we give here. First of all, behaviours may be thought of as partial processes. The behaviour is explicit representation of the history of the process. Entailments may be thought of as a partial processes which may perform an output event (the conclusion of the entailment).

In this system, justification is a reachability condition for processes, i.e., to output certain events certain other events must have already been performed. So, EP-1 asserts that any output event in the antecedent of each entailment in an extensional process is reachable given what events have been previously performed. EP-2 ensures that processes are deterministic. That is, two compatible behaviours (two processes with compatible histories) which are able to output at a given state on a particular channel, must output the same event. EP-3 states that if a process is able to perform an output event, then it must eventually perform one. In game semantic lingo, this says that the player may not give up. The rule EP-4 states that if a process can perform an output event at a certain stage, then it can certainly perform that output event at any later stage (say after first receiving some other input events and performing some other output events). The rule EP-5 allows the environment to input events without affecting the the output event the process is about to produce. EP-6 states that output events do not affect the justification (reachability) of other output events. Finally, EP-7 states that if for every possible input events on a given channel, a process is able to output the same event, then this process must then be able to perform this output event immediately without first receiving any of these inputs.

Remark 5.1.9 We have found it useful to think of behaviours and entailments using "landscape" diagrams. For example, the diagram on the left below is a representation of a behaviour and the diagram on the right below is a representation of an entailment, the black square representing the conclusion of the entailment.

In this way we may think of a justified output event in a behaviour as:



Using these diagrams, the premise and conclusion of the rule EP-4 are respectively the left and right diagrams below.

Lemma 5.1.10 *Let* $\mathcal{Q} : \Gamma \to \Delta$ *be a set of entailments satisfying EP-1 (i.e., a set of* $\mathcal{Q}$*-justified entailments). If* $\mathfrak{p}$ *and* $\mathfrak{q}$ *are* $\mathcal{Q}$*-preantecedents such that* $\mathfrak{p} \smile \mathfrak{q}$*, then* $\mathfrak{p} \vee \mathfrak{q}$ *is a* $\mathcal{Q}$*-preantecedent.*

Proof. This is obvious as any output event in $\mathfrak{p}$ is justified in $\mathfrak{p} \sqsubseteq \mathfrak{p} \vee \mathfrak{q}$, and similarly for output events in $\mathfrak{q}$. ∎

Lemma 5.1.11 *Let $\mathcal{Q} : \Gamma \to \Delta$ be a set of entailments satisfying EP-1. Suppose that $\mathfrak{p}$ and $\mathfrak{q}$ are $\mathcal{Q}$-preantecedents such that $\mathfrak{p} \sqsubset \mathfrak{q}$ and for all channels α either $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$, or there is an output event $\overline{\alpha}[a] \in \mathfrak{q}$ such that $\mathfrak{p}_\alpha * \overline{\alpha}[a] \sqsubseteq \mathfrak{q}_\alpha$. Then there is an output event $\overline{\beta}[b] \in \mathfrak{q}$ such that its justification $\mathfrak{u}$ is a prefix of $\mathfrak{p}$.*

Proof. Let α_i for $i \in \{1, \ldots, n\}$ where $n \geq 1$, be all the channels in which $\mathfrak{p}_{\alpha_i} \sqsubset \mathfrak{q}_{\alpha_i}$ and consider output events $\overline{\alpha}_i[a_i] \in \mathfrak{q}$ just past the frontier of $\mathfrak{p}$, i.e., $\mathfrak{p} * \overline{\alpha}_i[a_i] \sqsubseteq \mathfrak{q}$.

Consider $\overline{\alpha}_k[a_k] \in \mathfrak{q}$ where $k \in \{1, \ldots, n\}$. Its justification $\mathfrak{u}_k \in \mathcal{Q}$ must have its channel behaviour for α_k contained in $\mathfrak{p}$. If $\mathfrak{u}_k \not\sqsubseteq \mathfrak{p}$ then there exists a $k' \in \{1, \ldots, n\} \backslash \{k\}$ such that $\overline{\alpha}_{k'}[a_{k'}] \in \mathfrak{u}_k$ but $\overline{\alpha}_{k'}[a_{k'}] \notin \mathfrak{p}$. The justification of $\alpha_{k'}[a_{k'}]$ in $\mathfrak{u}_k$, say $\mathfrak{u}_{k'}$, must be a prefix of $\mathfrak{u}_k$ and thus, must have channel behaviours for α_k and $\alpha_{k'}$ contained in $\mathfrak{p}$. Continuing this process means that eventually we will find an output $\overline{\beta}[b]$ with justification $\mathfrak{u} \in \mathcal{Q}$ such that $\mathfrak{u} \sqsubseteq \mathfrak{p}$. ∎

Lemma 5.1.12 *Let $\mathcal{Q}$ be a set of entailments satisfying EP-1, EP-4, and EP-5. If $\mathfrak{p} \vdash \times \in \mathcal{Q}$, and $\mathfrak{q}$ is a $\mathcal{Q}$-preantecedent such that $\mathfrak{p} \sqsubseteq \mathfrak{q}$ and $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$, then $\mathfrak{q} \vdash \times \in \mathcal{Q}$.*

Proof. We proceed by induction on the number of events between $\mathfrak{p}$ and $\mathfrak{q}$. If there are no events between $\mathfrak{p}$ and $\mathfrak{q}$ then $\mathfrak{p} = \mathfrak{q}$ and therefore, $\mathfrak{p} \vdash \times \in \mathcal{Q}$.

So suppose that there are N events between $\mathfrak{p}$ and $\mathfrak{q}$ and consider the channels β_i, for $i \in I$, in which $\mathfrak{p}_{\beta_i} \sqsubset \mathfrak{q}_{\beta_i}$. There are two cases to consider: on some β_i we can add input events and on no β_i can we add an input event.

In the first case, use EP-5 to add an input event to $\mathfrak{p} \vdash x$. This new entailment and $\mathfrak{q}$ are now separated by $N - 1$ events so that by the inductive hypothesis $\mathfrak{q} \vdash \times \in \mathcal{Q}$.

In the second case we may conclude by Lemma 5.1.11 that there is an output event $\overline{\beta_k}[b] \in \mathfrak{q}$ (but not in $\mathfrak{p}$) for some $k \in I$ such that its justification is a prefix of $\mathfrak{p}$. By EP-4 we may add this event to $\mathfrak{p} \vdash \times$, i.e., $\mathfrak{p} * \overline{\beta}[b] \vdash \times \in \mathcal{Q}$. Applying the inductive hypothesis to $\mathfrak{p} * \overline{\beta}[b] \vdash \times$ and $\mathfrak{q}$ now yields $\mathfrak{q} \vdash \times \in \mathcal{Q}$. ∎

The following useful fact is a consequence of the previous lemma.

Corollary 5.1.13 *Let $\mathcal{Q}$ be a set of entailments satisfying EP-1, EP-4, and EP-5. If $\mathfrak{p} \vdash \times \in \mathcal{Q}$ and $\mathfrak{q}$ is a $\mathcal{Q}$-preantecedent such that $\mathfrak{p} \smile \mathfrak{q}$ and $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$, then $\mathfrak{p} \vee \mathfrak{q} \vdash \times \in \mathcal{Q}$.*

Proof. It is clear that $\mathfrak{p} \vee \mathfrak{q}$ is a $\mathcal{Q}$-preantecedent with $\mathfrak{p} \sqsubseteq (\mathfrak{p} \vee \mathfrak{q})$ and $\mathfrak{p}_\alpha = (\mathfrak{p} \vee \mathfrak{q})_\alpha$ so that by Lemma 5.1.12, $\mathfrak{p} \vee \mathfrak{q} \vdash \times \in \mathcal{Q}$. ∎

The next proposition shows that an extensional process is maximal.

Proposition 5.1.14 *Let $\mathcal{Q}, \mathcal{Q}' : \Gamma \to \Delta$ be extensional processes. If $\mathcal{Q} \subseteq \mathcal{Q}'$ then $\mathcal{Q} = \mathcal{Q}'$.*

Proof. To show equality we must show that $\mathcal{Q}' \subseteq \mathcal{Q}$. To this end, suppose that $\mathfrak{p}' \vdash \times \in \mathcal{Q}'$ and let $\mathfrak{p}$ be a $\mathcal{Q}$-preantecedent such that $\mathfrak{p} \sqsubseteq \mathfrak{p}'$ ($\mathfrak{p}$ may be the empty preantecedent). We proceed by induction on the number of events between $\mathfrak{p}$ and $\mathfrak{q}$. If there are no events between $\mathfrak{p}$ and $\mathfrak{q}$ then $\mathfrak{p} = \mathfrak{q}$ and therefore, $\mathfrak{q} \vdash \times \in \mathcal{Q}$.

So, suppose that there are N events between $\mathfrak{p}$ and $\mathfrak{q}$. There are two cases: we can add an input event on some channel α or on no channel can we add an input event.

In the first case use EP-5 to add an input event to $\mathfrak{p}$ which corresponds to an event in $\mathfrak{q}$. This preantecedent and $\mathfrak{q}$ are now separated by $N - 1$ events so that by the inductive hypothesis $\mathfrak{q} \vdash \times \in \mathcal{Q}$.

In the second case, consider $\mathfrak{p}$ to be a $\mathcal{Q}'$-preantecedent. By Lemma 5.1.11, there is an output event $\overline{\alpha}[a] \in \mathfrak{q}$ such that its justification is a prefix of $\mathfrak{p}$. This implies that $\mathfrak{p} * \overline{\alpha}[a]$ is a $\mathcal{Q}'$-preantecedent, and therefore, a $\mathcal{Q}$-preantecedent. $\mathfrak{p} * \overline{\alpha}[a]$ and $\mathfrak{q}$ are now separated by $N - 1$ events so that by our inductive hypothesis $\mathfrak{q} \vdash \times \in \mathcal{Q}$.

We have now shown that $\mathcal{Q}' \subseteq \mathcal{Q}$ and consequently $\mathcal{Q} = \mathcal{Q}'$. ∎

5.2 Proto-processes

A set of entailments $\mathcal{P}$ satisfying the conditions EP-1 through EP-3 is said to be a **proto-process**. Explicitly, a set of entailments $\mathcal{P}$ is a proto-process if it satisfies the following conditions:

EP-1 All $\mathcal{P}$-antecedents are $\mathcal{P}$-justified.

EP-2 If $\mathfrak{p} \vdash \overline{\alpha}[a],\ \mathfrak{q} \vdash \overline{\alpha}[b] \in \mathcal{P}$ such that $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} \smile \mathfrak{q}$, then $a = b$.

EP-3 For any $\mathcal{P}$-preantecedent $\mathfrak{p}$ and saturated $\mathcal{P}$-preantecedent $\mathfrak{q}$ where $\mathfrak{p} \sqsubseteq_i \mathfrak{q}$, there is a $\mathcal{P}$-antecedent $\mathfrak{p}'$ such that $\mathfrak{p} \sqsubseteq_i \mathfrak{p}' \sqsubseteq_i \mathfrak{q}$.

In this section we show how to translate a (syntactic) process f, into a set of entailments $\mathcal{F}$, such that $\mathcal{F}$ forms a proto-process. The main result of this section is that a proto-process $\mathcal{P}$ may be enlarged in such a way that it additionally satisfies the rules EP-4 $\sim$ EP-7, i.e., a proto-process may be enlarged to an extensional process. This gives a way to translate syntactic processes into extensional processes.

5.2.1 From (syntactic) processes to proto-processes

Given a (syntactic) process $f : \Gamma \to \Delta$, the translation of f into a set of entailments, denoted $\mathcal{F}$, is defined inductively as follows:

- If Γ or Δ (or both) contain a protocol with a source role, then set $\mathcal{F} = \emptyset$.
- If there are no protocols in Γ and Δ which contain a source role and
  - f is atomic, then set $\mathcal{F} = \{\vdash f\}$.
  - $f = \alpha\{a_i \mapsto f_i\}_{i \in I}$ or $f = \alpha(a_i \mapsto f_i)_{i \in I}$, then set $\mathcal{F} = \bigcup_{i \in I} \{\alpha[a_i] * \mathcal{F}_i\}$, where $\mathcal{F}_i$ is the translation of f_i.
  - $f = \overleftarrow{\alpha}[a](f')$ or $f = \overrightarrow{\alpha}[a](f')$, then set $\mathcal{F} = \{\vdash \overline{\alpha}[a],\ \overline{\alpha}[a] * \mathcal{F}'\}$, where $\mathcal{F}'$ is the translation of f'.

Example 5.2.1 (Translating (syntactic) processes)

1. Given the following process (no source roles; assume that the atomic morphisms are well-typed)

$$\alpha \left\{ \begin{array}{l} a \mapsto \overrightarrow{\beta}[a](f) \\ b \mapsto \alpha \left\{ \begin{array}{l} d \mapsto \overrightarrow{\beta}[b](g) \\ e \mapsto \overrightarrow{\beta}[b](h) \end{array} \right\} \\ c \mapsto \overrightarrow{\beta}[c](\beta \left(\begin{array}{l} d \mapsto \overleftarrow{\alpha}[k](i) \\ e \mapsto \overleftarrow{\alpha}[l](j) \end{array} \right)) \end{array} \right\}$$

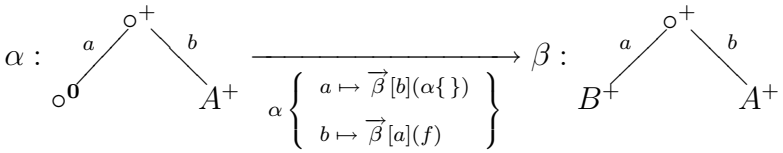
the translation produces the following set of entailments:

$$\begin{array}{|c||c|}\hline a & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[a], \qquad \begin{array}{|c||c|}\hline a & \overline{a} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash f, \qquad \begin{array}{|c||c|}\hline d & \\ b & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[b], \qquad \begin{array}{|c||c|}\hline d & \\ b & \overline{b} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash g,$$

$$\begin{array}{|c||c|}\hline e & \\ b & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[b], \qquad \begin{array}{|c||c|}\hline e & \\ b & \overline{b} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash h, \qquad \begin{array}{|c||c|}\hline & \\ c & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[c], \qquad \begin{array}{|c||c|}\hline & d \\ c & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\alpha}[k],$$

$$\begin{array}{|c||c|}\hline \overline{k} & d \\ c & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash i, \qquad \begin{array}{|c||c|}\hline & e \\ c & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\alpha}[l], \qquad \begin{array}{|c||c|}\hline \overline{l} & e \\ c & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash j$$

2. This next example illustrates how the translation handles a (sub)protocol with a source role. Consider the following protocols and process (where $f : A \to B$ is atomic):



Here a on α is an input transition to a protocol with a source role. The resulting set of entailments is:

$$\begin{array}{|c||c|}\hline b & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[a], \qquad \begin{array}{|c||c|}\hline b & \overline{a} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash f,$$

We now wish to establish that this translation procedure results in a proto-process. First however, there is an observation which is required.

Proposition 5.2.2 *The empty set $\emptyset : \Gamma \to \Delta$ is an extensional process if and only if there is a protocol with a source role in the domain or in the codomain (or both).*

Proof. We first prove the "if" direction. All the premises of the EP-rules, except for EP-3, involve having some entailments, and thus $\emptyset$ vacuously satisfies these rules. To see that $\emptyset$ satisfies EP-3 as well recall the definition of an antecedent behaviour: a behaviour is an antecedent behaviour in case each state at the frontier has either a flow or a sink role. If Γ or Δ contain a protocol with a source role, there can be no antecedent (or preantecedent) behaviours. Thus, $\emptyset$ satisfies EP-3, and is therefore an extensional process.

For the "only if" direction suppose neither Γ nor Δ contains a protocol with a source role. This implies that the empty behaviour is an preantecedent behaviour and, therefore, EP-3 would require $\emptyset$ to contain some entailment. This is a contradiction and therefore Γ or Δ must contain a protocol with a source role. ∎

Proposition 5.2.3 *If a set of entailments $\mathcal{F}$ is the result of translating a (syntactic) process $f : \Gamma \to \Delta$, then $\mathcal{F}$ is a proto-process.*

Proof. We proceed by structural induction on the process f. If f is atomic or one of Γ or Δ contains a source role, then the translation of f is clearly a proto-process.

So assume now that the translation fits one of the other cases. It must produce either

$$\text{(i)} \bigcup_{i \in I} \{\alpha[a_i] * \mathcal{F}_i\} \qquad \text{or} \qquad \text{(ii)} \ \{\vdash \overline{\alpha}[a],\ \overline{\alpha}[a] * \mathcal{F}'\}$$

where we inductively assume that the $\mathcal{F}_i$'s and $\mathcal{F}'$ are proto-processes. The goal is to show that this new set of entailments is a proto-process.

EP-1 (i) Let $\mathfrak{p}$ be a $(\bigcup_{i \in I}\{\alpha[a_i] * \mathcal{F}_i\})$-antecedent and suppose that $\overline{\beta}[b] \in \mathfrak{p}$. That $\mathfrak{p}$ is a $(\bigcup_{i \in I}\{\alpha[a_i] * \mathcal{F}_i\})$-antecedent implies that $\mathfrak{p}$ must be of the form $\alpha[a_k] * \mathfrak{p}'$, for some $k \in I$, where $\mathfrak{p}' \in \mathcal{F}_k$. So, $\overline{\beta}[b]$ must be in $\mathfrak{p}'$, and as $\mathcal{F}_k$ is a proto-process, it must be justified in $\mathcal{F}_k$, say by $\mathfrak{u}$, but this implies that $\alpha[a_k] * \mathfrak{u}$ justifies it in $\bigcup_{i \in I}\{\alpha[a_i] * \mathcal{F}_i\}$.

(ii) There are no output events in $\vdash \overline{\alpha}[a]$ so it justified, and any antecedent in $\overline{\alpha}[a] * \mathcal{F}'$ is justified using an argument similar to (i) above.

EP-2 (i) Suppose that for some $k \in I$, $\alpha[c] * \mathfrak{p} \vdash \overline{\beta}[a]$, $\alpha[c] * \mathfrak{q} \vdash \overline{\beta}[b] \in \alpha[c] * \mathcal{F}_k$ such that $(\alpha[c] * \mathfrak{p}) \smile (\alpha[c] * \mathfrak{q})$ and $(\alpha[c] * \mathfrak{p})_\beta = (\alpha[c] * \mathfrak{q})_\beta$. Then $\mathfrak{p} \vdash \overline{\beta}[a]$, $\mathfrak{q} \vdash \overline{\beta}[b] \in \mathcal{F}_k$ such that $\mathfrak{p} \smile \mathfrak{q}$ and $\mathfrak{p}_\beta = \mathfrak{q}_\beta$, and so $a = b$ by EP-2 in $\mathcal{F}_k$.

(ii) The entailment $\vdash \overline{\alpha}[c]$ and any entailment in $\overline{\alpha}[c] * \mathcal{F}'$ will never have the same channel behaviour on α, and therefore, the only case we need to consider is if $\overline{\alpha}[c] * \mathfrak{p} \vdash \overline{\beta}[a]$, $\overline{\alpha}[c] * \mathfrak{q} \vdash \overline{\beta}[b] \in \overline{\alpha}[c] * \mathcal{F}'$ such that $(\overline{\alpha}[c] * \mathfrak{p}) \smile (\overline{\alpha}[c] * \mathfrak{q})$ and $(\overline{\alpha}[c] * \mathfrak{p})_\beta = (\overline{\alpha}[c] * \mathfrak{q})_\beta$. Using an argument similar to (i) above, this implies $a = b$.

EP-3 (i) Let $\mathfrak{q}$ be a saturated $(\bigcup_{i \in I}\{\alpha[a_i] * \mathcal{F}_i\})$-preantecedent. It must be of the form $\alpha[a_k] * \mathfrak{q}'$, for some $k \in I$, where $\mathfrak{q}'$ is a saturated $\mathcal{F}_k$-preantecedent. By EP-3, $\mathfrak{q}' \vdash \times \in \mathcal{F}_k$, and thus, $\alpha[a_k] * \mathfrak{q}' \vdash \times \in \bigcup_{i \in I}\{\alpha[a_i] * \mathcal{F}_i\}$ satisfying EP-3.

(ii) Here, similarly to (i) above, any saturated $\{\vdash \overline{\alpha}[a], \overline{\alpha}[a] * \mathcal{F}'\}$-preantecedent $\mathfrak{q}$ is either of the form $\overline{\alpha}[a] * \mathfrak{q}'$ or $\mathfrak{q}'$, where $\mathfrak{q}' \in \mathcal{F}'$. Thus, an argument similar to (i) above suffices.

Thus, since each step produces a proto-process, the complete translation $\mathcal{F}$ is a proto-process. ∎

An important example of a proto-process is the **identity proto-process**. As it is known how to form the identity process (from the identity derivation; see Section 2.3), Proposition 5.2.3 tells us how to form the identity proto-process.

Example 5.2.4 (Identity proto-process) The identity process $\alpha : X \xrightarrow{1_X} \beta : X$ where X is the protocol

$$X = \{a : A, b : \{d : B, e : ( )\}, c : (f : C, g : \{\,\})\}$$

is defined to be:

$$1_X = \alpha \left\{ \begin{array}{l} a \mapsto \overrightarrow{\beta}[a](1_A) \\ b \mapsto \overrightarrow{\beta}[b](\alpha \left\{ \begin{array}{l} d \mapsto \overrightarrow{\beta}[d](1_B) \\ e \mapsto \overrightarrow{\beta}[e](1_{()}) \end{array} \right\}) \\ c \mapsto \overrightarrow{\beta}[c](\beta \left(\begin{array}{l} f \mapsto \overrightarrow{\beta}[f](1_C) \\ g \mapsto \overrightarrow{\beta}[g](1_{\{\}}) \end{array} \right)) \end{array} \right\}$$

The proto-process resulting from the translations is as follows:

$$\begin{array}{|c||c|}\hline a & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[a], \qquad \begin{array}{|c||c|}\hline a & \overline{a} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash 1_A \qquad \begin{array}{|c||c|}\hline b & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[b], \qquad \begin{array}{|c||c|}\hline d & \\ b & \overline{b} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[d],$$

$$\begin{array}{|c||c|}\hline d & \overline{d} \\ b & \overline{b} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash 1_B \qquad \begin{array}{|c||c|}\hline e & \\ b & \overline{b} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[e], \qquad \begin{array}{|c||c|}\hline e & \overline{e} \\ b & \overline{b} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash 1_{()} \qquad \begin{array}{|c||c|}\hline c & \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\beta}[c],$$

$$\begin{array}{|c||c|}\hline & f \\ c & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\alpha}[f], \qquad \begin{array}{|c||c|}\hline \overline{f} & f \\ c & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash 1_C \qquad \begin{array}{|c||c|}\hline & g \\ c & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash \overline{\alpha}[g], \qquad \begin{array}{|c||c|}\hline \overline{g} & g \\ c & \overline{c} \\ \hline \alpha & \beta \\ \hline \end{array} \vdash 1_{\{\}}$$

5.2.2 Proto-processes to extensional processes

The purpose of this section is to prove:

Theorem 5.2.5 *If $\mathcal{P}$ is a proto-process then there exists a unique extensional process $\mathcal{Q}$ with $\mathcal{P} \subseteq \mathcal{Q}$.*

The idea will be to enlarge a proto-process with additional entailments in such a way that this new set of entailments will satisfy the rules EP-4 $\sim$ EP-7. If this can be accomplished, the new set of entailments will satisfy all the rules of an extensional process. To this end, we prove a series of lemmas (one for each additional EP rule)

which allows us to add entailments to a proto-process such that the resulting set of entailments is a proto-process.

Lemma 5.2.6 *Suppose that $\mathcal{P}_i$ is a set of entailments satisfying EP-i for $i \in \{1,2,3\}$. An application of EP-4 to the entailments of $\mathcal{P}_i$ produces a larger set of entailments $\mathcal{P}'_i$ which satisfies EP-i. That is, if $\mathfrak{p} \vdash \overline{\alpha}[a]$, $\mathfrak{q} \vdash \overline{\beta}[b] \in \mathcal{P}_i$, such that $\mathfrak{p} \sqsubseteq \mathfrak{q}$ and $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$, then $\mathcal{P}'_i = \mathcal{P}_i \cup \{\mathfrak{q} * \overline{\alpha}[a] \vdash \overline{\beta}[b]\}$ satisfies EP-i.*

Proof. For each $i \in \{1,2,3\}$, we must show that $\mathcal{P}'_i$ (defined above) satisfies EP-i.

EP-1 Our new antecedent $\mathfrak{q} * \overline{\alpha}[a]$ is $\mathcal{P}_1$-justified (as $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{P}$) and so it is $\mathcal{P}'_1$-justified.

EP-2 Suppose that $\mathfrak{r} \vdash \overline{\beta}[c] \in \mathcal{P}_2$ such that $\mathfrak{r}_\beta = (\mathfrak{q} * \overline{\alpha}[a])_\beta$ and $\mathfrak{r} \smile (\mathfrak{q} * \overline{\alpha}[a])$. Then $\mathfrak{r}_\beta = \mathfrak{q}_\beta$ and $\mathfrak{r} \smile \mathfrak{q}$ in $\mathcal{P}_2$ so that $b = c$.

EP-3 Adding an output event to an entailment does not increase the number of $\mathcal{P}'_3$-preantecedents and thus, $\mathcal{P}'_3$ satisfies EP-3. ∎

This allows us to conclude:

Corollary 5.2.7 *If $\mathcal{P}$ is a proto-process, the closure of $\mathcal{P}$ with respect to EP-4 is a proto-process.*

Lemma 5.2.8 *Suppose that $\mathcal{P}_i$ is a set of entailments satisfying EP-i for $i \in \{1,2,3\}$. An application of EP-5 to an entailment in $\mathcal{P}_i$ produces a larger set of entailments $\mathcal{P}'_i$ which satisfies EP-i. That is, if $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{P}_i$ and $\beta[b]$ is a legal input event given $\mathfrak{p}$, then $\mathcal{P}'_i = \mathcal{P}_i \cup \{\mathfrak{p} * \beta[b] \vdash \overline{\alpha}[a]\}$ satisfies EP-i.*

Proof. For each $i \in \{1,2,3\}$, we must show that $\mathcal{P}'_i$ (defined above) satisfies EP-i.

EP-1 Adding an input event to a $\mathcal{P}_1$-antecedents will not affect its $\mathcal{P}_1$-justification.

EP-2 Suppose $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{P}_2$ and that $\beta[b]$ is a legal input event given $\mathfrak{p}$. Adding this input to $\mathfrak{p}$ gives an entailment $\mathfrak{p} * \beta[b] \vdash \overline{\alpha}[a] \in \mathcal{P}'_2$. Now suppose that there is a $\mathfrak{q} \vdash \overline{\alpha}[b] \in \mathcal{P}_2$ such that $(\mathfrak{p} * \beta[b])_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} * \beta[b] \smile \mathfrak{q}$. Then clearly, $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} \smile \mathfrak{q}$ in $\mathcal{P}_2$ so that $a = b$.

EP-3 Adding input events does not increase the number of $\mathcal{P}_3$-preantecedents and thus, $\mathcal{P}'_3$ satisfies EP-3. ∎

Similarly here we may conclude:

Corollary 5.2.9 *If $\mathcal{P}$ is a proto-process, the closure of $\mathcal{P}$ with respect to EP-5 is a proto-process.*

Enlarging a proto-process with respect to EP-6 has a slightly different flavor from the rest in that it cannot be done one step at a time. We must take the closure of the set with respect to EP-6 in order for it to work. Another difference is that the set we wish to enlarge must satisfy all the rules of a proto-process, unlike the other lemmas.

Lemma 5.2.10 *Let $\mathcal{P}$ be a proto-process. The closure of $\mathcal{P}$ with respect to EP-6 produces a larger set of entailments $\mathcal{P}'$ which is itself a proto-process.*

Proof. We must show that $\mathcal{P}'$ satisfies EP-1 $\sim$ EP-3.

EP-1 Suppose $\mathfrak{p}$ is a $\mathcal{P}'$-antecedent and consider an output event $\overline{\alpha}[a] \in \mathfrak{p}$. That $\mathfrak{p}$ is a $\mathcal{P}'$-antecedent implies that there is an $\mathcal{P}$-antecedent $\mathfrak{p}'$ such that $\mathfrak{p} \sqsubseteq_o \mathfrak{p}'$. As $\mathcal{P}$-antecedents are justified $\overline{\alpha}[a]$ must have some justifying antecedent in $\mathcal{P}$, say $\mathfrak{u}$, i.e., $\mathfrak{u} \vdash \overline{\alpha}[a] \in \mathcal{P}$ such that $\mathfrak{u} \sqsubseteq \mathfrak{p}'$. In $\mathfrak{u}$ there may be some channels γ_j such that $\mathfrak{p}_{\gamma_j} \sqsubset_o \mathfrak{u}_{\gamma_j}$, but simply applying EP-6 to the output events on these channels gives an entailment $\mathfrak{u}' \vdash \overline{\alpha}[a] \in \mathcal{P}'$ such that $\mathfrak{u}' \sqsubseteq \mathfrak{p}$. Thus, $\overline{\alpha}[a]$ is justified in $\mathcal{P}'$.

EP-2 Suppose that $\mathfrak{p}' \vdash \overline{\alpha}[a],\ \mathfrak{q}' \vdash \overline{\alpha}[b] \in \mathcal{P}'$ with $\mathfrak{q}'_\alpha = \mathfrak{p}'_\alpha$ and $\mathfrak{q}' \smile \mathfrak{p}'$. As EP-6 removes output events this implies that there are entailments $\mathfrak{p} \vdash \overline{\alpha}[a],\ \mathfrak{q} \vdash \overline{\alpha}[b] \in \mathcal{P}$ such that $\mathfrak{p}' \sqsubseteq_o \mathfrak{p}$ and $\mathfrak{q}' \sqsubseteq_o \mathfrak{q}$. Clearly $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$, and by induction on the number of "incompatible" output events, we will show that $\mathfrak{p} \smile \mathfrak{q}$ so that $a = b$ follows.

If there are no incompatible output events then $\mathfrak{p} \smile \mathfrak{q}$, so assume that there are N incompatible output events. Consider a particular one, say $\overline{\beta}[c] \in \mathfrak{p}$ and $\overline{\beta}[d] \in \mathfrak{q}$, and their respective justifications in $\mathcal{P}$: $\mathfrak{u} \vdash \overline{\beta}[c]$ and $\mathfrak{s} \vdash \overline{\beta}[d]$

where $\mathfrak{u} \sqsubset \mathfrak{p}$ and $\mathfrak{s} \sqsubset \mathfrak{q}$. Now $\mathfrak{u}_\beta = \mathfrak{s}_\beta$ and $\mathfrak{u}$ and $\mathfrak{s}$ have less than N incompatible events so that by the inductive hypothesis $\mathfrak{r} \smile \mathfrak{s}$ and so $c = d$. This now implies that $\mathfrak{p}$ and $\mathfrak{p}$ have $N-1$ incompatible events, so that again applying the inductive hypothesis $\mathfrak{p} \smile \mathfrak{q}$.

EP-3 Removing output events does not increase the number of $\mathcal{P}$-preantecedents and so $\mathcal{P}'$ satisfies EP-3. ∎

Lemma 5.2.11 *Suppose that $\mathcal{P}_i$ is a set of entailments satisfying EP-i for $i \in \{1,2,3\}$. An application of EP-7 to an entailment in $\mathcal{P}_i$ produces a larger set of entailments $\mathcal{P}'_i$ which satisfies EP-i. That is, if $\mathfrak{p}$ is a $\mathcal{P}$-antecedent and $\mathfrak{p} * \beta[b_i] \vdash \overline{\alpha}[a] \in \mathcal{P}_i$ for every possible legal input b_i on β, then $\mathcal{P}'_i = \mathcal{P}_i \cup \{\mathfrak{p} \vdash \overline{\alpha}[a]\}$ satisfies EP-i.*

Proof. For each $i \in \{1,2,3\}$, we must show that $\mathcal{P}'_i$ (defined above) satisfies EP-i.

EP-1 The new antecedent $\mathfrak{p}$ is justified in $\mathcal{P}_1$, and so is justified in $\mathcal{P}'_1$. Therefore, $\mathcal{P}'_1$ satisfies EP-1.

EP-2 Suppose that $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{P}'_2$ and $\mathfrak{q} \vdash \overline{\alpha}[b] \in \mathcal{P}_2$ with $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} \smile \mathfrak{q}$. Then as $\mathfrak{p} * \beta[b] \vdash \overline{\alpha}[a]$ for all possible inputs b_i on β, there is some event b_k such that in $\mathcal{P}_2$, $(\mathfrak{p} * \beta[b_k])_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} * \beta[b_k] \smile \mathfrak{q}$. It now follows that $a = b$.

EP-3 It suffices to show that any saturated $\mathcal{P}'_3$-preantecedent is itself a $\mathcal{P}'_3$-antecedent. This follows since given any saturated $\mathcal{P}'_3$-preantecedent $\mathfrak{q}$, it must also be a saturated $\mathcal{P}_3$-preantecedent. If input events were removed from $\mathfrak{q}$ then the resulting behaviour would no longer be saturated. So, applying EP-3 in $\mathcal{P}_3$ to $\mathfrak{q}$ implies that $\mathfrak{q}$ is a $\mathcal{P}_3$-antecedent and therefore, a $\mathcal{P}'_3$-antecedent. ∎

We conclude:

Corollary 5.2.12 *If $\mathcal{P}$ is a proto-process, the closure of $\mathcal{P}$ with respect to EP-7 is a proto-process.*

The previous four lemmas show that a proto-process $\mathcal{P}$ may be enlarged to a set of entailments $\mathcal{Q}$, so that $\mathcal{Q}$ is itself a proto-process which additionally satisfies EP-4 through EP-7, i.e., an extensional process.

To complete the proof of Theorem 5.2.5, it is necessary to show that that this enlarging procedure results in a unique extensional process. To this end, suppose that $\mathcal{P}$ is a proto-process and $\mathcal{Q}$ is any extensional process such that $\mathcal{P} \subseteq \mathcal{Q}$ and denote by $\mathcal{E}(\mathcal{P})$ the extensional process which results from enlarging $\mathcal{P}$. If $\mathfrak{p} \vdash \times \in \mathcal{E}(\mathcal{P})$ is the result of closing $\mathcal{P}$ with respect to some EP-rule, then, as $\mathcal{P} \subseteq \mathcal{Q}$, it must be in $\mathcal{Q}$ as well. Therefore, $\mathcal{E}(\mathcal{P}) \subseteq \mathcal{Q}$ and by Proposition 5.1.14, $\mathcal{E}(\mathcal{P}) = \mathcal{Q}$.

This now completes the proof of Theorem 5.2.5.

5.3 The polycategory of extensional processes

In this section we establish that protocols and extensional processes form a polycategory which we will denote by $\mathcal{EP}$. We begin by introducing some new notation. Let $\mathfrak{p}$ and $\mathfrak{q}$ be behaviours. If γ is a channel in $\mathfrak{p}$, the notation $\mathfrak{p}\backslash\gamma$ is used to represent the behaviour $\mathfrak{p}$ with all the events on γ (including the channel name) removed. For example,

$$\mathfrak{p} = \begin{array}{|c|c||c|c|} \hline \overline{a} & & c & e \\ b & & \overline{d} & \overline{f} \\ \hline \alpha & & \beta & \gamma \\ \hline \end{array} \qquad \mathfrak{p}\backslash\gamma = \begin{array}{|c|c||c|} \hline \overline{a} & & c \\ b & & \overline{d} \\ \hline \alpha & & \beta \\ \hline \end{array}$$

The notation $\mathfrak{p}||\mathfrak{q}$ is used to denote the behaviour whose input channel behaviours are the input channel behaviours of $\mathfrak{p}$ and $\mathfrak{q}$ and whose output channel behaviours are the output channel behaviours of $\mathfrak{p}$ and $\mathfrak{q}$ (assuming $\mathfrak{p}$ and $\mathfrak{q}$ have distinct channel names). For example,

$$\mathfrak{p} = \begin{array}{|c|c||c|} \hline \overline{a} & & c \\ b & & \overline{d} \\ \hline \alpha & & \beta \\ \hline \end{array} \qquad \mathfrak{q} = \begin{array}{|c|c||c|} \hline e & & \overline{g} \\ \overline{f} & & h \\ \hline \gamma & & \delta \\ \hline \end{array} \qquad \mathfrak{p}||\mathfrak{q} = \begin{array}{|c|c|c||c|} \hline \overline{a} & e & c & \overline{g} \\ b & \overline{f} & \overline{d} & h \\ \hline \alpha & \gamma & \beta & \delta \\ \hline \end{array}$$

We are now ready to show how to compose extensional processes.

5.3.1 Composition of extensional processes

Here, as in the case with syntactic processes (see Section 3.3.2), in order to compose two extensional processes $\mathcal{F}$ and $\mathcal{G}$ they must have exactly one channel name in common. This means that we must rename channels so that the channel names in $\mathcal{F}$ and $\mathcal{G}$ are distinct except for the channel on which they will be composed. This may be accomplished through channel name substitution, however, as we did with syntactic processes, we will assume that the channel names of any two extensional processes are distinct unless otherwise specified.

Definition 5.3.1 (Composition of extensional processes) Let

$$\mathcal{F} : \Gamma \to \gamma : X, \Delta \qquad \text{and} \qquad \mathcal{G} : \Gamma', \gamma : X \to \Delta'$$

be extensional processes. The set of entailments representing the composite of $\mathcal{F}$ and $\mathcal{G}$ on γ, denoted $\mathcal{F};_\gamma \mathcal{G} : \Gamma, \Gamma' \to \Delta, \Delta'$, is defined to be:

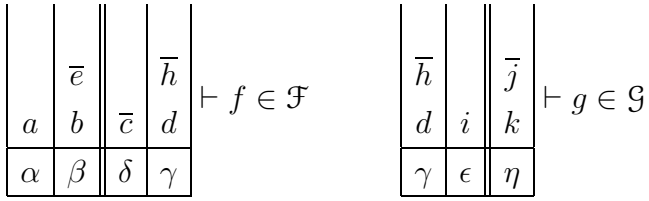
$$\begin{aligned}\mathcal{F};_\gamma \mathcal{G} = &\left\{(\mathfrak{p}\backslash\gamma||\mathfrak{q}\backslash\gamma) \vdash f \;;_\gamma g \;\mid\; \mathfrak{p} \vdash f \in \mathcal{F} \wedge \mathfrak{q} \vdash g \in \mathcal{G} \wedge \mathfrak{p}_\gamma = \mathfrak{q}^*_\gamma\right\} \\ &\cup \left\{(\mathfrak{p}\backslash\gamma||\mathfrak{q}\backslash\gamma) \vdash \overline{\alpha}[a] \;\mid\; \mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{F} \wedge \mathfrak{q} \text{ a } \mathcal{G}\text{-preantecedent} \wedge \mathfrak{p}_\gamma = \mathfrak{q}^*_\gamma\right\} \\ &\cup \left\{(\mathfrak{p}\backslash\gamma||\mathfrak{q}\backslash\gamma) \vdash \overline{\beta}[a] \;\mid\; \mathfrak{p} \text{ a } \mathcal{F}\text{-preantecedent} \wedge \mathfrak{q} \vdash \overline{\beta}[a] \in \mathcal{G} \wedge \mathfrak{p}_\gamma = \mathfrak{q}^*_\gamma\right\}\end{aligned}$$

In order to simplify the notation, the shorthand $\mathfrak{p};_\gamma \mathfrak{q}$ will be used to denote $\mathfrak{p}\backslash\gamma||\mathfrak{q}\backslash\gamma$ provided that $\mathfrak{p}_\gamma = \mathfrak{q}^*_\gamma$.

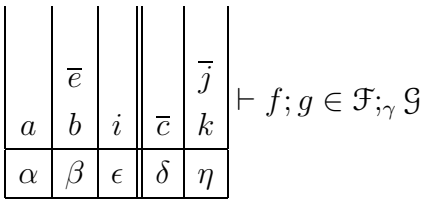
Composition of extensional processes may become clearer with a couple of examples.

Example 5.3.2 (Composing extensional processes)

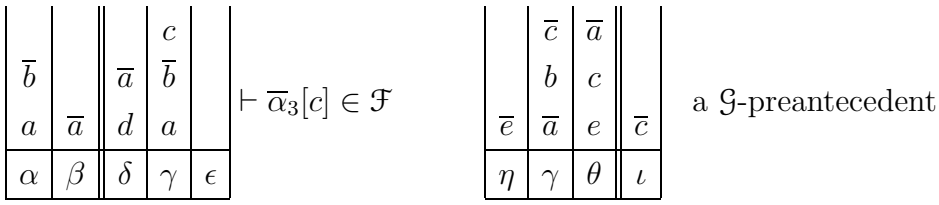
1. The following two atomic entailments may be composed on γ,



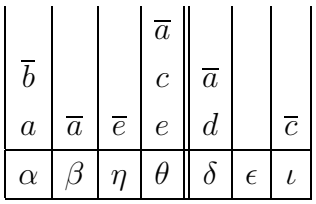
with their composite given by:



2. The following entailment in $\mathcal{F}$ and $\mathcal{G}$-preantecedent may be composed on γ,



with their composite given by:



Lemma 5.3.3 *If* $\mathfrak{p} ;_\gamma \mathfrak{q} \vdash \times$, $\mathfrak{p}' ;_\gamma \mathfrak{q}' \vdash \times' \in \mathcal{F};_\gamma \mathcal{G}$ *such that* $\mathfrak{p} ;_\gamma \mathfrak{q} \smile \mathfrak{p}' ;_\gamma \mathfrak{q}'$, *then* $\mathfrak{p} \smile \mathfrak{p}'$ *and* $\mathfrak{q} \smile \mathfrak{q}'$.

Proof. We will show that $\mathfrak{p}$ and $\mathfrak{p}'$ are compatible; a similar argument can be used to show that $\mathfrak{q}$ and $\mathfrak{q}'$ are compatible.

The behaviours $\mathfrak{p}$ and $\mathfrak{p}'$ are compatible on channels other than γ by definition, so what needs to be shown is that they are compatible on γ. To this end, suppose they are not compatible on γ and consider the first position in which they differ, say by the events $a \in \mathfrak{p}$ and $b \in \mathfrak{p}'$ in $\mathcal{F}$ (dually $\overline{a} \in \mathfrak{q}$ and $\overline{b} \in \mathfrak{q}'$ in $\mathcal{G}$). There are two cases to consider:

(i) a and b are output events. Consider their justifications $\mathfrak{u} \vdash \overline{\gamma}[a] \in \mathfrak{p}$ and $\mathfrak{u}' \vdash \overline{\gamma}[b] \in \mathfrak{p}'$. Clearly, $\mathfrak{u}_\gamma = \mathfrak{u}'_\gamma$ and $\mathfrak{u} \smile \mathfrak{u}'$ so that by EP-2, $a = b$.

(ii) a and b are input events. This implies that they are output events in $\mathfrak{q}$ and $\mathfrak{q}'$ and moreover, this must be the first place in which they differ on γ. Therefore, by repeating the argument used in (i) in $\mathcal{G}$, $a = b$

Thus, $\mathfrak{p}$ and $\mathfrak{p}'$ are compatible. ∎

Proposition 5.3.4 *Let $\mathcal{F} : \Gamma \to \gamma : X, \Delta$ and $\mathcal{G} : \Gamma', \gamma : X \to \Delta'$ be extensional processes. The set of entailments $\mathcal{F};_\gamma \mathcal{G} : \Gamma, \Gamma' \to \Delta, \Delta'$ forms an extensional process.*

Proof. We must show that $\mathcal{F};_\gamma \mathcal{G}$ satisfies EP-1 through EP-7.

EP-1 Suppose $\mathfrak{p};_\gamma \mathfrak{q}$ is a $\mathcal{F};_\gamma \mathcal{G}$-antecedent and $\overline{\alpha}[a]$ is an output event in $\mathfrak{p};_\gamma \mathfrak{q}$; without loss of generality assume that $\alpha \in \mathcal{F}$. It follows from the definition of composition that $\mathfrak{p}$ is justified in $\mathcal{F}$ and $\mathfrak{q}$ is justified in $\mathcal{G}$ so that $\overline{\alpha}[a]$ is justified in $\mathfrak{p}$, say by $\mathfrak{u} \vdash \overline{\alpha}[a]$.

By EP-6 (in $\mathcal{F}$), output events on γ may be removed from $\mathfrak{u}$ so that either $\mathfrak{u}_\gamma = \emptyset$ or the last event in $\mathfrak{u}_\gamma$ is an input event.

If $\mathfrak{u}_\gamma = \emptyset$ then $\mathfrak{u} \vdash \overline{\alpha}[a]$ justifies $\overline{\alpha}[a]$ in $\mathcal{F};_\gamma \mathcal{G}$. If $\mathfrak{u}_\gamma \neq \emptyset$, then in $\mathcal{F}$ the last event on γ is an input event $\gamma[c]$, which means that it is an output event in $\mathfrak{q}$. Moreover, it is justified in $\mathfrak{q}$, say by $\mathfrak{u}' \vdash \overline{\gamma}[c]$, and so $\mathfrak{u}' * \overline{\gamma}[c]$ is a justified behaviour in $\mathcal{G}$. Therefore, $\mathfrak{u};_\gamma (\mathfrak{u}' * \gamma[c]) \vdash \overline{\alpha}[a] = \mathfrak{u};_\gamma \mathfrak{u}' \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$ and $\mathfrak{u};_\gamma \mathfrak{u}' \sqsubseteq \mathfrak{p};_\gamma \mathfrak{q}$ so that $\overline{\alpha}[a]$ is justified in $\mathcal{F};_\gamma \mathcal{G}$.

If $\alpha \in \mathcal{G}$ a similar argument in which we reverse the components shows that $\overline{\alpha}[a]$ is justified in $\mathcal{F};_\gamma \mathcal{G}$.

EP-2 Suppose that $\mathfrak{p};_\gamma \mathfrak{q} \vdash \overline{\alpha}[a],\ \mathfrak{p}';_\gamma \mathfrak{q}' \vdash \overline{\alpha}[b] \in \mathcal{F};_\gamma \mathcal{G}$ such that $(\mathfrak{p};_\gamma \mathfrak{q})_\alpha = (\mathfrak{p}';_\gamma \mathfrak{q}')_\alpha$ and $\mathfrak{p};_\gamma \mathfrak{q} \sqsubseteq \mathfrak{p}';_\gamma \mathfrak{q}'$. Without loss of generality assume that $\alpha \in \mathcal{F}$. This implies $\mathfrak{p} \vdash \overline{\alpha}[a],\ \mathfrak{p}' \vdash \overline{\alpha}[b] \in \mathcal{F}$ such that $\mathfrak{p}_\alpha = \mathfrak{p}_\alpha$ and by Lemma 5.3.3, $\mathfrak{p} \smile \mathfrak{p}'$ so that by EP-2 in $\mathcal{F}$, $a = b$.

$\alpha \in \mathcal{G}$ is handled similarly by considering $\mathfrak{q} \vdash \overline{\alpha}[a],\ \mathfrak{q}' \vdash \overline{\alpha}[b] \in \mathcal{G}$.

EP-3 Suppose that $\mathfrak{p}||\mathfrak{p}'$ is a $\mathcal{F};_\gamma \mathcal{G}$-preantecedent and $\mathfrak{q}||\mathfrak{q}'$ is a saturated $\mathcal{F}\ ;_\gamma \mathcal{G}$-preantecedent such that $\mathfrak{p}||\mathfrak{p}' \sqsubseteq_i \mathfrak{q}||\mathfrak{q}'$.

Consider (the possible unjustified) behaviours $\mathfrak{p}$ in $\mathcal{F}$ and $\mathfrak{p}'$ in $\mathcal{G}$. Any output event $\overline{\alpha}[a_i] \in \mathfrak{p}$ must be justified in $\mathcal{F};_\gamma \mathcal{G}$, say by $\mathfrak{u}_i;_\gamma \mathfrak{u}_i'$ so that $\mathfrak{u}_i \vdash \overline{\alpha}[a_i]$ justifies $\overline{\alpha}[a_i]$ in $\mathcal{F}$. Similarly we may justify output events in $\mathfrak{p}'$ in $\mathcal{G}$.

Choose the justification $\mathfrak{u};_\gamma \mathfrak{u}'$ such that $\mathfrak{u}^*_\gamma = \mathfrak{u}'_\gamma$ is maximal. By Lemma 5.3.3, all the $\mathfrak{u}_i$'s are compatible and all the $\mathfrak{u}_i'$'s are compatible so that $\mathfrak{p} \vee \mathfrak{u}$ and $\mathfrak{p}' \vee \mathfrak{u}'$ are respectively $\mathcal{F}$ and $\mathcal{G}$-preantecedents. Notice that also $\mathfrak{q} \vee \mathfrak{u}$ and $\mathfrak{q}' \vee \mathfrak{u}'$ are also $\mathcal{F}$ and $\mathcal{G}$-preantecedents respectively (as they are separated from $\mathfrak{p}$ and $\mathfrak{p}'$ by input events). Consider the last state on $\mathfrak{u}_\gamma$: it may be atomic, an input state, or an output state.

(i) Suppose it is atomic. In this case, at least one of $\mathfrak{q} \vee \mathfrak{u}$ and $\mathfrak{q}' \vee \mathfrak{u}'$ is saturated; they both may be. Suppose both are saturated. If they are both atomic saturated, by EP-3, this implies that $\mathfrak{q} \vee \mathfrak{u} \vdash f \in \mathcal{F}$ and $\mathfrak{q}' \vee \mathfrak{u}' \vdash g \in \mathcal{G}$, and therefore, $(\mathfrak{q} \vee \mathfrak{u});_\gamma (\mathfrak{q}' \vee \mathfrak{u}') \vdash f; g = \mathfrak{q}||\mathfrak{q}' \vdash f; g \in \mathcal{F};_\gamma \mathcal{G}$, which satisfies EP-3 in $\mathcal{F};_\gamma \mathcal{G}$.

It may also be the case that one is atomic saturated and the other is (non-atomic) saturated. Suppose that $\mathfrak{q}' \vee \mathfrak{u}'$ is atomic saturated and $\mathfrak{q} \vee \mathfrak{u}$ is (non-atomic) saturated. By EP-3, $\mathfrak{q} \vee \mathfrak{u} \vdash \overline{\alpha}[a] \in \mathcal{F}$ for some output event $\overline{\alpha}[a]$ where $\alpha \neq \gamma$. Thus $(\mathfrak{q} \vee \mathfrak{u});_\gamma (\mathfrak{q}' \vee \mathfrak{u}') \vdash \overline{\alpha}[a] = \mathfrak{q}||\mathfrak{q}' \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$ which satisfies EP-3.

Now suppose only one is saturated and without loss of generality suppose that it is $\mathfrak{q} \vee \mathfrak{u}$ which is saturated. By EP-3 again, this implies that $\mathfrak{q} \vee \mathfrak{u} \vdash \overline{\alpha}[a] \in \mathcal{F}$ for some output event $\overline{\alpha}[a]$, and thus, $(\mathfrak{q} \vee \mathfrak{u});_\gamma (\mathfrak{q}' \vee \mathfrak{u}') \vdash \overline{\alpha}[a] = \mathfrak{q}||\mathfrak{q}' \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$ satisfying EP-3.

(ii) Suppose the last event on $\mathfrak{u}_\gamma$ is not atomic. Then, one of $\mathfrak{q} \vee \mathfrak{u}$ and $\mathfrak{q}' \vee \mathfrak{u}'$ is saturated. Without loss of generality suppose that $\mathfrak{q} \vee \mathfrak{u}$ is saturated. By EP-3, this implies that $\mathfrak{q} \vee \mathfrak{u} \vdash \overline{\alpha}[a] \in \mathcal{F}$ for some output event $\overline{\alpha}[a]$. If $\alpha = \gamma$ then $\mathfrak{u} * \overline{\gamma}[a]$ and $\mathfrak{u} * \gamma[a]$ are respectively $\mathcal{F}$ and $\mathcal{G}$-preantecedents such that $(\mathfrak{u} * \overline{\gamma}[a]);_\gamma (\mathfrak{u} * \gamma[a])$ is saturated in $\mathcal{F};_\gamma \mathcal{G}$. Thus, we may apply this argument on this preantecedent recursively to get an $\mathcal{F};_\gamma \mathcal{G}$-antecedent which satisfies EP-3.

So suppose that $\alpha \neq \gamma$. Then $(\mathfrak{q} \vee \mathfrak{u});_\gamma (\mathfrak{q}' \vee \mathfrak{u}') \vdash \overline{\alpha}[a] = \mathfrak{q} || \mathfrak{q}' \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$ satisfying EP-3.

EP-4 Suppose that $\mathfrak{p};_\gamma \mathfrak{q} \vdash \overline{\alpha}[a],\ \mathfrak{p}';_\gamma \mathfrak{q}' \vdash \overline{\beta}[b] \in \mathcal{F};_\gamma \mathcal{G}$ with $(\mathfrak{p};_\gamma \mathfrak{q})_\alpha = (\mathfrak{p}';_\gamma \mathfrak{q}')_\alpha$ and $\mathfrak{p};_\gamma \mathfrak{q} \sqsubseteq \mathfrak{p}';_\gamma \mathfrak{q}'$. Without loss of generality suppose that $\alpha \in \mathcal{F}$. There are two cases depending on whether $\beta \in \mathcal{F}$ or $\beta \in \mathcal{G}$.

(i) Suppose $\beta \in \mathcal{F}$. This implies that $\mathfrak{q}$ and $\mathfrak{q}'$ are compatible $\mathcal{G}$-preantecedents so that by Lemma 5.1.10, $\mathfrak{q} \vee \mathfrak{q}'$ is a $\mathcal{G}$-preantecedent. Similarly, by Corollary 5.1.13, $\mathfrak{p} \vee \mathfrak{p}' \vdash \overline{\beta}[b] \in \mathcal{F}$. Notice that $(\mathfrak{p} \vee \mathfrak{p}')_\alpha = (\mathfrak{q} \vee \mathfrak{q}')_\alpha$. Then, $\mathfrak{p} \vdash \overline{\alpha}[a],\ \mathfrak{p} \vee \mathfrak{p}' \vdash \overline{\beta}[b] \in \mathcal{F}$ such that $\mathfrak{p}_\alpha = \mathfrak{p}'_\alpha$ and $\mathfrak{p} \sqsubseteq \mathfrak{p} \vee \mathfrak{p}'$ so that by EP-4, $\mathfrak{p} \vee \mathfrak{p}' * \overline{\alpha}[a] \vdash \overline{\beta}[b] \in \mathcal{F}$. Composing with $\mathfrak{q} \vee \mathfrak{q}'$ yields $(\mathfrak{p} \vee \mathfrak{p}' * \overline{\alpha}[a]);_\gamma (\mathfrak{q} \vee \mathfrak{q}') \vdash \overline{\beta}[b] = (\mathfrak{p}';_\gamma \mathfrak{q}') * \overline{\alpha}[a] \vdash \overline{\beta}[b] \in \mathcal{F};_\gamma \mathcal{G}$.

(ii) Suppose $\beta \in \mathcal{G}$. By Corollary 5.1.13, $\mathfrak{p} \vee \mathfrak{p}' \vdash \overline{\alpha}[a] \in \mathcal{F}$ and $\mathfrak{q} \vee \mathfrak{q}' \vdash \overline{\beta}[b] \in \mathcal{G}$. This implies that $(\mathfrak{p} \vee \mathfrak{p}') * \overline{\alpha}[a]$ is a $\mathcal{F}$-preantecedent, and as $(\mathfrak{p} \vee \mathfrak{p}')_\alpha = (\mathfrak{q} \vee \mathfrak{q}')_\alpha$ we may form $(\mathfrak{p} \vee \mathfrak{p}' * \overline{\alpha}[a]);_\gamma (\mathfrak{q} \vee \mathfrak{q}') \vdash \overline{\beta}[b] = (\mathfrak{p}';_\gamma \mathfrak{q}') * \overline{\alpha}[a] \vdash \overline{\beta}[b] \in \mathcal{F};_\gamma \mathcal{G}$.

EP-5 Let $\mathfrak{p};_\gamma \mathfrak{q} \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$. If $\beta[b]$ is a legal input event for this entailment, then it is certainly a legal input event in its corresponding component so that $(\mathfrak{p};_\gamma \mathfrak{q}) * \beta[b] \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$.

EP-6 Suppose $(\mathfrak{p};_\gamma \mathfrak{q}) * \overline{\beta}[b] \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$. Without loss of generality assume that $\alpha \in \mathcal{F}$. There are two cases depending on whether $\beta \in \mathcal{F}$ or $\beta \in \mathcal{G}$.

(i) If $\beta \in \mathcal{F}$ then $\mathfrak{p} * \overline{\beta}[b] \vdash \overline{\alpha}[a] \in \mathcal{F}$, which, by EP-6, implies that $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{F}$. Therefore, $\mathfrak{p};_\gamma \mathfrak{q} \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$.

(ii) If $\beta \in \mathcal{G}$, by the definition of composition, $\mathfrak{q} * \overline{\beta}[b]$ must be a $\mathcal{G}$-preantecedent and therefore, $\mathfrak{q}$ must also be a $\mathcal{G}$-preantecedent (if $\mathfrak{q} * \overline{\beta}[b]$ is any $\mathcal{G}$-antecedent, by EP-6, $\mathfrak{r}$ must also be a $\mathcal{G}$-antecedent). Thus, $\mathfrak{p};_\gamma \mathfrak{q} \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$.

The case where $\alpha \in \mathcal{G}$ is handled similarly.

EP-7 Suppose that $\{\mathfrak{r} * \beta[b_i] \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G} \mid i \in I\}$, where $\{b_i \mid i \in I\}$ is the set of all possible input events on β given $\mathfrak{r}$. Without loss of generality suppose that $\alpha \in \mathcal{F}$. There are two cases depending on whether $\beta \in \mathcal{F}$ or $\beta \in \mathcal{G}$.

(i) If $\beta \in \mathcal{F}$ then $\mathfrak{r} * \beta[b_i] \vdash \overline{\alpha}[a]$ came from $\mathfrak{p}_i * \beta[b_i] \vdash \overline{\alpha}[a] \in \mathcal{F}$ and a $\mathcal{G}$-preantecedent $\mathfrak{q}_i$, for each $i \in I$. We know that all the $\mathfrak{p}_i$ and $\mathfrak{q}_i$ are justified and all the $\mathfrak{p}_i$ are compatible and all the $\mathfrak{q}$ are compatible (Lemma 5.3.3). Thus, we may form $\bigvee_i \mathfrak{p}_i$ and $\bigvee_i \mathfrak{q}_i$ which are $\mathcal{F}$ and $\mathcal{G}$-preantecedents respectively (Corollary 5.1.13). Clearly then, $(\bigvee_i \mathfrak{p}_i) * \beta[b_i] \vdash \overline{\alpha}[a] \in \mathcal{F}$ for $i \in I$, and therefore, $\bigvee_i \mathfrak{p}_i \vdash \overline{\alpha}[a] \in \mathcal{F}$ by EP-7. This implies that $(\bigvee_i \mathfrak{p}_i);_\gamma (\bigvee_i \mathfrak{q}_i) \vdash \overline{\alpha}[a] = \mathfrak{r} \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$.

(ii) If $\beta \in \mathcal{F}$ then $\mathfrak{r} * \beta[b_i] \vdash \overline{\alpha}[a]$ came from $\mathfrak{p}_i \vdash \overline{\alpha}[a] \in \mathcal{F}$ and a $\mathcal{G}$-preantecedent $\mathfrak{q}_i * \beta[b_i]$, for each $i \in I$. By definition, all the $\mathfrak{q}_i$'s are justified so that each $\mathfrak{q}_i$ is a $\mathcal{G}$-preantecedent, and therefore, $\mathfrak{p}_i;_\gamma \mathfrak{q}_i \vdash \overline{\alpha}[a] = \mathfrak{r} \vdash \overline{\alpha}[a] \in \mathcal{F};_\gamma \mathcal{G}$, for any $i \in I$.

We have shown that the set of entailments $\mathcal{F};_\gamma \mathcal{G}$ satisfies EP-1 $\sim$ EP-7 and is, therefore, an extensional process. ∎

Next it is shown that the identity extensional process behaves correctly. We begin with a remark on the structure of the identity extensional process.

Remark 5.3.5 Let $\alpha : X \xrightarrow{1_X} \beta : X$ be the identity extensional process. If $\mathfrak{p}$ is a legal sequence of events for X in the domain that ends at:

- the last state (atomic) Z, then $\mathfrak{p}||\mathfrak{p}^* \vdash 1_Z \in 1_X$.
- an output event $\overline{\alpha}[a]$, then $(\mathfrak{p}\backslash\overline{a})||\mathfrak{p}^* \vdash \overline{\alpha}[a] \in 1_X$.
- an input event $\alpha[a]$, then $\mathfrak{p}||(\mathfrak{p}\backslash a)^* \vdash \overline{\beta}[a] \in 1_X$.

This follows directly from the definition of the identity process and the translation from a process to an extensional process (see Section 5.2.2).

Proposition 5.3.6 *$\alpha : X \xrightarrow{1_X} \beta : X$ is the identity extensional process on X. That is, given an extensional process $\mathcal{F} : \Gamma \to \alpha : X, \Delta$, the composite $\mathcal{F};_\alpha 1_X = \mathcal{F}$ (up to a renaming of channels), and given an extensional process $\mathcal{G} : \Gamma, \beta : X \to \Delta$, the composite $1;_\beta \mathcal{G} = \mathcal{G}$ (up to a renaming of channels).*

Proof. We will prove that $\mathcal{F} = \mathcal{F};_\alpha 1_X$; the statement $\mathcal{G} = 1_X;_\beta \mathcal{G}$ is dual. The idea will be to prove that $\mathcal{F} \subseteq \mathcal{F};_\alpha 1_X$, which, by Proposition 5.1.14, then implies that $\mathcal{F} = \mathcal{F};_\alpha 1_X$. To this end, let $\mathfrak{p} \vdash \times \in \mathcal{F}$. The goal is to show that there is a entailment $\mathfrak{p}' \vdash \times' \in \mathcal{F};_\alpha 1_X$ such that if α is substituted for β in this entailment we get back $\mathfrak{p} \vdash \times \in \mathcal{F}$.

There are three cases to consider.

(i) $\times = f$, where f is atomic. This implies that the last state of $\mathfrak{p}_\alpha$ is atomic, say Z. As $\mathfrak{p}_\alpha$ is a legal sequence of events for X (in the codomain) that ends at atomic Z, by Remark 5.3.5, $\mathfrak{p}^*_\alpha || \mathfrak{p}_\alpha \vdash 1_Z \in 1_X$, and therefore $\mathfrak{p};_\alpha (\mathfrak{p}^*_\alpha || \mathfrak{p}_\alpha) \vdash f; 1_Z = (\mathfrak{p} \backslash \alpha) || \mathfrak{p}_\alpha \vdash f \in \mathcal{F};_\alpha 1_X$, which is equivalent to $\mathfrak{p} \vdash f$ up to a renaming of β to α.

(ii) $\times = \overline{\alpha}[a]$. This implies that $\mathfrak{p} * \overline{\alpha}[a]$ is a $\mathcal{F}$-preantecedent and that $\mathfrak{p}^*_\alpha * \alpha[a]$ is a legal sequence of events for X (in the domain). By our remark above, this implies that $(\mathfrak{p}^*_\alpha * \alpha[a]) || \mathfrak{p}_\alpha \vdash \overline{\beta}[a] \in 1_X$. Therefore, $(\mathfrak{p} * \overline{\alpha}[a]);_\alpha ((\mathfrak{p}^*_\alpha * \alpha[a]) || \mathfrak{p}_\alpha) \vdash \overline{\beta}[a] = (\mathfrak{p} \backslash \alpha) || \mathfrak{p}_\alpha \vdash \overline{\beta}[a] \in \mathcal{F};_\alpha 1_X$, which is equivalent to $\mathfrak{p} \vdash \overline{\alpha}[a]$ up to a renaming of β to α.

(iii) $\times = \overline{\gamma}[c]$, where $\gamma \neq \alpha$. In this case we must consider events in $\mathfrak{p}$ on α.

- If there are no events on α, then $\mathfrak{p} \vdash \overline{\gamma}[c] \in \mathcal{F};_\alpha 1_X$.
- Suppose that the last event on α is an output $\overline{\alpha}[a]$. This implies that it is an input event in 1_X and that $\mathfrak{p}^*_\alpha * \alpha[a]$ is a legal sequence of events for X (in the domain) so that, by our remark above, $(\mathfrak{p}^*_\alpha * \alpha[a]) || \mathfrak{p}_\alpha \vdash \overline{\beta}[a] \in 1_X$. Hence, $(\mathfrak{p}^*_\alpha * \alpha[a]) || (\mathfrak{p}_\alpha * \overline{\beta}[a])$ is a 1_X-preantecedent, and

$\mathfrak{p};_\alpha ((\mathfrak{p}^*_\alpha * \alpha[a])||(\mathfrak{p}_\alpha * \overline{\beta}[a])) \vdash \overline{\gamma}[c] = (\mathfrak{p}\backslash\alpha)||(\mathfrak{p}_\alpha * \overline{\beta}[a]) \vdash \overline{\gamma}[c] \in \mathcal{F};_\alpha 1_X$, which is equivalent to $\mathfrak{p} \vdash \overline{\gamma}[c]$ up to a renaming of β to α.

- Suppose that the last event on α is an input $\alpha[a]$. This implies that it is an output event in 1_X and that $\mathfrak{p}^*_\alpha * \overline{\alpha}[a]$ is a legal sequence of events for X (in the domain) so that, by our remark above, $\mathfrak{p}^*_\alpha||(\mathfrak{p}_\alpha * \beta[a]) \vdash \overline{\alpha}[a] \in 1_X$. Then, $(\mathfrak{p}^*_\alpha * \overline{\alpha}[a])||(\mathfrak{p}_\alpha * \beta[a])$ is a 1_X-preantecedent, and therefore, $\mathfrak{p};_\alpha ((\mathfrak{p}^*_\alpha * \overline{\alpha}[a])||(\mathfrak{p}_\alpha * \beta[a])) \vdash \overline{\gamma}[c] = (\mathfrak{p}\backslash\alpha)||(\mathfrak{p}_\alpha * \beta[a]) \vdash \overline{\gamma}[c] \in \mathcal{F};_\alpha 1_X$, which is equivalent to $\mathfrak{p} \vdash \overline{\gamma}[c]$ up to a renaming of β to α. ∎

It is left to show that composition of extensional processes is associative and satisfies the interchange law.

Proposition 5.3.7 *Composition of extensional processes is associative. That is, if*

$$\mathcal{F} : \Gamma \to \gamma : X, \Delta \qquad \mathcal{G} : \Gamma', \gamma : X \to \delta : Y, \Delta' \qquad \mathcal{H} : \Gamma'', \delta : Y \to \Delta''$$

are extensional processes, then $(\mathcal{F} ;_\gamma \mathcal{G}) ;_\delta \mathcal{H} = \mathcal{F} ;_\gamma (\mathcal{G} ;_\delta \mathcal{H})$.

Proof. Composition of atomic entailments is associative as composition is associative in the underlying polycategory. That is, if $\mathfrak{p} \vdash (f;_\gamma g);_\delta h \in (\mathcal{F} ;_\gamma \mathcal{G}) ;_\delta \mathcal{H}$, then $\mathfrak{p} \vdash f;_\gamma (g;_\delta h) \in \mathcal{F} ;_\gamma (\mathcal{G} ;_\delta \mathcal{H})$.

To show that the composition of non-atomic entailments is associative suppose that $\alpha \in \mathcal{F}$ and consider the entailment

$$(\mathfrak{p};_\gamma \mathfrak{q});_\delta \mathfrak{r} \vdash \overline{\alpha}[a] \in (\mathcal{F} ;_\gamma \mathcal{G}) ;_\delta \mathcal{H}$$

This implies that $\mathfrak{r}$ is a $\mathcal{H}$-preantecedent and $\mathfrak{p};_\gamma \mathfrak{q} \vdash \overline{\alpha}[a] \in \mathcal{F} ;_\gamma \mathcal{G}$ such that $\mathfrak{q}_\delta = \mathfrak{r}^*_\delta$. This in turn implies that $\mathfrak{q}$ is a $\mathcal{G}$-preantecedent and $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{F}$ such that $\mathfrak{p}_\gamma = \mathfrak{q}^*_\gamma$. From these observations it should now be clear that $\mathfrak{q};_\delta \mathfrak{r}$ is a $\mathcal{G};_\delta \mathcal{H}$-preantecedent so that

$$\mathfrak{p};_\gamma (\mathfrak{q};_\delta \mathfrak{r}) \vdash \overline{\alpha}[a] \in \mathcal{F} ;_\gamma (\mathcal{G} ;_\delta \mathcal{H})$$

The argument is similar if $\alpha \in \mathcal{H}$, and slightly different if $\alpha \in \mathcal{G}$. We describe the latter case and so suppose $\alpha \in \mathcal{G}$ and consider

$$(\mathfrak{p};_\gamma \mathfrak{q});_\delta \mathfrak{r} \vdash \overline{\alpha}[a] \in (\mathcal{F} ;_\gamma \mathcal{G}) ;_\delta \mathcal{H}$$

This implies that $\mathfrak{r}$ is a $\mathcal{H}$-preantecedent and $\mathfrak{p};_\gamma \mathfrak{q} \vdash \overline{\alpha}[a] \in \mathcal{F} ;_\gamma \mathcal{G}$ such that $\mathfrak{q}_\delta = \mathfrak{r}^*_\delta$. This in turn implies that $\mathfrak{p}$ is a $\mathcal{F}$-preantecedent and $\mathfrak{q} \vdash \overline{\alpha}[a] \in \mathcal{G}$ such that $\mathfrak{p}_\gamma = \mathfrak{q}^*_\gamma$. From these observations it should now be clear that $\mathfrak{q};_\delta \mathfrak{r} \vdash \overline{\alpha}[a] \in \mathcal{G} ;_\delta \mathcal{H}$ and therefore,

$$\mathfrak{p};_\gamma (\mathfrak{q};_\delta \mathfrak{r}) \vdash \overline{\alpha}[a] \in \mathcal{F} ;_\gamma (\mathcal{G} ;_\delta \mathcal{H})$$

This argument may also be used to show that any entailment in $\mathcal{F} ;_\gamma (\mathcal{G} ;_\delta \mathcal{H})$ is an entailment in $(\mathcal{F} ;_\gamma \mathcal{G}) ;_\delta \mathcal{H}$. Thus, we may conclude that both inclusions hold and composition of extensional processes is associative. ∎

With some small changes the proof of associativity may be used to prove that composition satisfies the interchange law and so we conclude:

Proposition 5.3.8 *Composition of extensional processes satisfies the interchange law.*

All the requirements for polycategories are now satisfied proving:

Theorem 5.3.9 *Protocols and extensional processes form a polycategory.*

The polycategory of protocols and extensional process built over an arbitrary polycategory $\mathbf{A}$ will be denoted by $\mathcal{EP}_{\mathbf{A}}$.

5.3.2 Poly-sums and poly-products

The purpose of this section is to show that the polycategory $\mathcal{EP}_{\mathbf{A}}$ has sums and products. We begin by defining sums and products of extensional processes.

Let $\mathcal{F}_i : \Gamma, \gamma : X_i \to \Delta$, for $i \in I$, be extensional processes. The **sum** (or **coproduct**) on γ of the $\mathcal{F}_i$'s, denoted $\gamma\langle\mathcal{F}_i\rangle_{i\in I} : \Gamma, \gamma : \sum_{i\in I} X_i \to \Delta$, is defined to be the set $\{\gamma[i] * \mathcal{F}_i \mid i \in I\}$. For example, if $\mathfrak{p} \vdash \times \in \mathcal{F}_k$ then $\gamma[k] * \mathfrak{p} \vdash \times \in \gamma\langle\mathcal{F}_i\rangle_{i\in I}$.

The **product** is constructed dually, i.e., if $\mathcal{G}_j : \Gamma \to \delta : Y_j, \Delta$, for $j \in J$, are extensional processes, then the product of the $\mathcal{G}_j$'s on δ, denoted $\delta\langle\mathcal{G}_j\rangle_{j\in J} : \Gamma \to \delta : \prod_{j\in J} Y_j, \Delta$, is defined to be the set $\{\delta[i] * \mathcal{G}_j \mid j \in J\}$.

The extensional process for the k^{th} injection on γ is the set $\mathcal{B}_k = \overline{\gamma}[k] * 1_{X_k}$ where γ is the codomain channel of 1_{X_k}. The extensional process for the k^{th} projection on γ is $\mathcal{P}_k = \overline{\gamma}[k] * 1_{X_k}$ (the same set as the injection), where γ is the domain channel of 1_{X_k}.

We will now work with the sum; the dual properties hold for the product. Note that it still must be shown that what we have defined as the "sum" is actually a sum. First, however, it is necessary to show that the "sum" satisfies the requirements of an extensional process.

Proposition 5.3.10 *$\gamma\langle\mathcal{F}_i\rangle_{i\in I}$ is an extensional process.*

Proof. We must show that $\gamma\langle\mathcal{F}_i\rangle_{i\in I}$ satisfies EP-1 $\sim$ EP-7.

EP-1 Suppose that $\gamma[k] * \mathfrak{p}$, for some $k \in I$, is a $\gamma\langle\mathcal{F}_i\rangle_{i\in I}$-antecedent and $\overline{\alpha}[a]$ is an output event in $\gamma[k] * \mathfrak{p}$. Then $\overline{\alpha}[a]$ must be justified in $\mathcal{F}_k$, say by $\mathfrak{u}$, so that $\gamma[k] * \mathfrak{u}$ justifies it in $\gamma\langle\mathcal{F}_i\rangle_{i\in I}$. Thus, all $\gamma\langle\mathcal{F}_i\rangle_{i\in I}$-antecedents are $\gamma\langle\mathcal{F}_i\rangle_{i\in I}$-justified.

EP-2 Suppose that $\gamma[k] * \mathfrak{p} \vdash \overline{\alpha}[a],\ \gamma[l] * \mathfrak{q} \vdash \overline{\alpha}[b] \in \gamma\langle\mathcal{F}_i\rangle_{i\in I}$, for some $k, l \in I$, with $(\gamma[k] * \mathfrak{p})_\alpha = (\gamma[l] * \mathfrak{q})_\alpha$ and $\gamma[k] * \mathfrak{p} \smile \gamma[l] * \mathfrak{q}$. Since they are compatible, $k = l$, and therefore, $\mathfrak{p} \vdash \overline{\alpha}[a]$, $\mathfrak{q} \vdash \overline{\alpha}[b] \in \mathcal{F}_k$ with $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} \smile \mathfrak{q}$, and as $\mathcal{F}_k$ satisfies EP-2, $a = b$.

EP-3 Let $\mathfrak{q}$ be a saturated $\gamma\langle\mathcal{F}_i\rangle_{i\in I}$-preantecedent. It must be of the form $\gamma[k] * \mathfrak{q}'$, for some $k \in I$, where $\mathfrak{q}'$ is a saturated $\mathcal{F}_k$-preantecedent. By EP-3, $\mathfrak{q}' \vdash \times \in \mathcal{F}_k$, and thus, $\gamma[k] * \mathfrak{q} \vdash \times \in \gamma\langle\mathcal{F}_i\rangle_{i\in I}$ satisfying EP-3.

EP-4 Suppose that $\gamma[k] * \mathfrak{p} \vdash \overline{\alpha}[a],\ \gamma[l] * \mathfrak{q} \vdash \overline{\beta}[b] \in \gamma\langle\mathcal{F}_i\rangle_{i\in I}$, for some $k, l \in I$, with $(\gamma[k] * \mathfrak{p})_\alpha = (\gamma[l] * \mathfrak{q})_\alpha$ and $\gamma[k] * \mathfrak{p} \sqsubseteq \gamma[l] * \mathfrak{q}$. It is clear then that $k = l$, and therefore, $\mathfrak{p} \vdash \overline{\alpha}[a]$, $\mathfrak{q} \vdash \overline{\beta}[b] \in \mathcal{F}_k$ with $\mathfrak{p}_\alpha = \mathfrak{q}_\alpha$ and $\mathfrak{p} \sqsubseteq \mathfrak{q}$. As $\mathcal{F}_k$ satisfies EP-4, $\mathfrak{q} * \overline{\alpha}[a] \vdash \overline{\beta}[a] \in \mathcal{F}_k$ and thus, $\gamma[k] * \mathfrak{q} * \overline{\alpha}[a] \vdash \overline{\beta}[b] \in \gamma\langle\mathcal{F}_i\rangle_{i\in I}$.

EP-5 Let $\gamma[k] * \mathfrak{p} \vdash \overline{\alpha}[a] \in \gamma\langle \mathcal{F}_i \rangle_{i \in I}$, for some $k \in I$. If $\beta[b]$ is a legal input event for this entailment, then it is certainly a legal input event for $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{F}_k$. Thus, $\mathfrak{p} * \beta[b] \vdash \overline{\alpha}[a] \in \mathcal{F}_k$ and so, $\gamma[k] * \mathfrak{p} * \beta[b] \vdash \overline{\alpha}[a] \in \gamma\langle \mathcal{F}_i \rangle_{i \in I}$.

EP-6 Let $\gamma[k] * \mathfrak{p} * \overline{\beta}[b] \vdash \overline{\alpha}[a] \in \gamma\langle \mathcal{F}_i \rangle_{i \in I}$, for some $k \in I$. Then, $\mathfrak{p} * \overline{\beta}[b] \vdash \overline{\alpha}[a] \in \mathcal{F}_k$, and as $\mathcal{F}_k$ satisfies EP-6, $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{F}_k$, and thus, $\gamma[k] * \mathfrak{p} \vdash \overline{\alpha}[a] \in \gamma\langle \mathcal{F}_i \rangle_{i \in I}$.

EP-7 Suppose $\{\gamma[k] * \mathfrak{p} * \beta[b_i] \vdash \overline{\alpha}[a] \mid i \in I\} \in \gamma\langle \mathcal{F}_i \rangle_{i \in I}$, for some $k \in I$, where $\{b_i \mid i \in I\}$ is the set of all possible input events on β given $\gamma[k] * \mathfrak{p}$. This implies that $\{\mathfrak{p} * \beta[b_i] \vdash \overline{\alpha}[a] \mid i \in I\} \in \mathcal{F}_k$, and as $\mathcal{F}_k$ satisfies EP-7, $\mathfrak{p} \vdash \overline{\alpha}[a] \in \mathcal{F}_k$, and therefore, $\gamma[k] * \mathfrak{p} \vdash \overline{\alpha}[a] \in \gamma\langle \mathcal{F}_i \rangle_{i \in I}$. ∎

We now show that $\gamma\langle \mathcal{F}_i \rangle_{i \in I}$ is a coproduct of extensional processes.

Proposition 5.3.11 *Let $\mathcal{F} : \Gamma, \gamma : X_i \to \Delta$, for $i \in I$, be extensional processes. Then $\gamma\langle \mathcal{F}_i \rangle_{i \in I} : \Gamma, \gamma : \sum_{i \in I} X_i \to \Delta$ is the poly-coproduct of the $\mathcal{F}_i$'s, $i \in I$.*

Proof. We must show that the following four properties hold:

(i) $\gamma\langle \mathcal{B}_i \rangle_{i \in I} = 1_{\sum_{i \in I} X_i}$

(ii) $\mathcal{B}_k \;;_\gamma \gamma\langle \mathcal{F}_i \rangle_{i \in I} = \mathcal{F}_k$

(iii) $\gamma\langle \mathcal{F}_i \rangle_{i \in I} \;;_\alpha \mathcal{H} = \gamma\langle \mathcal{F}_i \;;_\alpha \mathcal{H} \rangle_{i \in I}$, where $\gamma \neq \alpha$

(iv) $\mathcal{H} \;;_\alpha \gamma\langle \mathcal{F}_i \rangle_{i \in I} = \gamma\langle \mathcal{H} \;;_\alpha \mathcal{F}_i \rangle_{i \in I}$, where $\alpha \neq \gamma$

We take each one in turn.

(i) The goal will be to show that for any $\mathcal{F}_i : \Gamma, \gamma : X_i \to \Delta$, where $i \in I$, and for $\mathcal{B}_k : \alpha : X_k \to \gamma : \sum_{i \in I} X_i$

$$\alpha\langle \mathcal{B}_i \rangle_{i \in I} \;;_\gamma \gamma\langle \mathcal{F}_i \rangle_{i \in I} = \alpha\langle \mathcal{F}_i \rangle_{i \in I}$$

which shows that $\alpha\langle \mathcal{B}_i \rangle_{i \in I}$ acts as the identity on $\sum_{i \in I} X_i$. Consider an entailment in $\mathfrak{q} \vdash \times \in \alpha\langle \mathcal{B}_i \rangle_{i \in I} \;;_\gamma \gamma\langle \mathcal{F}_i \rangle_{i \in I}$. There are three cases to consider depending on whether $\times$ is atomic, an output event on α, or an output event

on a channel other than α. However, these cases are subsumed if, in the following, we simply consider $\mathfrak{r}$ to be a 1_{X_k}-preantecedent and $\mathfrak{p}$ to be a $\mathcal{F}_k$-preantecedent, such that one is an antecedent (it does not matter which). Then, by the definition of $\mathcal{B}_k$, composition, and coproducts, $\mathfrak{q} \vdash \times$ must be of the form:

$$\begin{aligned}\mathfrak{q} \vdash \times &= (\alpha[k] * \overline{\gamma}[k] * \mathfrak{r});_\gamma (\gamma[k] * \mathfrak{p}) \vdash \times \\ &= (\alpha[k] * \mathfrak{r});_\gamma \mathfrak{p} \vdash \times \\ &= \alpha[k] * (\mathfrak{r};_\gamma \mathfrak{p}) \vdash \times\end{aligned}$$

By the definition of the identity extensional process, $\mathfrak{r};_\gamma \mathfrak{p} \in \mathcal{F}_k$, and so $\alpha[k] * (\mathfrak{r};_\gamma \mathfrak{p}) \vdash \times \in \gamma\langle\mathcal{F}_i\rangle_{i\in I}$ (up to renaming α to γ). Therefore, $\alpha\langle\mathcal{B}_i\rangle_{i\in I};_\gamma \gamma\langle\mathcal{F}_i\rangle_{i\in I} \subseteq \gamma\langle\mathcal{F}_i\rangle_{i\in I}$, and, by Proposition 5.1.14, $\alpha\langle\mathcal{B}_i\rangle_{i\in I};_\gamma \gamma\langle\mathcal{F}_i\rangle_{i\in I} = \gamma\langle\mathcal{F}_i\rangle_{i\in I}$. Thus, $\alpha\langle\mathcal{B}_i\rangle_{i\in I}$ is the identity on $\sum_{i\in I} X_i$.

(ii) Let $\mathfrak{q} \vdash \times \in \mathcal{B}_k;_\gamma \gamma\langle\mathcal{F}_i\rangle_{i\in I}$. Again, in the following, assume that $\mathfrak{r}$ is a 1_{X_k}-preantecedent and that $\mathfrak{p}$ is a $\mathcal{F}_k$-preantecedent. By the definition of $\mathcal{B}_k$, composition, and coproducts, $\mathfrak{q} = (\overline{\gamma}[k] * \mathfrak{r});_\gamma (\gamma[k] * \mathfrak{p}) = \mathfrak{r};_\gamma \mathfrak{p}$, and hence, $\mathfrak{q} \vdash \times \in \mathcal{F}_k$ (up to renaming of channels). Therefore, by Proposition 5.1.14, $\mathcal{B}_k;_\gamma \gamma\langle\mathcal{F}_i\rangle_{i\in I} = \mathcal{F}_k$.

(iii) Given a $\gamma\langle\mathcal{F}_i\rangle_{i\in I}$-antecedent $\gamma[k] * \mathfrak{p}$, for some $k \in I$, and a $\mathcal{H}$-preantecedent $\mathfrak{q}$, the following equality holds

$$(\gamma[k] * \mathfrak{p});_\alpha \mathfrak{q} = \gamma[k] * (\mathfrak{p};_\alpha \mathfrak{q}) \qquad (\text{when } \alpha \neq \gamma)$$

so that any entailment $(\gamma[k]*\mathfrak{p});_\alpha \mathfrak{q} \vdash \times \in \gamma\langle\mathcal{F}_i\rangle_{i\in I};_\alpha \mathcal{H}$, may be written as $\gamma[k]*(\mathfrak{p};_\alpha \mathfrak{q}) \vdash \times \in \gamma\langle\mathcal{F}_i;_\alpha \mathcal{H}\rangle_{i\in I}$ and vise versa, which shows that $\gamma\langle\mathcal{F}_i\rangle_{i\in I};_\alpha \mathcal{H} = \gamma\langle\mathcal{F}_i;_\alpha \mathcal{H}\rangle_{i\in I}$.

(iv) That $\mathcal{H};_\alpha \gamma\langle\mathcal{F}_i\rangle_{i\in I} = \gamma\langle\mathcal{H};_\alpha \mathcal{F}_i\rangle_{i\in I}$ is true follows from the equality

$$\mathfrak{q};_\alpha (\gamma[k] * \mathfrak{p}) = \gamma[k] * (\mathfrak{q};_\alpha \mathfrak{p}) \qquad (\text{when } \alpha \neq \gamma)$$

and an argument similar to (iii) above. ∎

With duality this now proves:

Proposition 5.3.12 *$\mathcal{EP}_{\mathbf{A}}$ has finite poly-sums and poly-products.*

5.3.3 Extensional processes are soft

The goal of this section is to prove the following theorem:

Theorem 5.3.13 *$\mathcal{EP}_{\mathbf{A}}$ is the free polycategory generated from $\mathbf{A}$ under (finite) sums and products.*

Notice that the inclusion functor $\mathbf{A} \xrightarrow{\mathcal{I}} \mathcal{EP}_{\mathbf{A}}$ is a full inclusion and the objects of $\mathcal{EP}_{\mathbf{A}}$ are the objects of $\Sigma\Pi_{\mathbf{A}}$ (i.e., they are generated under sums and products from the objects of $\mathbf{A}$). Thus, if it can be shown that the inclusion $\mathcal{I}$ is soft, the Whitman Theorem, Theorem 4.2.10, may be used to show that $\mathcal{EP}_{\mathbf{A}}$ is equivalent to $\Sigma\Pi_{\mathbf{A}}$, and hence, is the free polycategory generated from $\mathbf{A}$ under finite sums and products. Thus:

Proposition 5.3.14 *The inclusion morphism of polycategories $\mathbf{A} \xrightarrow{\mathcal{I}} \mathcal{EP}_{\mathbf{A}}$ is soft.*

Recall that in order to show that $\mathcal{I}$ is soft, we need to show that $\mathcal{I}$ is a semi-soft extension and a soft extension. That is, that the map

$$\sum_{(\alpha,i)\in\mathfrak{I}(\Gamma;\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{(\alpha,i)} \xrightarrow{\{\overline{\alpha}[i] * \mathcal{F}\}_{(\alpha,i)}} \mathrm{Hom}(\Gamma;\Delta)$$

is a surjection, and that

$$\begin{array}{ccc} \displaystyle\sum_{((\alpha,i),(\beta,j))\in\mathfrak{I}(\Gamma,\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{((\alpha,i),(\beta,j))} & & \\ \Big\downarrow{\scriptstyle \sum_{(\alpha,i)} \{\overline{\beta}[j]*\mathcal{F}\}_{((\alpha,i),(\beta,j))}} & \Big\downarrow{\scriptstyle \sum_{(\beta,j)} \{\overline{\alpha}[i]*\mathcal{F}\}_{((\alpha,i),(\beta,j))}} & \\ \displaystyle\sum_{(\gamma,k)\in\mathfrak{I}(\Gamma,\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{(\gamma,k)} & & \\ \Big\downarrow{\scriptstyle \{\overline{\gamma}[k]*\mathcal{F}'\}_{(\gamma,k)}} & & \\ \mathrm{Hom}(\Gamma;\Delta) & & \end{array}$$

is a coequalizer diagram.

Proof. To see that $\mathcal{I}$ is a semi-soft extension consider the output poly-hom-set $\mathrm{Hom}(\Gamma;\Delta)$. Any behaviour in $\mathrm{Hom}(\Gamma;\Delta)$ started out saturated and therefore, must be of the form $\overline{\alpha}[a]*\mathcal{F}$ where $\mathcal{F}\in\sum_{(\alpha,i)}[\mathrm{Hom}(\Gamma;\Delta)]_{(\alpha,i)}$. Thus, the map

$$\sum_{(\alpha,i)\in\mathfrak{I}(\Gamma;\Delta)} [\mathrm{Hom}(\Gamma;\Delta)]_{(\alpha,i)} \xrightarrow{\{} \overline{\alpha}[i]*\mathcal{F}\}_{(\alpha,i)}\mathrm{Hom}(\Gamma;\Delta)$$

is a surjection and $\mathcal{I}$ is a semi-soft extension.

That $\mathcal{I}$ is soft follows follows from an argument similar to the syntactic case. Explicitly, suppose the diagram has a coequalizer (q,Q) and let $h:Q\to\mathrm{Hom}(\Gamma;\Delta)$ be the unique map such that $q;h=\{\overline{\gamma}[k]*\mathcal{F}_{(\gamma,k)}\}$. But $\{\overline{\gamma}[k]*\mathcal{F}_{(\gamma,k)}\}$ is a surjection, and therefore, so is h.

Now consider the two following equivalent extensional processes $\overline{\alpha}[i]*\mathcal{F}$ and $\overline{\beta}[j]*\mathcal{G}$ in $\mathrm{Hom}(\Gamma;\Delta)$. That they are equivalent means that they must be of the form

$$\overline{\alpha}[i]*(\overline{\beta}[j]*\mathcal{F}') \qquad \text{and} \qquad \overline{\beta}[j]*(\overline{\alpha}[i]*\mathcal{G}')$$

respectively. However, this implies that the extensional processes $\mathcal{F}'$ and $\mathcal{G}'$ must be equivalent in $\sum_{((\alpha,i),(\beta,j))}[\mathrm{Hom}(\Gamma;\Delta)]_{((\alpha,i),(\beta,j))}$, and therefore, must be coequalized in Q establishing that h is an injection.

Therefore, the map h is a bijection and $Q \cong \mathrm{Hom}(\Gamma; \Delta)$ establishing that

$$\left(\{\overline{\gamma}[k] * \mathcal{F}_{(\gamma,k)}\}_{(\gamma,k)},\ \mathrm{Hom}(\Gamma; \Delta)\right)$$

is the coequalizer. ∎

5.3.4 The additive units

In this section we explain how the additive units are handled in this system as they are handled in a somewhat subtle manner. (Indeed, when Dominique Hughes first asked about the units, and we tried to explain it to him, we instantly got lost in the subtleties.)

Recall that we are using the notation $\sum_{\emptyset} = 0$ and $\prod_{\emptyset} = 1$ to denote the initial and final objects respectively. In what follows it will be shown that any map in which the domain contains the initial object and the codomain contains the final object is the empty set of entailments.

Notice that if, for any map, the domain contains the initial object or the codomain contains the final object, i.e.,

$$\Gamma, 0 \vdash \Delta \qquad \text{or} \qquad \Gamma \vdash 1, \Delta$$

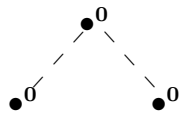
then the frontier of the empty behaviour has a source role (as the initial object in the domain and the final object in the codomain have a source role), and so cannot be the antecedent of any entailment. Thus, the empty set of entailments results. Recall that in this situation the empty set does constitute an extensional process (Proposition 5.2.2). Moreover, notice that if we were to compose an extensional process with either

$$\Gamma, 0 \vdash \Delta \qquad \text{or} \qquad \Gamma \vdash 1, \Delta$$

the resulting frontier would also contain a protocol with a source role and no entailments could result.

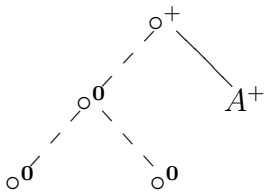
The roles are also used to determine where a given protocol is an initial or a final object. That is, the initial (resp. final) object may be "hidden" within some

structure and the roles are used to determine this. For example, $0 \cong 0 + 0$ is the initial object, in the domain, its role is calculated as



Similarly for $1 \cong 1 \times 1$ in the codomain or any other "hidden" initial or final objects.

Consider a protocol in the domain of the form



which contains an initial object and so is really just the protocol A. As we can see from the tree, this is indeed the case as the only legal transition is the one which leads to A.

In the free polycategory with sums and products one does not expect a map $1 \to 0$. This map is prevented from occurring by the rule EP-3. As both 1 in the domain and 0 in the codomain are "output" protocols the empty behaviour is saturated in $1 \to 0$. As the empty behaviour is both a preantecedent and a saturated preantecedent it fits the premise of EP-3. However, it is clear that the empty behaviour cannot be made into an entailment as there is nothing to output, and so, this "set" of entailments is not an extensional process.

Chapter 6

Conclusion and Further Directions

In this thesis we began by introducing a logic $\Sigma\Pi$, and constructing a (syntactic) polycategory of protocols and processes from this logic. This polycategory was shown to be the free polycategory with finite sums and finite products. To characterize polycategories of this type we chose to use Joyal's notion of softness, and so it was necessary to extend this notion to the polycategorical case. Following the lead of Joyal and Cockett and Seely, we prove a "Whitman theorem" which gives the characterization. Next, it is shown how proofs in our logic can be interpreted as concurrent channel-based processes by providing a process semantics. These process are organized into a polycategory of protocols and extensional processes. Using softness, we show that this polycategory is equivalent to the syntactic polycategory. This then establishes that every extensional process is the denotation of a unique cut-free proof in $\Sigma\Pi$, proving that this model is full and faithfully complete.

Further directions The next obvious step is to add the multiplicative connectives of linear logic into our interpretation. We have a fairly detailed idea of how this can be accomplished, but it still needs to be fully formalized. Additionally, from the viewpoint of process semantics, one would also like to model infinite processes. This leads into fixed point logics and circular proofs [San02a] where there is considerable gaps to be filled.

Bibliography

[AJ94] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *Journal of Symbolic Logic*, 59(2):543–574, 1994.

[AJM00] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163, 2000.

[AM98a] S. Abramsky and G. McCusker. Call-by-value games. In *Proceedings of CSL '97*, number 1414 in Lecture Notes in Computer Science, pages 1–17. Springer-Verlag, 1998.

[AM98b] S. Abramsky and G. McCusker. Game semantics. In H. Schwichtenberg and U. Berger, editors, *Logic and Computation: Proceedings of the 1997 Marktoberdorf Summer School*, 1998.

[AM99] S. Abramsky and P.-A. Melliès. Concurrent games and full completeness. In *Proc. IEEE Symposium of Logic in Computer Science*, pages 431–442. IEEE Computer Society Press, 1999.

[BHS03] R. Blute, M. Hamano, and P. J. Scott. Softness of hypercoherences and MALL full completeness. *Submitted*, 2003. Preprint available at: `<http://www.site.uottawa.ca/∼phil/>`.

[Bla92] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:183–220, 1992.

[BW99] M. Barr and C. Wells. *Category theory for computer science*. Les Publications CRM, third edition, 1999.

[Che02] E. Cheng. Weak n-categories: opetopic and multitopic foundations. October 2002. Submitted to the *Journal of Pure and Applied Algebra*. Also at <http://www.dpmms.cam.ac.uk/∼elgc2/>.

[CKS03] J.R.B. Cockett, J. Koslowski, and R.A.G. Seely. Morphisms and modules for poly-bicategories. *Theory and Applications of Categories*, 11(2):15–74, 2003.

[CS97] J.R.B. Cockett and R.A.G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997. Corrected version available at <http://www.math.mcgill.ca/rags/>.

[CS01] J.R.B. Cockett and R.A.G. Seely. Finite sum-product logic. *Theory and Applications of Categories*, 8(5):63–99, 2001.

[DH00] V. Danos and R. Harmer. Probabilistic game semantics. In *Proc. IEEE Symposium of Logic in Computer Science, Santa Barbara, June 2000*. IEEE Computer Society Press, 2000.

[DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[GTL89] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types*. Number 7 in Cambridge Tracts in Theoetical Computer Science. Cambridge University Press, 1989. Available at <http://www.dcs.qmw.ac.uk/∼pt/stable/Proofs+Types.html>.

[HM99] R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *Proc. IEEE Symposium of Logic in Computer Science*. IEEE Computer Society Press, 1999.

[HO00] J.M.E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163:285–408, 2000.

[Joy95a] A. Joyal. Free bicomplete categories. *Math. Reports XVII, Acad. Sci. Canada*, pages 219–224, 1995.

[Joy95b] A. Joyal. Free bicompletion of enriched categories. *Math. Reports XVII, Acad. Sci. Canada*, pages 213–218, 1995.

[Joy95c] A. Joyal. Free lattices, communication, and money games. In *Proc. of the 10th International Congress of Logic, Methodology, and Philosophy of Science*, Firenze, 1995.

[Lai98] J. Laird. *A semantic analysis of control*. PhD thesis, University of Edinburgh, 1998.

[Lam69] J. Lambek. Deductive systems and categories II. In *Category theory, homology theory and their applications I (1968)*, volume 86, pages 76–122. LNM, Springer-Verlag, Berlin, Heidelberg and New York, 1969.

[LS86] J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*. Cambridge University Press, 1986.

[McC96] G. McCusker. Games and full abstraction for FPC. In *Proc. IEEE Symposium on Logic in Computer Science*, pages 174–183. IEEE Computer Society Press, 1996.

[ML98] S. Mac Lane. *Categories for the working mathematician*. Springer GTM 5, second edition, 1998.

[Nic96] H. Nickau. *Hereditarily sequential functionals: a game-theoretic approach to sequentiality*. PhD thesis, Universität-Gesamthochschule-Siegen, 1996.

[San02a] L. Santocanale. A calculus of circular proofs and its categorical semantics. In *Foundations of Software Science and Computation Structures 02*, volume 2303 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[San02b] L. Santocanale. Free μ-lattices. *Journal of Pure and Applied Algebra*, 169:227–264, March 2002.

[Sco93] D. S. Scott. A type-theoretical alternative to CUCH, ISWIM and OWHY. *Theoretical Computer Science*, 121:411–440, 1993.

[Whi41] P. M. Whitman. Free lattices. *Ann. of Math*, 42:325–330, 1941.

Appendix A

Resolving Critical Pairs

The following notation will be used to reduce the number of cases of certain rewrites. Let $\delta(g)$ denote any of the following morphism,

$$\delta\{a_i \mapsto g_i\}, \quad \delta(a_i \mapsto g_i), \quad \overleftarrow{\delta}[a](g), \quad \overrightarrow{\delta}[a](g)$$

Then, $f ;_\gamma \delta(g) \Longrightarrow \delta(f ;_\gamma g)$ denotes

$$f ;_\gamma \delta\{a_i \mapsto g_i\} \Longrightarrow \delta\{a_i \mapsto f ;_\gamma g_i\}, \qquad f ;_\gamma \overleftarrow{\delta}[a](g) \Longrightarrow \overleftarrow{\delta}[a](f ;_\gamma g),$$
$$f ;_\gamma \delta(a_i \mapsto g_i) \Longrightarrow \delta(a_i \mapsto f ;_\gamma g_i), \qquad f ;_\gamma \overrightarrow{\delta}[a](g) \Longrightarrow \overrightarrow{\delta}[a](f ;_\gamma g)$$

and similarly for $\delta(f) ;_\gamma g \Longrightarrow \delta(f ;_\gamma g)$.

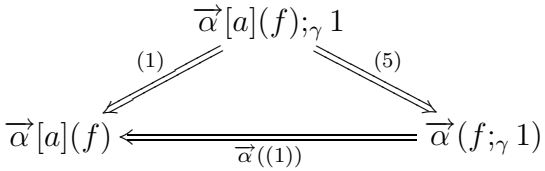
The resolutions are as follows. The dual rewrite, if there is one, is indicated in $[-]$.

(1)-(2) obvious.

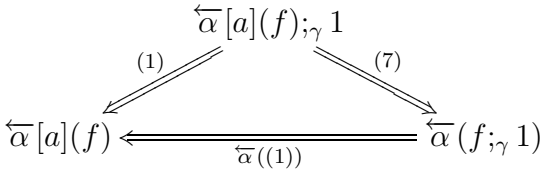
(1)-(3) **[(2)-(4)]**

$$\begin{array}{ccc} & \alpha\{a_i \mapsto f_i\}_i ;_\gamma 1 & \\ {\scriptstyle (1)}\swarrow & & \searrow{\scriptstyle (3)} \\ \alpha\{a_i \mapsto f_i\}_i & \xleftarrow[\alpha\{(1)\}]{} & \alpha\{a_i \mapsto f_i ;_\gamma 1\}_i \end{array}$$

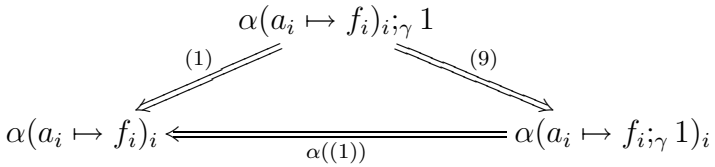
(1)-(5) [(2)-(6)]



(1)-(7) [(2)-(8)]

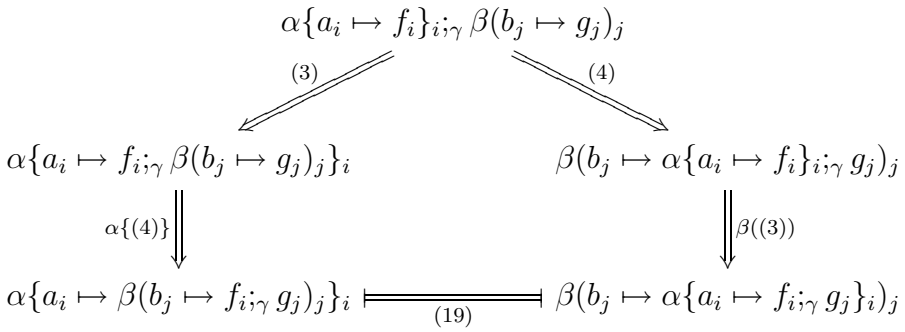


(1)-(9) [(2)-(10)]

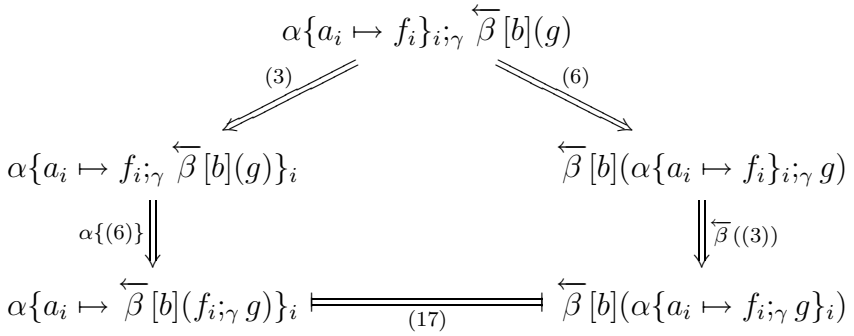


This handles all critical pairs involving (1) and (2). We now look at any critical pairs involving (3) and (4).

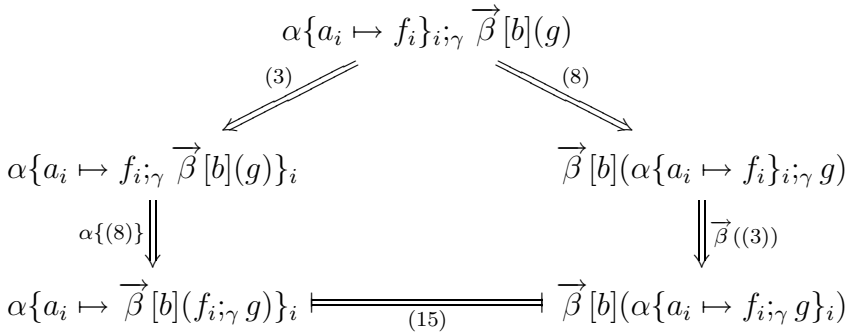
(3)-(4)



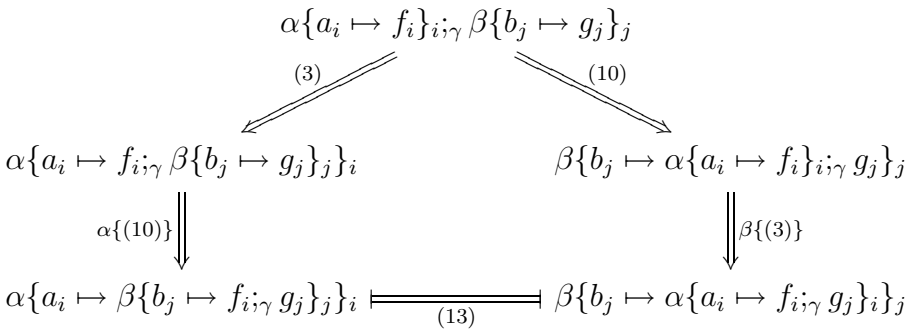
(3)-(6) [(4)-(5)]



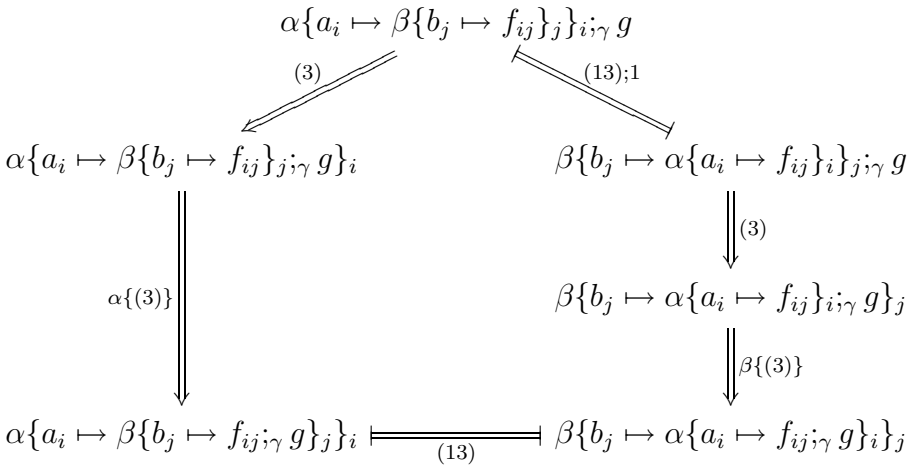
(3)-(8) [(4)-(7)]



(3)-(10) [(4)-(9)]



(3)-(13) [(4)-(14)]

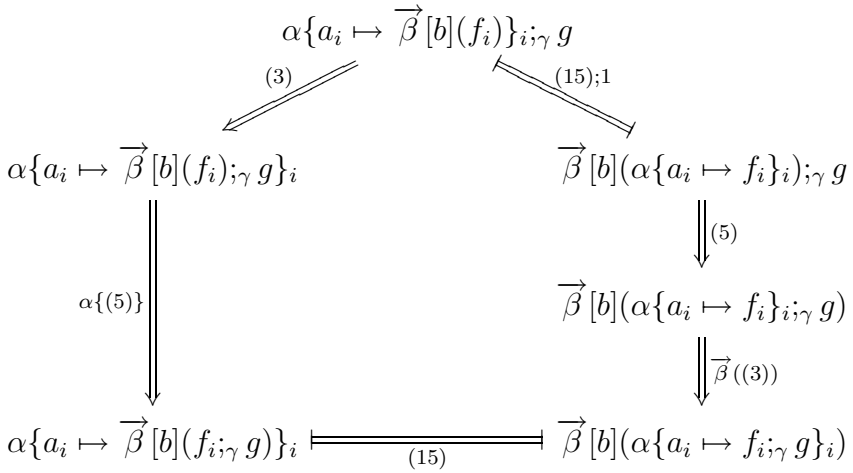


(3)-(15) [(4)-(16)]

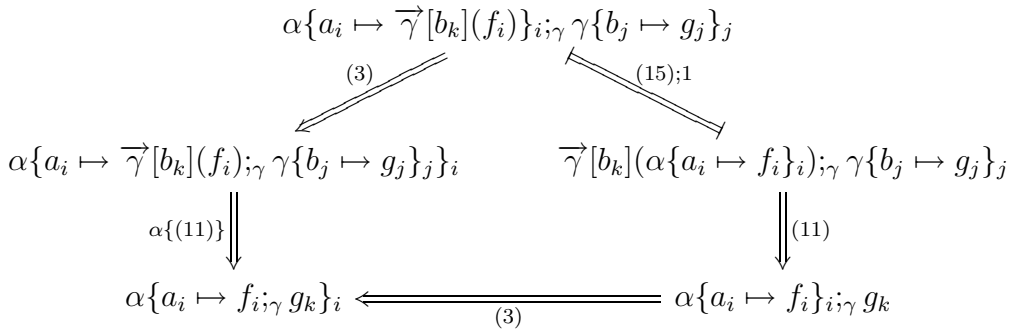
$$\alpha\{a_i \mapsto \overrightarrow{\beta}\,[b_k](f_i);_\gamma g\}_i \Longleftarrow \alpha\{a_i \mapsto \overrightarrow{\beta}\,[b_k](f_i)\}_i;_\gamma g \Longmapsto \overrightarrow{\beta}\,[b_k](\alpha\{a_i \mapsto f_i\}_i);_\gamma g$$

There are two subcases to consider.

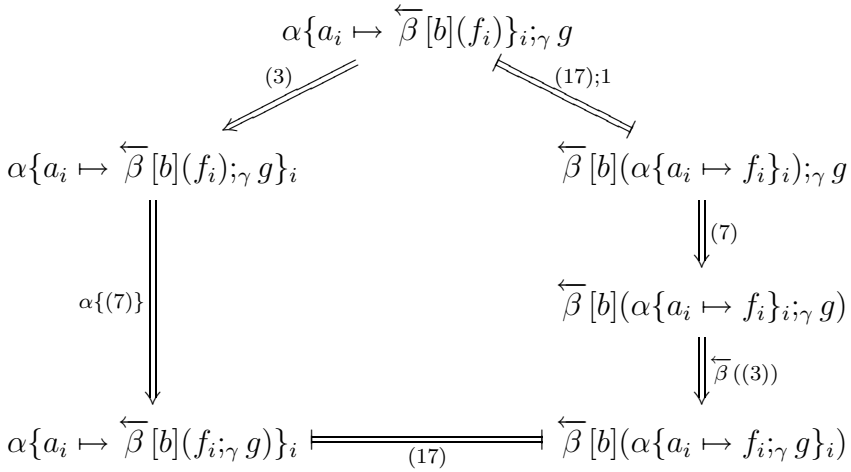
(a) $\beta \neq \gamma$.



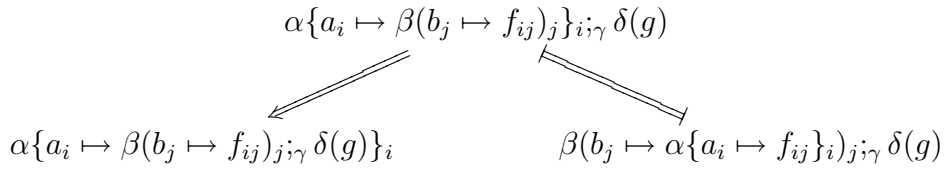
(b) $\beta = \gamma$. Here we may assume that $g = \gamma\{b_j \mapsto g_j\}_j$.



(3)-(17) [(4)-(18)]

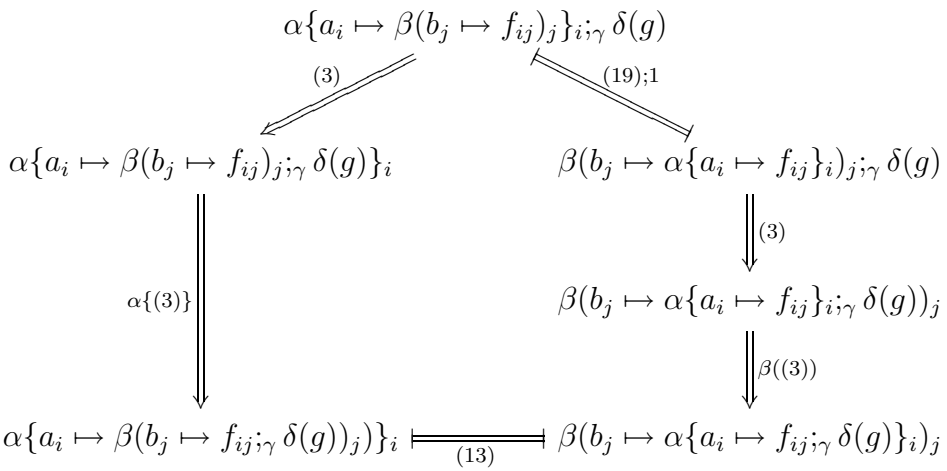


(3)-(19) [(4)-(19)]

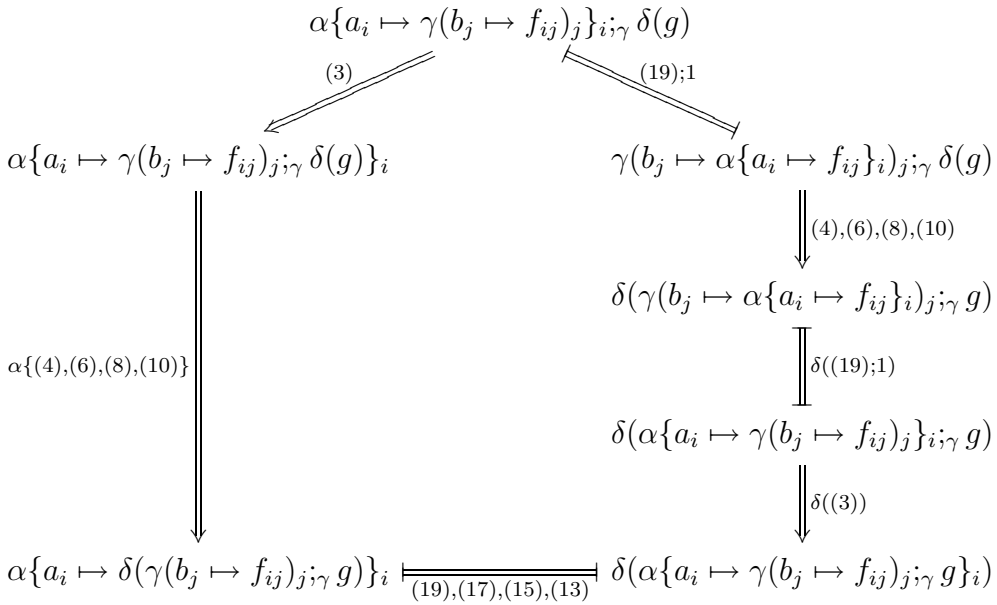


There are three subcases to consider.

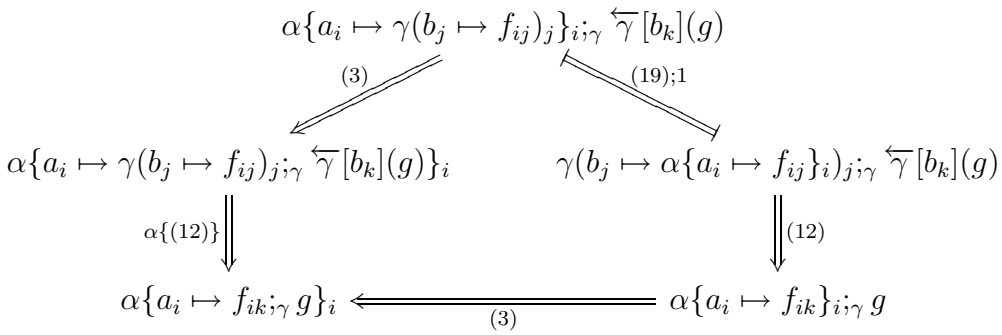
(a) $\beta \neq \gamma$



(b) $\beta = \gamma$ and $\delta \neq \gamma$

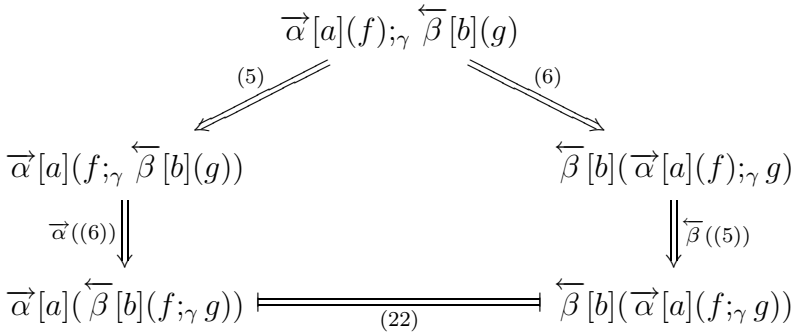


(c) $\beta = \gamma$ and $\delta(g) = \overleftarrow{\gamma}[b_k](g)$

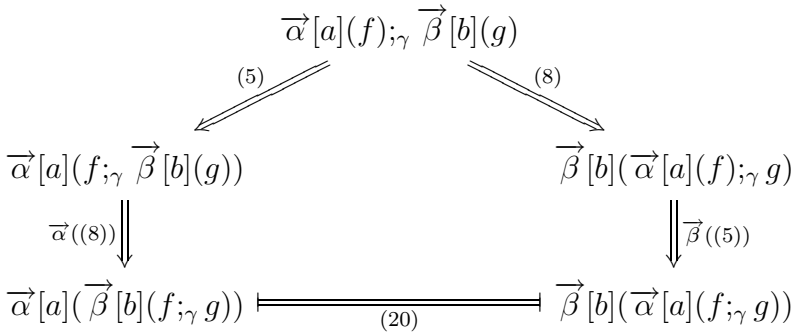


This handles all critical pairs involving (3) and (4). We now look at any critical pairs involving (5) and (6).

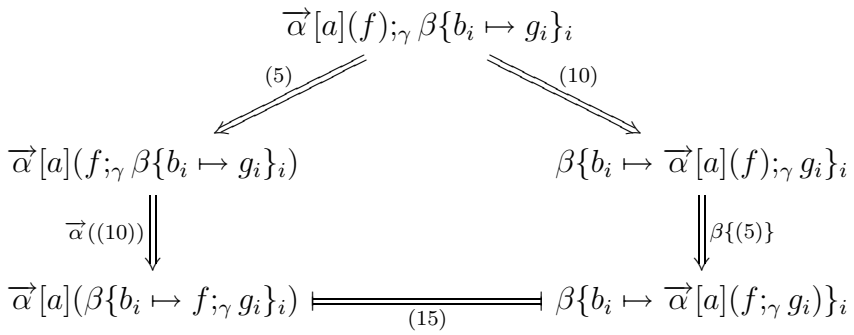
(5)-(6)



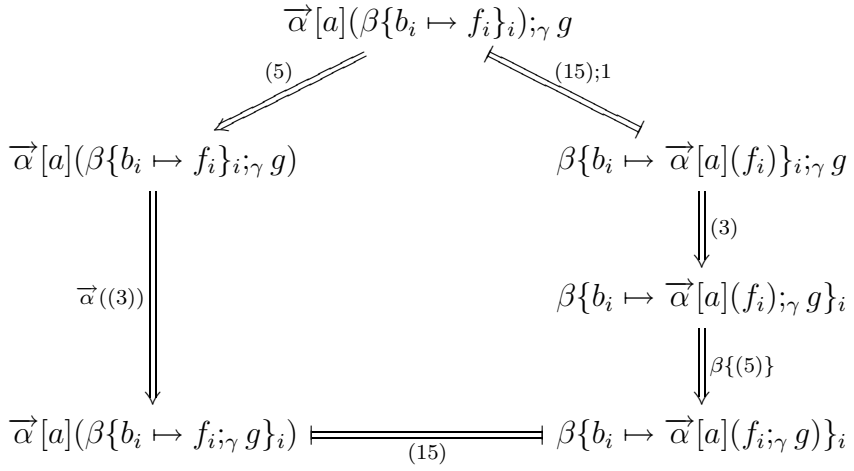
(5)-(8) [(6)-(7)]



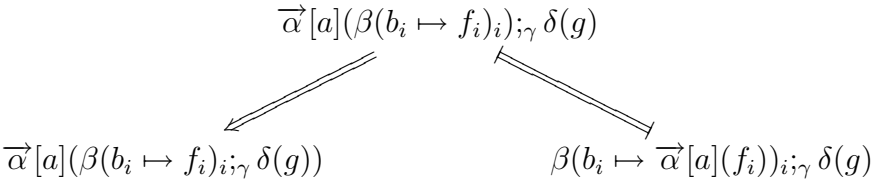
(5)-(10) [(6)-(9)]



(5)-(15) [(6)-(16)]

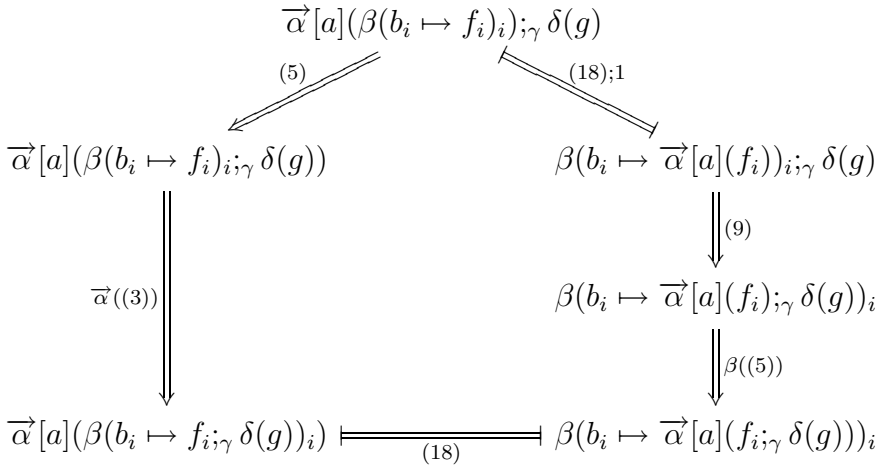


(5)-(18) [(6)-(17)]

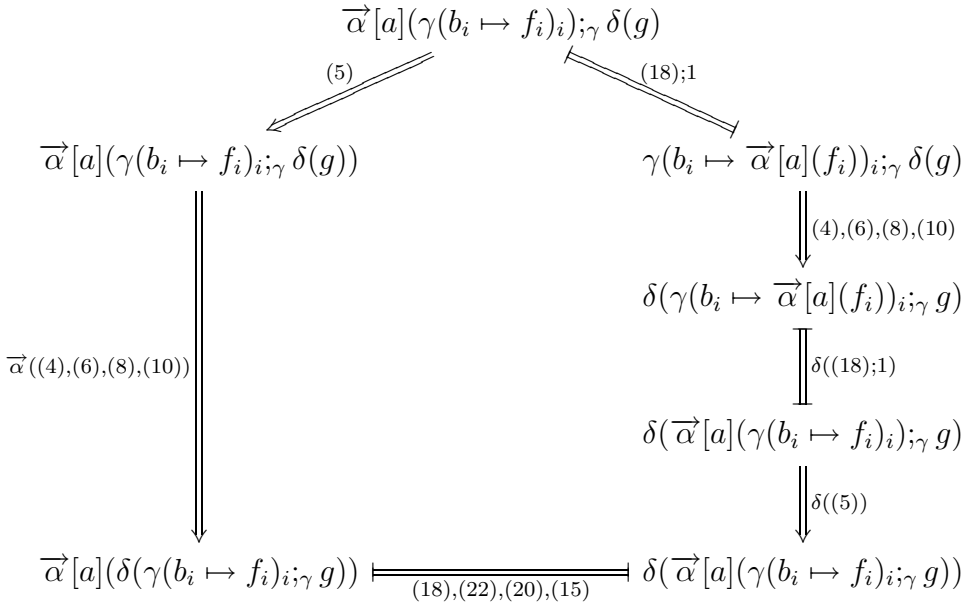


There are three subcases to consider.

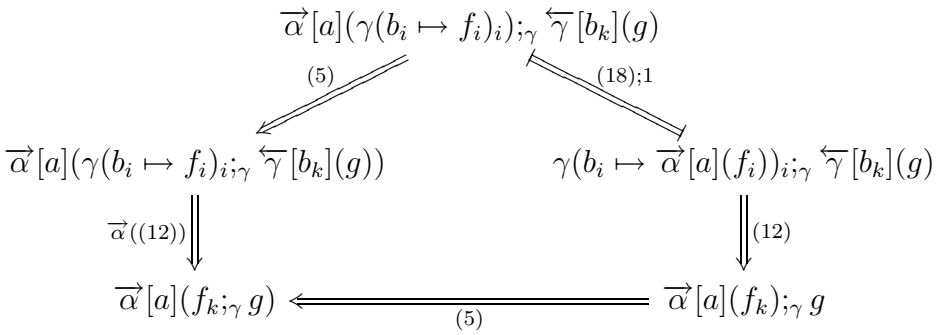
(a) $\beta \neq \gamma$.



(b) $\beta = \gamma$ and $\delta \neq \gamma$.



(c) $\beta = \gamma$ and $\delta(g) = \overleftarrow{\gamma}[b_k](g)$.

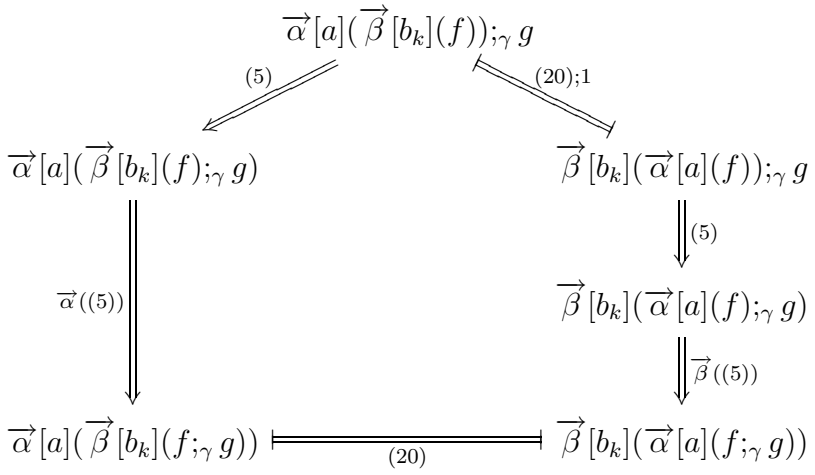


(5)-(20) [(6)-(21)]

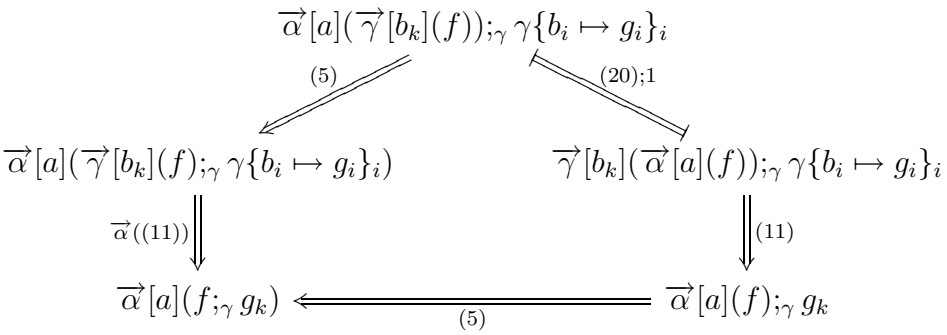
$$\overrightarrow{\alpha}[a](\overrightarrow{\beta}[b_k](f);_\gamma g) \Longleftarrow \overrightarrow{\alpha}[a](\overrightarrow{\beta}[b_k](f));_\gamma g \Longmapsto \overrightarrow{\beta}[b_k](\overrightarrow{\alpha}[a](f));_\gamma g$$

There are two subcases to consider.

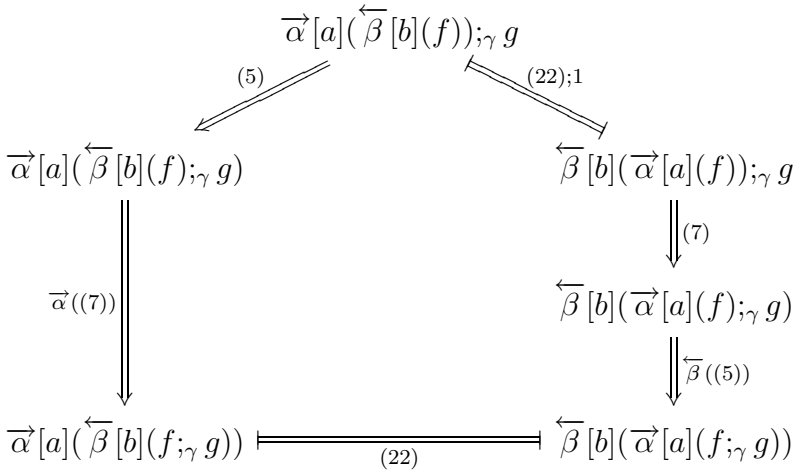
(a) $\beta \neq \gamma$.



(b) $\beta = \gamma$. Here we may assume that $g = \gamma\{b_i \mapsto g_i\}_i$.

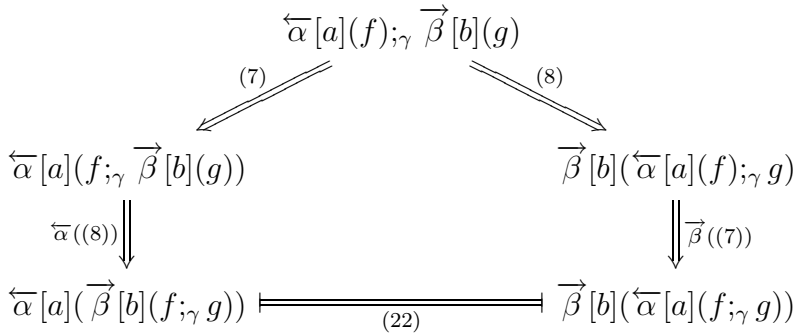


(5)-(22) [(6)-(21)]

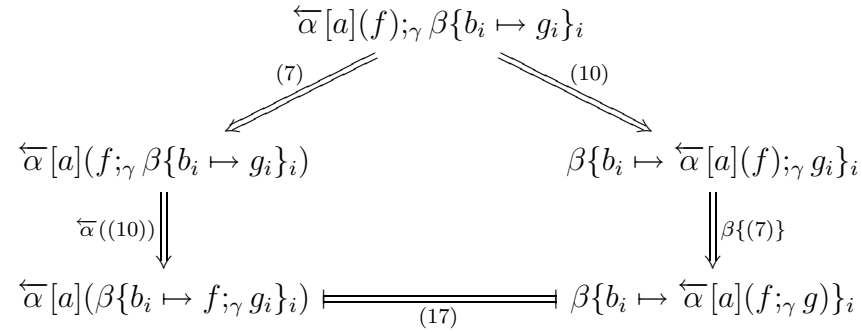


This handles all critical pairs involving (5) and (6). We now look at any critical pairs involving (7) and (8).

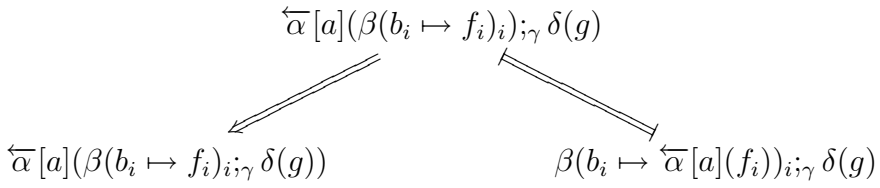
(7)-(8)



(7)-(10) [(8)-(9)]

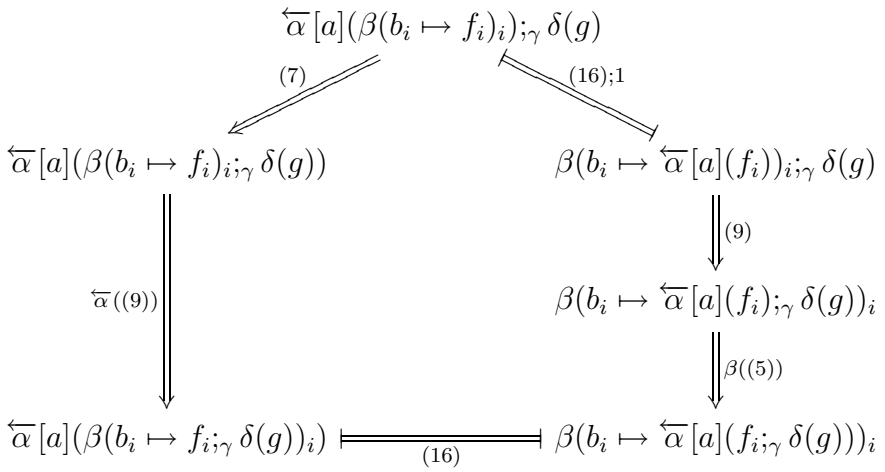


(7)-(16) [(8)-(15)]

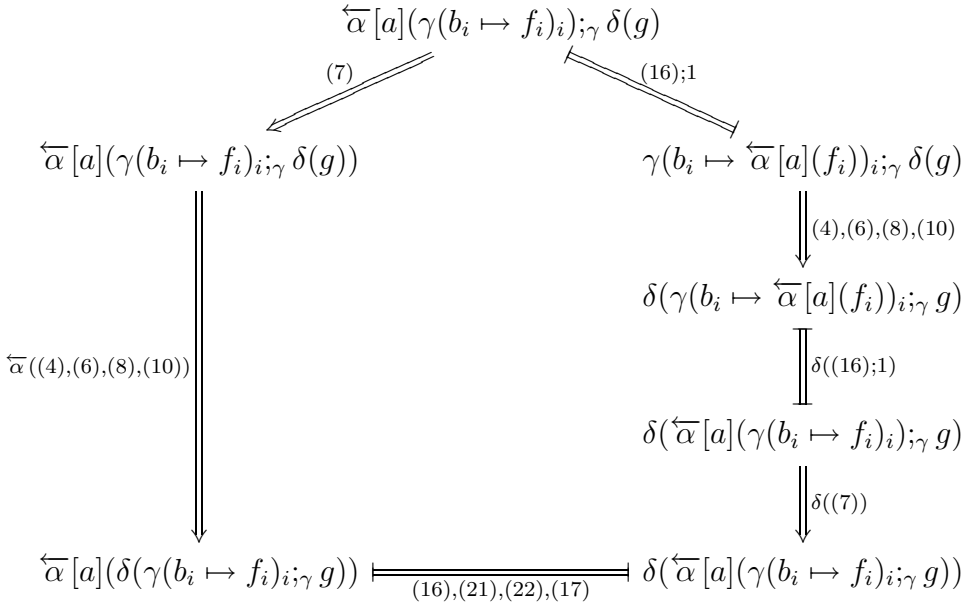


There are three subcases to consider.

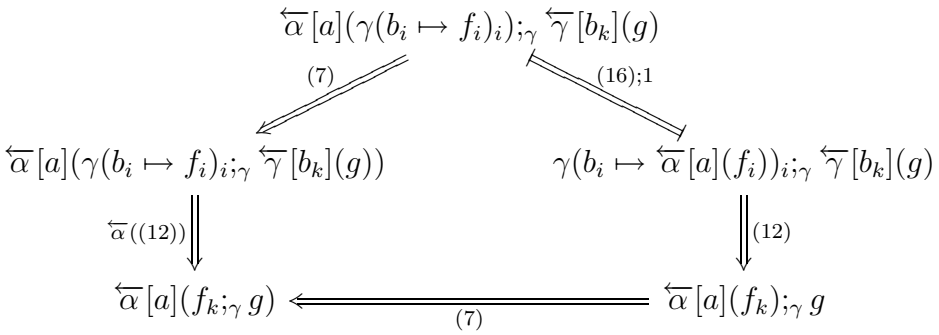
(a) $\beta \neq \gamma$



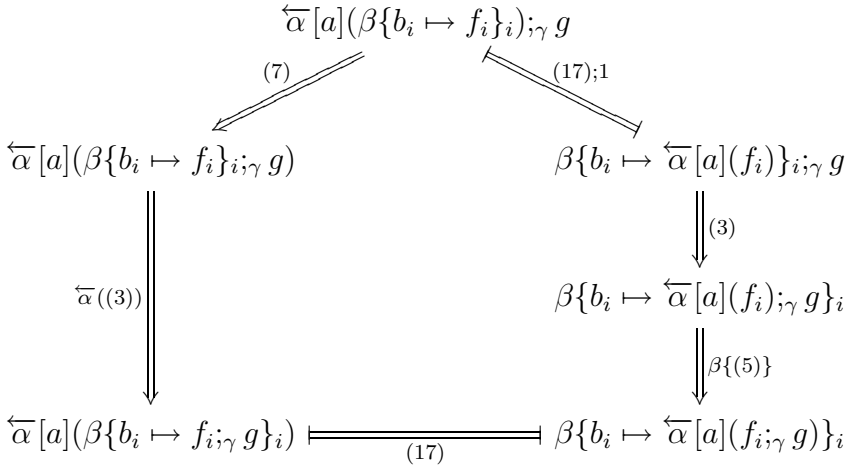
(b) $\beta = \gamma$ and $\delta \neq \gamma$.



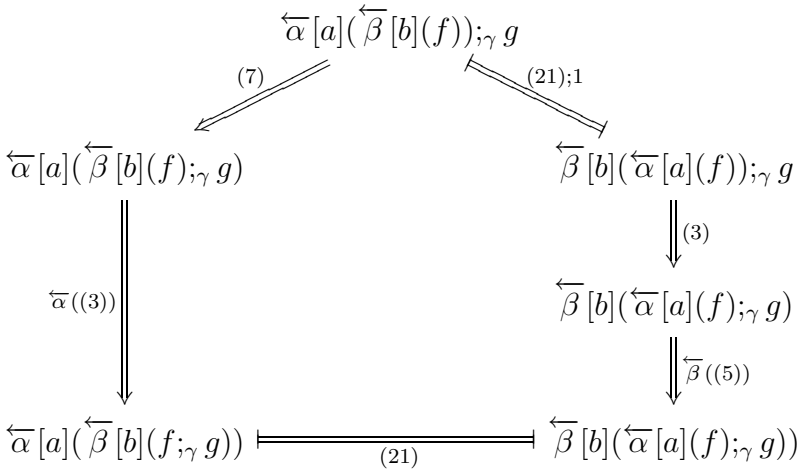
(c) $\beta = \gamma$ and $\delta(g) = \overleftarrow{\gamma}[b_k](g)$.



(7)-(17) [(8)-(18)]



(7)-(21) [(8)-(20)]

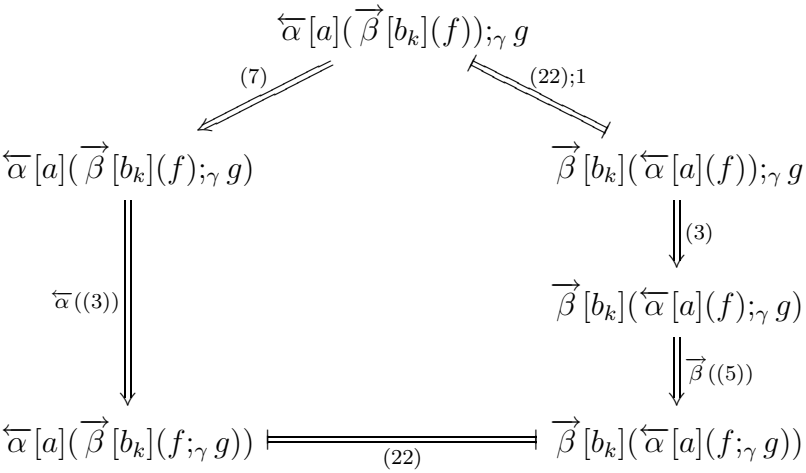


(7)-(22) [(8)-(22)]

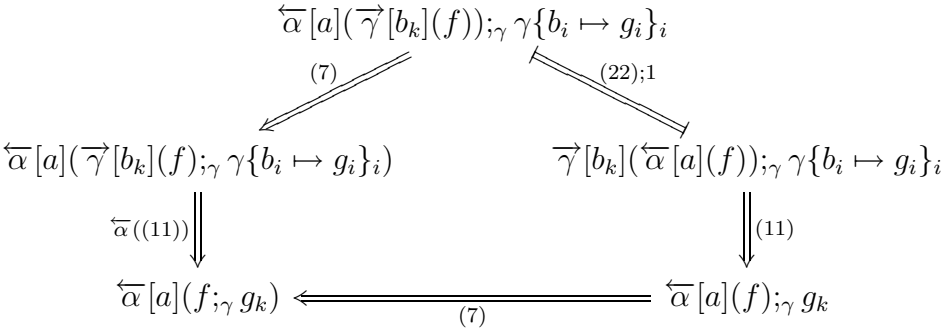
$$\overleftarrow{\alpha}[a](\overrightarrow{\beta}[b_k](f);_\gamma g) \Longleftarrow \overleftarrow{\alpha}[a](\overrightarrow{\beta}[b_k](f));_\gamma g \Longmapsto \overrightarrow{\beta}[b_k](\overleftarrow{\alpha}[a](f));_\gamma g$$

There are two subcases to consider.

(a) $\beta \neq \gamma$.

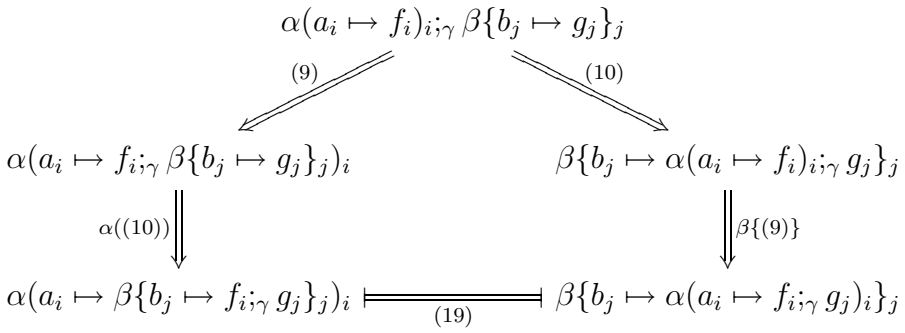


(b) $\beta = \gamma$. Here we may assume that $g = \gamma\{b_i \mapsto g_i\}_i$.

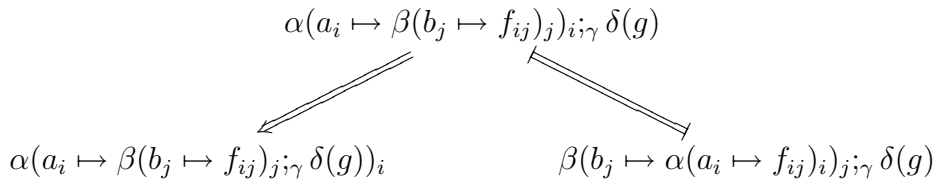


This handles all critical pairs involving (7) and (8). We move on now to look at any critical pairs involving (9) and (10).

(9)-(10)

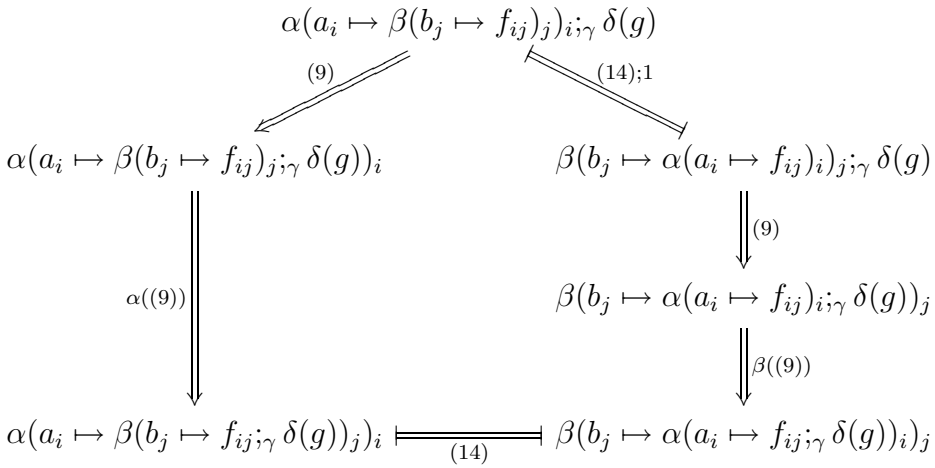


(9)-(14) [(10)-(13)]

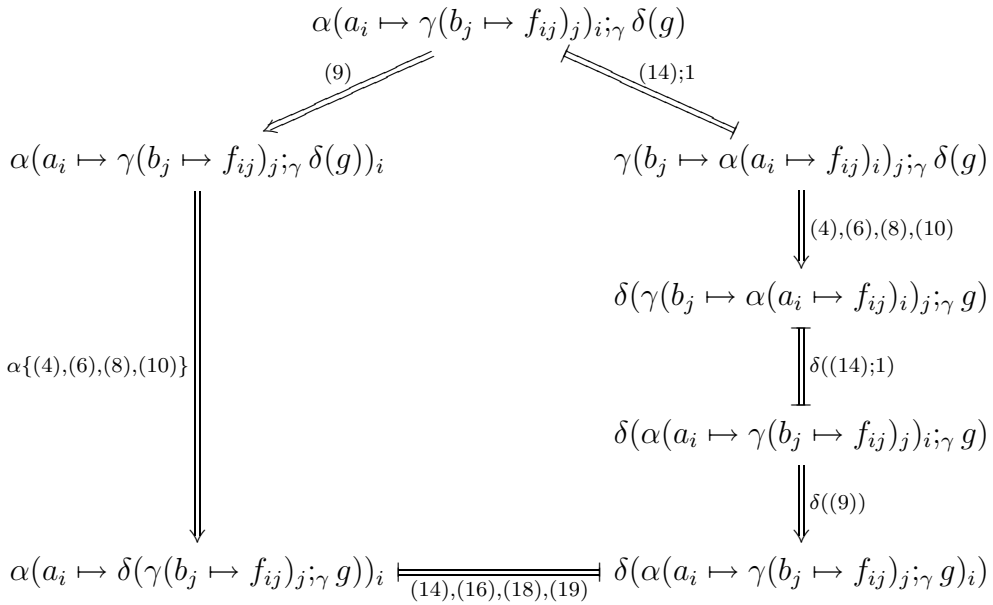


There are three subcases to consider.

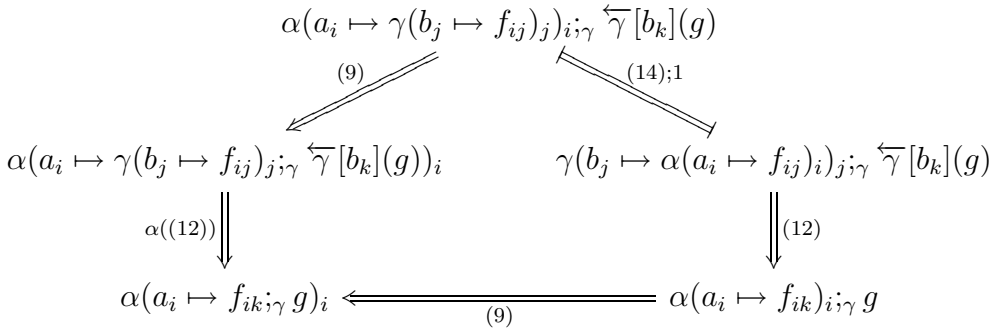
(a) $\beta \neq \gamma$.



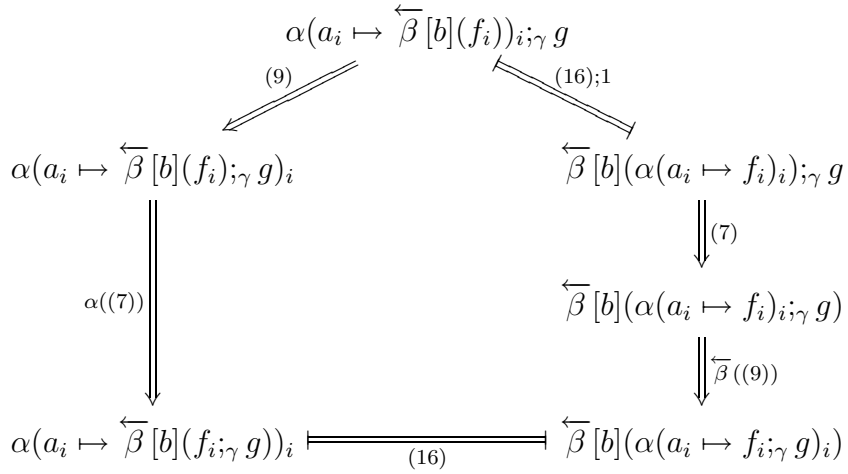
(b) $\beta = \gamma$ and $\delta \neq \gamma$.



(c) $\beta = \gamma$ and $\delta(g) = \overleftarrow{\gamma}\,[b_k](g)$



(9)-(16) [(10)-(15)]

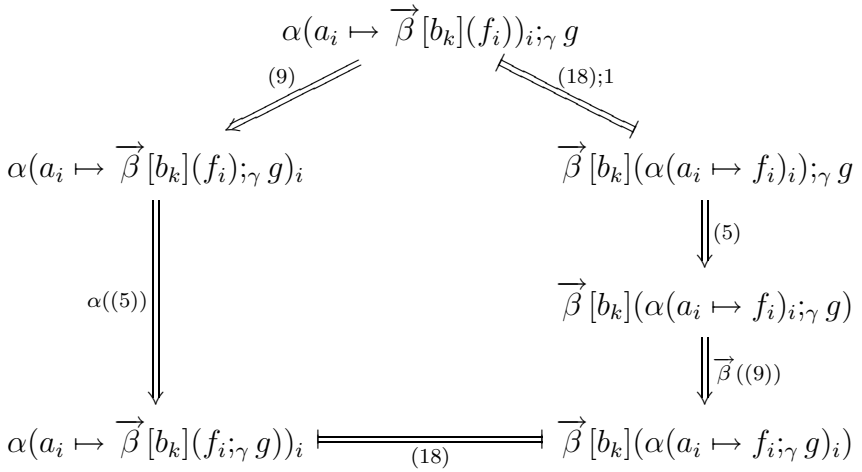


(9)-(18) [(10)-(17)]

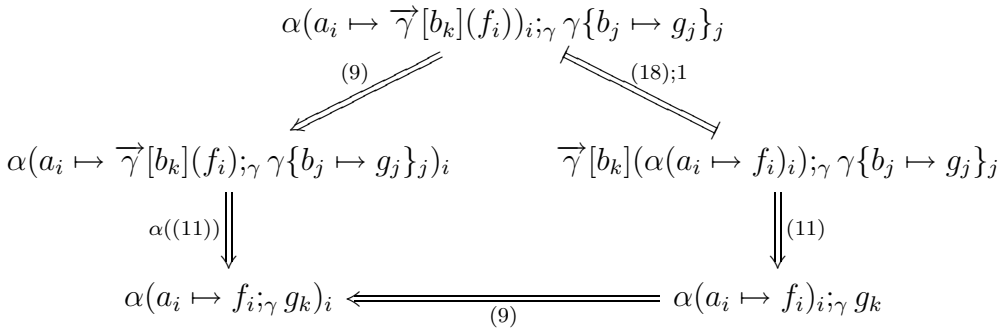
$$\alpha(a_i \mapsto \overrightarrow{\beta}\,[b_k](f_i);_\gamma g)_i \Longleftarrow \alpha(a_i \mapsto \overrightarrow{\beta}\,[b_k](f_i))_i;_\gamma g \Longmapsto \overrightarrow{\beta}\,[b_k](\alpha(a_i \mapsto f_i)_i);_\gamma g$$

There are two subcases to consider.

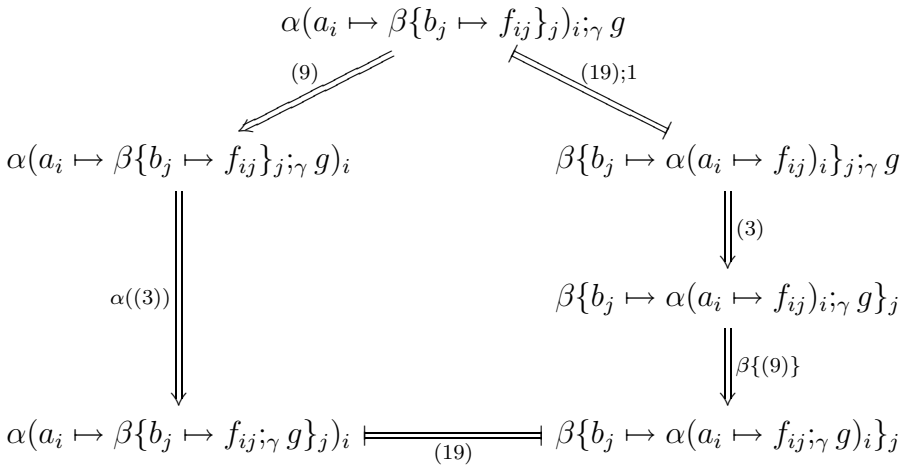
(a) $\beta \neq \gamma$.



(b) $\beta = \gamma$. Here we may assume that $g = \gamma\{b_j \mapsto g_j\}_j$.

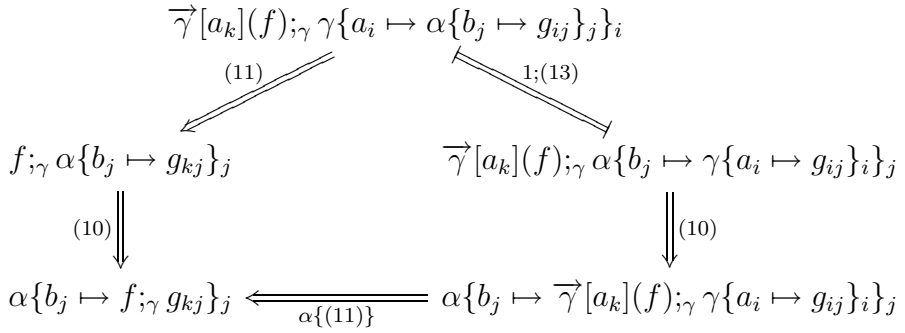


(9)-(19) [(10)-(19)]



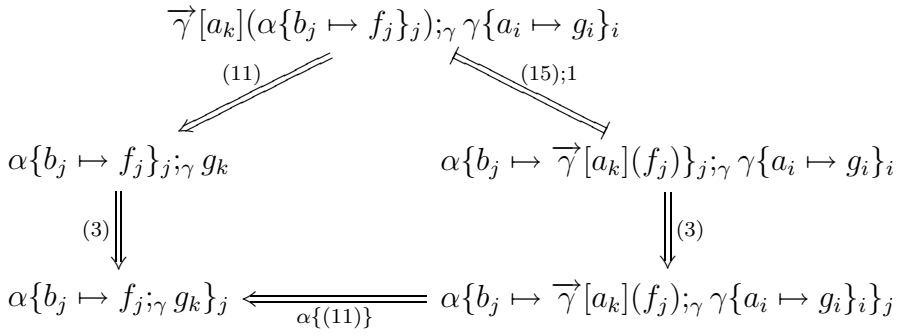
This handles all critical pairs involving (9) and (10). We move on now to look at any critical pairs involving (11) and (12).

(11)-(13) [(12)-(14)]

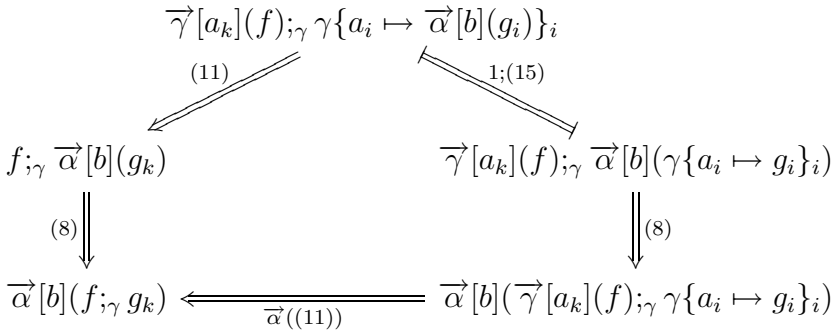


(11)-(15) [(12)-(16)] There are two subcases on the structure.

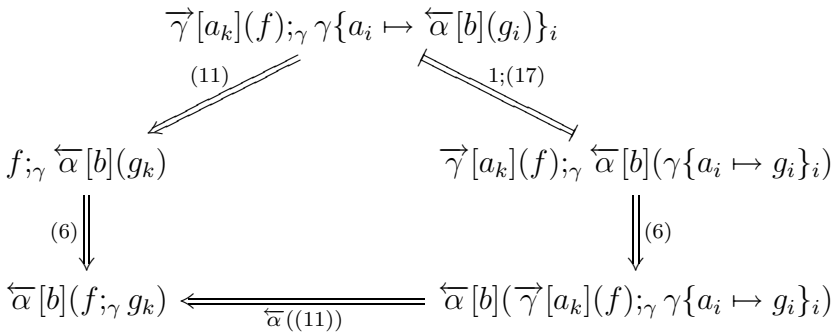
(a) $\overrightarrow{\gamma}[a_k](\alpha\{b_j \mapsto f_j\}_j);_\gamma \gamma\{a_i \mapsto g_i\}_i$



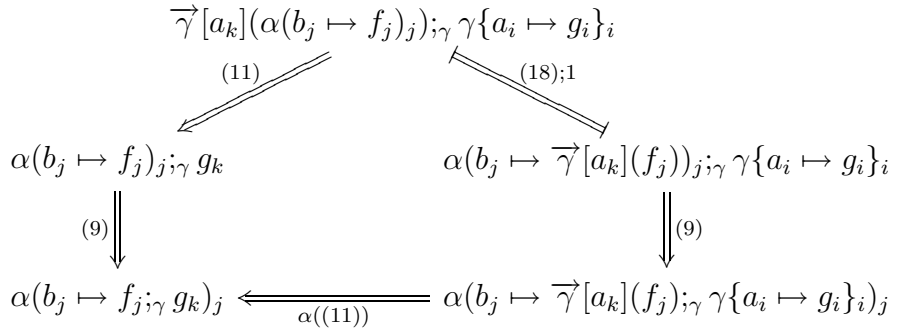
(b) $\overrightarrow{\gamma}[a_k](f);_\gamma \gamma\{a_i \mapsto \overrightarrow{\alpha}[b](g_i)\}_i$



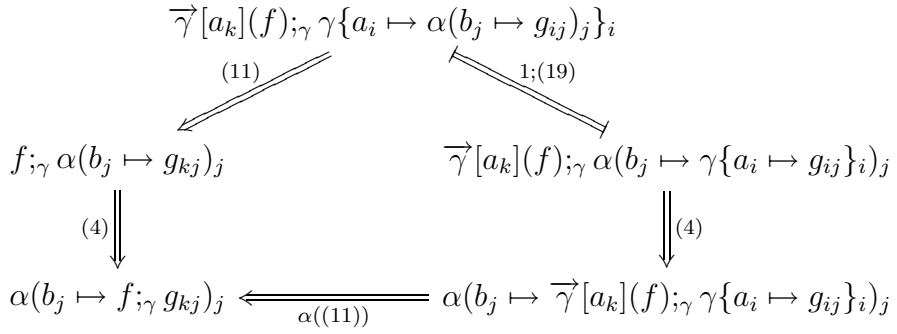
(11)-(17) [(12)-(18)]



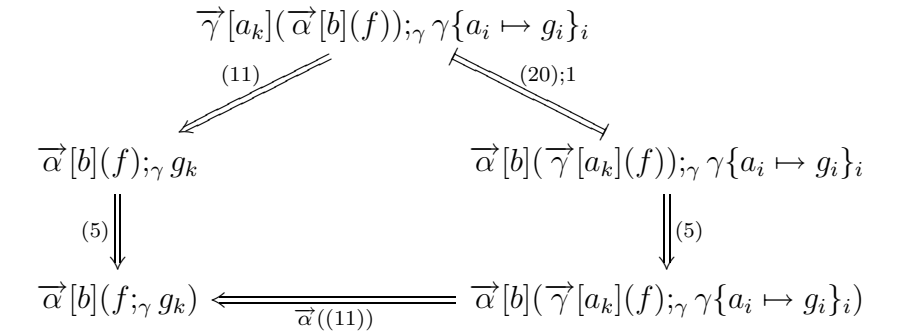
(11)-(18) [(12)-(17)]



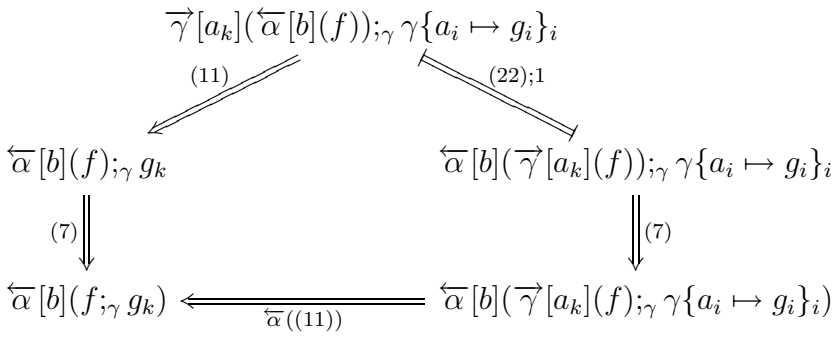
(11)-(19) [(12)-(19)]



(11)-(20) [(12)-(21)]



(11)-(22) [(12)-(22)]



This handles all critical pairs involving (11) and (12).